

Lin, Shu, 1937-

Error control coding: fundamentals and applications / Shu Lin, Daniel J. Costello.—2nd ed.
p. cm.

Includes bibliographical references and index.

ISBN 0-13-042672-5

1. Error-correcting codes (Information theory) I. Costello, Daniel J., 1942– II. Title.

QA268.L55 2003

005.7'2—dc22

2004040060

Vice President and Editorial Director, ECS: *Marcia J. Horton*Vice President and Director of Production and Manufacturing, ESM: *David W. Riccardi*Editorial Assistant: *Carole Synder*Executive Managing Editor: *Vince O'Brien*Managing Editor: *David A. George*Production Editor: *Kevin Bradley*Director of Creative Services: *Paul Belfanti*Art Director: *Jayne Conte*Cover Designer: *Bruce Kenselaar*Art Editor: *Greg Dulles*Manufacturing Manager: *Trudy Pisciotti*Manufacturing Buyer: *Lynda Castillo*Senior Marketing Manager: *Holly Stark**About the Cover:* Image courtesy of Burstein Collection/Corbis.

© 1983, 2004 by Pearson Education, Inc.

Pearson Prentice Hall

Pearson Education, Inc.

Upper Saddle River, NJ 07458

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

Pearson Prentice Hall® is a trademark of Pearson Education, Inc.

The author and publisher of this book have used their best efforts in preparing this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, expressed or implied with regard to these programs or the documentation contained in this book. The author and publisher shall not be liable in any event for incidental or consequential damages in connection with or arising out of the furnishing, performance or use of these programs.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-042672-5

Pearson Education Ltd., London

Pearson Education Australia Pty. Ltd., Sydney

Pearson Education Singapore, Pte. Ltd.

Pearson Education North Asia Ltd., Hong Kong

Pearson Education Canada, Inc., Toronto

Pearson Educación de Mexico, S.A. de C.V.

Pearson Education—Japan, Tokyo

Pearson Education Malaysia, Pte. Ltd.

Pearson Education, Inc., Upper Saddle River, New Jersey

Contents

Preface	ix
1 Coding for Reliable Digital Transmission and Storage	1
1.1 Introduction	1
1.2 Types of Codes	3
1.3 Modulation and Coding	5
1.4 Maximum Likelihood Decoding	10
1.5 Types of Errors	13
1.6 Error Control Strategies	14
1.7 Performance Measures	15
1.8 Coded Modulation	21
Bibliography	23
2 Introduction to Algebra	25
2.1 Groups	25
2.2 Fields	31
2.3 Binary Field Arithmetic	37
2.4 Construction of Galois Field $GF(2^m)$	42
2.5 Basic Properties of a Galois Field $GF(2^m)$	47
2.6 Computations Using Galois Field $GF(2^m)$ Arithmetic	54
2.7 Vector Spaces	55
2.8 Matrices	61
Problems	63
Bibliography	65
3 Linear Block Codes	66
3.1 Introduction to Linear Block Codes	66
3.2 Syndrome and Error Detection	72
3.3 The Minimum Distance of a Block Code	76
3.4 Error-Detecting and Error-Correcting Capabilities of a Block Code	78
3.5 Standard Array and Syndrome Decoding	82
3.6 Probability of an Undetected Error for Linear Codes over a BSC	90
3.7 Single-Parity-Check Codes, Repetition Codes, and Self-Dual Codes	94
Problems	95
Bibliography	97
4 Important Linear Block Codes	99
4.1 Hamming Codes	100
4.2 A Class of Single-Error-Correcting and Double-Error-Detecting Codes	102
4.3 Reed–Muller Codes	105
4.4 Other Constructions for Reed–Muller Codes	114

4.5	The Squaring Construction of Codes	119
4.6	The (24, 12) Golay Code	125
4.7	Product Codes	128
4.8	Interleaved Codes	131
	Problems	132
	Bibliography	134
5	Cyclic Codes	136
5.1	Description of Cyclic Codes	136
5.2	Generator and Parity-Check Matrices of Cyclic Codes	143
5.3	Encoding of Cyclic Codes	146
5.4	Syndrome Computation and Error Detection	150
5.5	Decoding of Cyclic Codes	155
5.6	Cyclic Hamming Codes	162
5.7	Error-Trapping Decoding	166
5.8	Improved Error-Trapping Decoding	173
5.9	The (23, 12) Golay Code	175
5.10	Shortened Cyclic Codes	179
5.11	Cyclic Product Codes	184
5.12	Quasi-Cyclic Codes	185
	Problems	188
	Bibliography	192
6	Binary BCH Codes	194
6.1	Binary Primitive BCH Codes	194
6.2	Decoding of BCH Codes	205
6.3	Iterative Algorithm for Finding the Error-Location Polynomial $\sigma(X)$	209
6.4	Simplified Iterative Algorithm for Finding the Error-Location Polynomial $\sigma(X)$	212
6.5	Finding the Error-Location Numbers and Error Correction	215
6.6	Correction of Errors and Erasures	217
6.7	Implementation of Galois Field Arithmetic	217
6.8	Implementation of Error Correction	224
6.9	Weight Distribution and Error Detection of Binary BCH Codes	227
6.10	Remarks	230
	Problems	230
	Bibliography	231
7	Nonbinary BCH Codes, Reed–Solomon Codes, and Decoding Algorithms	234
7.1	q -ary Linear Block Codes	234
7.2	Primitive BCH Codes over $GF(q)$	236
7.3	Reed–Solomon Codes	237
7.4	Decoding of Nonbinary BCH and RS Codes: The Berlekamp Algorithm	241
7.5	Decoding with the Euclidean Algorithm	248
7.6	Frequency-Domain Decoding	255
7.7	Correction of Errors and Erasures	263

Problems	269
Bibliography	270
8 Majority-Logic Decodable and Finite Geometry Codes	273
8.1 One-Step Majority-Logic Decoding	273
8.2 A Class of One-Step Majority-Logic Decodable Codes	282
8.3 Other One-Step Majority-Logic Decodable Codes	290
8.4 Multiple-Step Majority-Logic Decoding	296
8.5 Euclidean Geometry	304
8.6 Euclidean Geometry Codes	309
8.7 Twofold EG Codes	319
8.8 Projective Geometry and Projective Geometry Codes	325
8.9 Remarks	331
Problems	332
Bibliography	335
9 Trellises for Linear Block Codes	338
9.1 Finite-State Machine Model and Trellis Representation of a Code	338
9.2 Bit-Level Trellises for Binary Linear Block Codes	342
9.3 State Labeling	351
9.4 Structural Properties of Bit-Level Trellises	354
9.5 State Labeling and Trellis Construction Based on the Parity-Check Matrix	360
9.6 Trellis Complexity and Symmetry	367
9.7 Trellis Sectionalization and Parallel Decomposition	374
9.8 Low-Weight Subtrellises	380
9.9 Cartesian Product	382
Problems	390
Bibliography	391
10 Reliability-Based Soft-Decision Decoding Algorithms for Linear Block Codes (<i>Contributed by Marc P. C. Fossorier</i>)	395
10.1 Soft-Decision Decoding	395
10.2 Reliability Measures and General Reliability-Based Decoding Schemes	400
10.3 Sufficient Conditions on the Optimality of a Decoded Codeword	402
10.4 Generalized Minimum Distance and Chase Decoding Algorithms	407
10.5 Weighted Erasure Decoding	413
10.6 A [*] Maximum Likelihood Decoding Algorithm Based on Iterative Processing of the Least Reliable Positions	417
10.7 Reduced List Syndrome Decoding Algorithm	419
10.8 Most Reliable Independent Position Reprocessing Decoding Algorithms	422
10.9 Weighted Majority-Logic Decoding	439
10.10 Iterative Reliability-Based Decoding of One-Step Majority-Logic Decodable Codes	442

Problems	447
Bibliography	448
11 Convolutional Codes	453
11.1 Encoding of Convolutional Codes	454
11.2 Structural Properties of Convolutional Codes	486
11.3 Distance Properties of Convolutional Codes	506
Problems	510
Bibliography	513
12 Optimum Decoding of Convolutional Codes	515
12.1 The Viterbi Algorithm	516
12.2 Performance Bounds for Convolutional Codes	525
12.3 Construction of Good Convolutional Codes	538
12.4 Implementation and Performance of the Viterbi Algorithm	544
12.5 The Soft-Output Viterbi Algorithm (SOVA)	558
12.6 The BCJR algorithm	563
12.7 Punctured and Tail-Biting Convolutional Codes	582
Problems	598
Bibliography	602
13 Suboptimum Decoding of Convolutional Codes	605
13.1 The ZJ (Stack) Sequential Decoding Algorithm	606
13.2 The Fano Sequential Decoding Algorithm	620
13.3 Performance Characteristics of Sequential Decoding	626
13.4 Code Construction for Sequential Decoding	640
13.5 Majority-Logic Decoding	645
13.6 Performance Characteristics of Majority-Logic Decoding	670
13.7 Code Construction for Majority-Logic Decoding	677
Problems	685
Bibliography	688
14 Trellis-Based Soft-Decision Decoding Algorithms	691
14.1 The Viterbi Decoding Algorithm	691
14.2 A Recursive Maximum Likelihood Decoding Algorithm	695
14.3 A Suboptimum Iterative Decoding Algorithm Based on a Low-Weight Subtrellis	704
14.4 The MAP Decoding Algorithm	711
14.5 MAP Decoding Based on a Sectionalized Trellis	718
14.6 Max-log-MAP Decoding Algorithm	726
Problems	734
Bibliography	735
15 Concatenated Coding, Code Decomposition, and Multistage Decoding	739
15.1 Single-Level Concatenated Codes	739
15.2 Multilevel Concatenated Codes	743
15.3 A Soft-Decision Multistage Decoding	748
15.4 Decomposition of Codes	750
15.5 An Iterative Multistage MLD Algorithm	754

15.6	Concatenated Coding Schemes with Convolutional Inner Codes	760
15.7	Binary Concatenation	761
	Problems	763
	Bibliography	764
16	Turbo Coding	766
16.1	Introduction to Turbo Coding	767
16.2	Distance Properties of Turbo Codes	783
16.3	Performance Analysis of Turbo Codes	807
16.4	Design of Turbo Codes	814
16.5	Iterative Decoding of Turbo Codes	826
	Problems	844
	Bibliography	847
17	Low-Density Parity-Check Codes	851
17.1	Introduction to LDPC Codes	852
17.2	Tanner Graphs for Linear Block Codes	855
17.3	A Geometric Construction of LDPC Codes	858
17.4	EG-LDPC Codes	860
17.5	PG-LDPC Codes	866
17.6	Decoding of LDPC Codes	871
17.7	Code Construction by Column and Row Splitting	885
17.8	Breaking Cycles in Tanner Graphs	892
17.9	Shortened Finite-Geometry LDPC Codes	898
17.10	Construction of Gallager LDPC Codes	902
17.11	Masked EG-Gallager LDPC Codes	906
17.12	Construction of Quasi-Cyclic Codes by Circulant Decomposition	912
17.13	Construction of LDPC Codes Based on Finite Geometries over $GF(p^s)$	917
17.14	Random LDPC Codes	920
17.15	Irregular LDPC Codes	922
17.16	Graph-Theoretic LDPC Codes	929
17.17	Construction of LDPC Codes Based on Balanced Incomplete Block Designs	935
17.18	Construction of LDPC Codes Based on Shortened RS Codes with Two Information Symbols	938
17.19	Concatenations with LDPC and Turbo Codes	944
	Problems	945
	Bibliography	947
18	Trellis-Coded Modulation	952
18.1	Introduction to Trellis-Coded Modulation	953
18.2	TCM Code Construction	980
18.3	TCM Performance Analysis	992
18.4	Rotationally Invariant TCM	998
18.5	Multidimensional TCM	1015

Problems	1056
Bibliography	1059
19 Block Coded Modulation	1063
19.1 Distance Concepts	1063
19.2 Multilevel Block Modulation Codes	1064
19.3 Multistage Decoding of Multilevel BCM Codes	1075
19.4 Concatenated Coded Modulation	1081
19.5 Product Coded Modulation	1088
19.6 Multilevel Coded Modulation for Unequal Error Protection	1090
Problems	1100
Bibliography	1101
20 Burst-Error-Correcting Codes	1104
20.1 Introduction	1104
20.2 Decoding of Single-Burst-Error-Correcting Cyclic Codes	1105
20.3 Single-Burst-Error-Correcting Codes	1107
20.4 Phased-Burst-Error-Correcting Codes	1118
20.5 Burst-and-Random-Error-Correcting Codes	1119
Problems	1124
Bibliography	1125
21 Burst-Error-Correcting Convolutional Codes	1127
21.1 Bounds on Burst-Error-Correcting Capability	1127
21.2 Burst-Error-Correcting Convolutional Codes	1128
21.3 Interleaved Convolutional Codes	1139
21.4 Burst-and-Random-Error-Correcting Convolutional Codes	1142
Problems	1153
Bibliography	1154
22 Automatic-Repeat-Request Strategies	1156
22.1 Basic ARQ Schemes	1156
22.2 Selective-Repeat ARQ System with Finite Receiver Buffer	1163
22.3 ARQ Schemes with Mixed Modes of Retransmission	1171
22.4 Hybrid ARQ Schemes	1174
22.5 A Class of Half-Rate Invertible Codes	1178
22.6 Type-II Hybrid Selective-Repeat ARQ with Finite Receiver Buffer	1181
22.7 Hybrid ARQ Systems Using Convolutional Codes	1190
22.8 A Concatenated Coded Modulation Hybrid ARQ System	1192
Problems	1197
Bibliography	1198
A Tables of Galois Fields	1204
B Minimal Polynomials of Elements in $GF(2^m)$	1227
C Generator Polynomials of Binary Primitive BCH Codes of Length up to $2^{10} - 1$	1231
Index	1249

Preface

This book owes its beginnings to the pioneering work of Claude Shannon in 1948 on reliable communication over noisy transmission channels. Shannon's central theme was that if the signaling rate of the system is less than the channel capacity, reliable communication can be achieved if one chooses proper encoding and decoding techniques. The design of good codes and of efficient decoding methods, initiated by Hamming, Golay, and others in the late 1940s, has since occupied the energies of many researchers. Much of this work is highly mathematical in nature, and a thorough understanding requires an extensive background in modern algebra and probability theory. This requirement has impeded many engineers and computer scientists who are interested in applying these techniques to practical systems. One of the purposes of this book is to present the essentials of this highly complex material in such a manner that it can be understood and applied with only a minimum of mathematical background.

Coding research in the 1950s and 1960s was devoted primarily to developing the theory of efficient encoders and decoders. In 1970 the first author published a book entitled *An Introduction to Error-Correcting Codes*, which presented the fundamentals of the previous two decades of work covering both block and convolutional codes. The approach was to explain the material in an easily understood manner, with a minimum of mathematical rigor. Then, in 1983, the authors published the first edition of this book. The approach again stressed the fundamentals of coding. In addition, new material on many of the practical applications of coding developed during the 1970s was introduced. Other major additions included a comprehensive treatment of the error-detecting capabilities of block codes and an emphasis on soft decoding methods for convolutional codes.

In the 1980s and 1990s, the coding field exploded with new theoretical developments, several of which have had significant practical consequences. Three of these new developments stand out in particular: the application of binary convolutional and block codes to expanded (nonbinary) modulation alphabets, the development of practical soft decoding methods for block codes, and the discovery of soft-input, soft-output iterative decoding techniques for block and convolutional codes. These new developments have revolutionized the way coding is applied to practical systems, affecting the design of high-speed data modems, digital mobile cellular telephony, satellite and space communications, and high-density data storage, to name just a few. A total of seven new chapters covering these topics have been added to this edition: two chapters on trellis- and block-coded modulation techniques, three chapters on soft decoding methods for block codes, and two chapters on turbo and low-density parity-check codes and iterative decoding.

Because of the significant amount of new material and the need to maintain an emphasis on coding fundamentals, it was not possible to include certain topics in this edition. For example, the new developments in algebraic geometry codes and erasure correcting codes are not covered. Also, although the codes developed in the book can be applied to data storage systems, the specific peculiarities of the storage channel are not directly addressed. Similarly, it was not possible to give a comprehensive treatment of coding for fading channels. In addition to the new

chapters noted, all the chapters in the first edition have been thoroughly revised and updated in the second edition. A brief description of each chapter follows, highlighting changes from the first edition.

Chapter 1 presents an overview of coding for error control in data transmission and storage systems. A brief discussion of modulation and demodulation serves to place coding in the context of a complete system. Two new sections, introducing the concepts of coded modulation, coding gain, and the Shannon limit, have been added. Chapter 2 develops those concepts from modern algebra that are necessary to understand the material in later chapters. The presentation is at a level that can be understood by students in the senior year as well as by practicing engineers and computer scientists.

Chapters 3 through 10 cover in detail the fundamentals of block codes. Linear block codes are presented in Chapter 3. Included is material on the error detection capability of linear codes. Several important classes of linear codes are introduced in Chapter 4. New material on Reed–Muller codes has been added to this chapter. The basic structure and properties of cyclic codes are presented in Chapter 5, and syndrome-based decoding methods are introduced. The important class of BCH codes is presented in detail in Chapter 6. A discussion of hardware and software implementation of BCH decoders is included, as well as the use of BCH codes for error detection. Chapter 7 includes an expanded coverage of Reed–Solomon codes. New material on the Euclidean algorithm and frequency-domain decoding has been added. Chapter 8 provides detailed coverage of majority-logic decodable codes, including the important classes of Euclidean and projective geometry codes. Chapters 9 and 10 are both completely new. Chapter 9 develops the theory of the trellis structure of block codes, laying the groundwork for the introduction of trellis-based soft decoding algorithms in Chapter 14. Chapter 10, written by Professor Marc Fossorier, presents comprehensive coverage of reliability-based soft decoding methods for block codes and includes an introduction to iterative decoding techniques.

Chapters 11 through 13 are devoted to the presentation of the fundamentals of convolutional codes. Convolutional codes are introduced in Chapter 11, with the encoder state diagram serving as the basis for studying code structure and distance properties. New material on feedback encoders and input–output weight enumerating functions has been added. Chapter 12 covers optimum decoding methods for convolutional codes, with an emphasis on the (maximum likelihood) Viterbi decoding algorithm for both hard and soft demodulator decisions. New material has been added on the soft-output Viterbi algorithm, the (maximum a posteriori probability) BCJR algorithm, and the code modification techniques of puncturing and tail-biting. A detailed performance analysis based on encoder weight enumerating functions is also included. Chapter 13 covers suboptimum decoding methods for convolutional codes, with an emphasis on sequential decoding, using both the ZJ (stack) and Fano algorithms, and majority-logic decoding. The analytically difficult problem of the computational performance of sequential decoding is discussed without including detailed proofs, and new material on soft-decision versions of sequential and majority-logic decoding has been added.

Chapter 14 extends the soft decoding methods introduced for convolutional codes in Chapter 12 to block codes. This completely new chapter makes extensive use of the block code trellis structures introduced in Chapter 9.

Chapters 15 through 19 cover the important advances in the field since the publication of the first edition. Chapter 15 discusses the important concepts of code concatenation, multistage decoding, and code decomposition. These topics lay the groundwork for the new coding techniques presented in the next four chapters. Chapters 16 through 19 are completely new. Chapter 16 introduces the area of parallel concatenation, or turbo coding, and its related iterative decoding techniques based on the BCJR algorithm presented in Chapter 12. Performance analysis based on the uniform interleaver technique and the EXIT chart concept is included. Chapter 17 presents a thorough coverage of low-density parity-check codes based on algebraic, random, and combinatoric construction methods. Several decoding methods are discussed, and a complete development of soft-decision belief propagation decoding is included. The area of coded modulation is covered in Chapters 18 and 19. The fundamentals of trellis-coded modulation are presented in Chapter 18. Sections on rotationally invariant codes and multidimensional signal sets are included. Block-coded modulation is covered in Chapter 19. The important concepts of multilevel modulation and multistage decoding are included.

The book concludes with three chapters on burst-error correction and automatic-repeat-request (ARQ) strategies. Chapters 20 and 21 cover methods for correcting the burst errors and combinations of burst and random errors commonly encountered on fading channels. Both block (Chapter 20) and convolutional (Chapter 21) burst-error-correcting codes are included. Chapter 22 is devoted to the ARQ error control schemes used on two-way communication channels. Both pure ARQ (error detection with retransmission) and hybrid ARQ (a combination of error correction and error detection with retransmission) are discussed.

Several additional features make the book useful both as a classroom text and as a comprehensive reference for engineers and computer scientists involved in the design of error control systems. Three appendices include the algebraic background used in the construction of most block codes. Many tables of the best known block and convolutional codes for various decoding methods are presented throughout the book. These tables can be used by system designers to select the best code for a given application. In this edition, a consistent octal notation has been adopted for the generator and parity-check polynomials that define the codes listed in these tables. Many examples of codes used in practical applications and computer-simulated performance curves for specific coding systems are also included. A set of homework problems is given at the end of each chapter. Most of these problems are relatively straightforward applications of material covered in the text, although some more advanced problems are included. Instructors can obtain solutions to selected problems from the publisher. References are also given at the end of each chapter. Although no attempt was made to compile a complete bibliography on coding, the references listed serve to provide additional detail on topics covered in the book.

The book can be used as a text for an introductory course on coding at the senior or beginning graduate level or a more comprehensive full-year graduate course. It also can be used as a self-study guide for engineers and computer scientists

in industry who want to learn the fundamentals of coding and how they can be applied to the design of error control systems.

As a text the book can be used as the basis for a two-semester sequence in coding theory, with Chapters 1–10 on the fundamentals of block codes covered in one semester and the remaining chapters on convolutional codes and advanced block code topics in a second semester. Another possibility is to cover Chapters 1–8 and 11–13, which include the fundamentals of both block and convolutional codes, in one semester, followed by a second semester devoted to advanced topics. Alternatively portions of the book can be covered in a one-semester course. A course on block codes comprise Chapters 1–7 plus selected topics from Chapters 8–10, 14–15, 17, 19–20, and 22, whereas Chapters 1, 11–13, 16, 18, and 21 provide a thorough coverage of convolutional codes.

We would like to express our sincere appreciation to Professor Marc Fossorier, who, in addition to writing Chapter 10, spent many long hours reading and rereading drafts of various chapters. We also wish to thank the many graduate students and postdoctoral associates who offered their comments and helped in the preparation of the book, including running computer simulations, drafting figures, compiling tables, and converting the manuscript to LaTeX format. These persons include Yu Kou, Cathy Liu, Rose Shao, Diana Stojanovic, Jun Xu, Lei Chen, Oscar Takeshita, Gil Shamir, Adrish Banerjee, Arvind Sridharan, Ching He, Wei Zhang, and Ali Pusane. In particular, Yu Kou, Cathy Liu, and Adrish Banerjee deserve special mention for overseeing the preparation of the final version of the manuscript.

We are grateful to the National Science Foundation and to the National Aeronautics and Space Administration for their continuing support of our research in the coding field. Without their assistance, our interest in coding could never have developed to the point of writing this book. We also thank the University of Hawaii, Manoa, the University of California, Davis, the University of Notre Dame, and the Humboldt Foundation of Germany for their support of our efforts in writing this book and for providing facilities.

Finally, we would like to give special thanks to our wives, children, and grandchildren for their continuing love and affection throughout this project.

SHU LIN
University of California, Davis
University of Hawaii, Manoa

DANIEL J. COSTELLO, JR.
University of Notre Dame

CHAPTER 1

Coding for Reliable Digital Transmission and Storage

1.1 INTRODUCTION

In recent years, there has been an increasing demand for efficient and reliable digital data transmission and storage systems. This demand has been accelerated by the emergence of large-scale, high-speed data networks for the exchange, processing, and storage of digital information in the commercial, governmental, and military spheres. A merging of communications and computer technology is required in the design of these systems. A major concern of the system designer is the control of errors so that the data can be reliably reproduced.

In 1948, Shannon [1] demonstrated in a landmark paper that, by proper encoding of the information, errors induced by a noisy channel or storage medium can be reduced to any desired level without sacrificing the rate of information transmission or storage, as long as the information rate is less than the capacity of the channel. Since Shannon's work much effort has been expended on the problem of devising efficient encoding and decoding methods for error control in a noisy environment. Recent developments have contributed toward achieving the reliability required by today's high-speed digital systems, and the use of coding for error control has become an integral part in the design of modern communication and digital storage systems.

The transmission and storage of digital information have much in common. Both processes transfer data from an information source to a destination (or user). A typical transmission (or storage) system may be represented by the block diagram shown in Figure 1.1. The *information source* can be either a person or a machine—for example, a digital computer, or a data terminal. The source output, which is to be communicated to the destination, can be either a continuous waveform or a sequence of discrete symbols. The *source encoder* transforms the source output into a sequence of binary digits (bits) called the *information sequence \mathbf{u}* . In the case of a continuous source, this process involves analog-to-digital (A/D) conversion. The source encoder is ideally designed so that (1) the number of bits per unit time required to represent the source output is minimized; and (2) the source output can be unambiguously reconstructed from the information sequence \mathbf{u} . The subject of source coding is not discussed in this book. For a thorough treatment of this important topic, see references [2], [3], [4] and [5].

The *channel encoder* transforms the information sequence \mathbf{u} into a discrete *encoded sequence \mathbf{v}* called a *codeword*. In most instances \mathbf{v} is also a binary sequence, although in some applications nonbinary codes have been used. The design and implementation of channel encoders to combat the noisy environment in which codewords must be transmitted or stored is one of the major topics of this book.

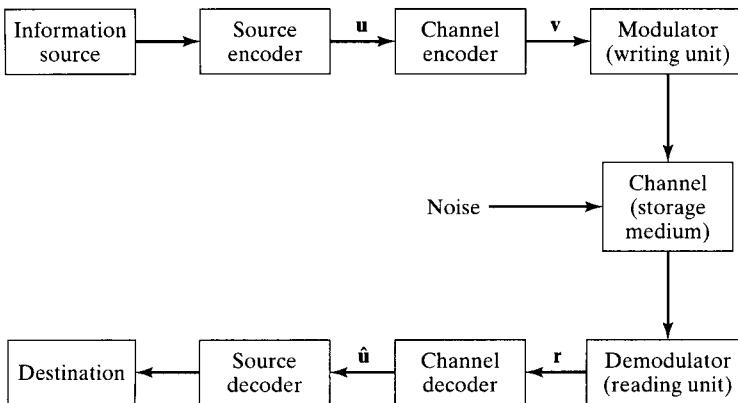


FIGURE 1.1: Block diagram of a typical data transmission or storage system.

Discrete symbols are not suitable for transmission over a physical channel or recording on a digital storage medium. The *modulator* (or *writing unit*) transforms each output symbol of the channel encoder into a waveform of duration T seconds that is suitable for transmission (or recording). This waveform enters the *channel* (or *storage medium*) and is corrupted by noise. Typical transmission channels include telephone lines, mobile cellular telephony, high-frequency (HF) radio, telemetry, microwave and satellite links, optical fiber cables, and so on. Typical storage media include core and semiconductor memories, magnetic tapes, drums and discs, compact discs, optical memory units, and so on. Each of these examples is subject to various types of noise disturbances. On a telephone line, the disturbance may come from switching impulse noise, thermal noise, or crosstalk from other lines. On magnetic discs (or compact discs), surface defects and dust particles are regarded as noise disturbances. The *demodulator* (or *reading unit*) processes each received waveform of duration T and produces either a discrete (quantized) or a continuous (unquantized) output. The sequence of demodulator outputs corresponding to the encoded sequence v is called the *received sequence r*.

The *channel decoder* transforms the received sequence r into a binary sequence \hat{u} called the *estimated information sequence*. The decoding strategy is based on the rules of channel encoding and the noise characteristics of the channel (or storage medium). Ideally, \hat{u} will be a replica of the information sequence u , although the noise may cause some *decoding errors*. Another major topic of this book is the design and implementation of channel decoders that minimize the probability of decoding error.

The *source decoder* transforms the *estimated information sequence* \hat{u} into an *estimate* of the source output and delivers this estimate to the *destination*. When the source is continuous, this process involves digital-to-analog (D/A) conversion. In a well-designed system, the estimate will be a faithful reproduction of the source output except when the channel (or storage medium) is very noisy.

To focus attention on the channel encoder and channel decoder, (1) the information source and source encoder can be combined into a *digital source* with output u ; (2) the modulator (or writing unit), the channel (or storage medium), and

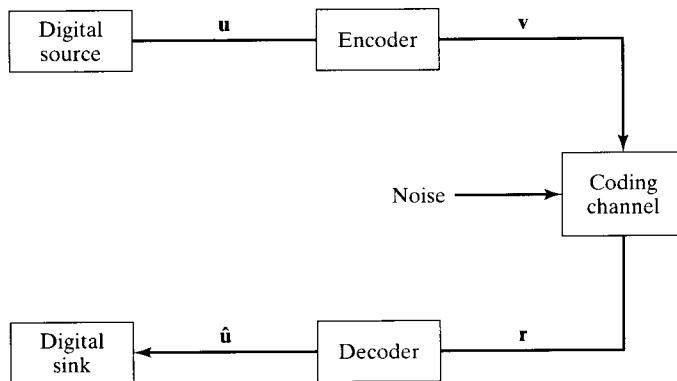


FIGURE 1.2: Simplified model of a coded system.

the demodulator (or reading unit) can be combined into a *coding channel* with input v and output r ; and (3) the source decoder and destination can be combined into a *digital sink* with input \hat{u} . These combinations result in the simplified block diagram shown in Figure 1.2.

The major engineering problem that is addressed in this book is to design and implement the channel encoder/decoder pair such that (1) information can be transmitted (or recorded) in a noisy environment as fast (or as densely) as possible; (2) the information can be reliably reproduced at the output of the channel decoder; and (3) the cost of implementing the encoder and decoder falls within acceptable limits.

1.2 TYPES OF CODES

Two structurally different types of codes are in common use today: *block codes* and *convolutional codes*. The encoder for a block code divides the information sequence into message blocks of k information bits (symbols) each. A message block is represented by the binary k -tuple $\mathbf{u} = (u_0, u_1, \dots, u_{k-1})$, called a *message*. (In block coding, the symbol \mathbf{u} is used to denote a k -bit message rather than the entire information sequence.) There are a total of 2^k different possible messages. The encoder transforms each message \mathbf{u} independently into an n -tuple $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ of discrete symbols, called a *codeword*. (In block coding, the symbol \mathbf{v} is used to denote an n -symbol block rather than the entire encoded sequence.) Therefore, corresponding to the 2^k different possible messages, there are 2^k different possible codewords at the encoder output. This set of 2^k codewords of length n is called an (n, k) *block code*. The ratio $R = k/n$ is called the *code rate*, and it can be interpreted as the number of information bits entering the encoder per transmitted symbol. Because the n -symbol output codeword depends only on the corresponding k -bit input message; that is, each message is encoded independently, the encoder is memoryless and can be implemented with a combinational logic circuit.

In a binary code, each codeword \mathbf{v} is also binary. Hence, for a binary code to be useful; that is, to have a different codeword assigned to each message, $k \leq n$, or $R \leq 1$. When $k < n$, $n - k$ redundant bits are added to each message to form a

TABLE 1.1: A binary block code
with $k = 4$ and $n = 7$.

Messages	Codewords
(0 0 0 0)	(0 0 0 0 0 0 0)
(1 0 0 0)	(1 1 0 1 0 0 0)
(0 1 0 0)	(0 1 1 0 1 0 0)
(1 1 0 0)	(1 0 1 1 1 0 0)
(0 0 1 0)	(1 1 1 0 0 1 0)
(1 0 1 0)	(0 0 1 1 0 1 0)
(0 1 1 0)	(1 0 0 0 1 1 0)
(1 1 1 0)	(0 1 0 1 1 1 0)
(0 0 0 1)	(1 0 1 0 0 0 1)
(1 0 0 1)	(0 1 1 1 0 0 1)
(0 1 0 1)	(1 1 0 0 1 0 1)
(1 1 0 1)	(0 0 0 1 1 0 1)
(0 0 1 1)	(0 1 0 0 0 1 1)
(1 0 1 1)	(1 0 0 1 0 1 1)
(0 1 1 1)	(0 0 1 0 1 1 1)
(1 1 1 1)	(1 1 1 1 1 1 1)

codeword. These redundant bits provide the code with the capability of combating the channel noise. For a fixed code rate R , more redundant bits can be added by increasing the number of message bits k and the block length n of the code while holding the ratio k/n constant. How to choose these redundant bits to transmit information reliably over a noisy channel is the major problem in designing the encoder. An example of a binary block code with $k = 4$ and $n = 7$ is shown in Table 1.1. Chapters 3 through 10, 14, 15, 17, 19, and 20 are devoted to the analysis, design, and decoding of block codes for controlling errors in a noisy environment.

The encoder for a convolutional code also accepts k -bit blocks of the information sequence \mathbf{u} and produces an encoded sequence (code sequence) \mathbf{v} of n -symbol blocks. (In convolutional coding, the symbols \mathbf{u} and \mathbf{v} are used to denote sequences of blocks rather than a single block.) However, each encoded block depends not only on the corresponding k -bit message block at the same time unit but also on m previous message blocks. Hence, the encoder has a *memory order* of m . The set of all possible encoded output sequences produced by the encoder forms the code. The ratio $R = k/n$ is called the *code rate*. Because the encoder contains memory, it must be implemented with a sequential logic circuit.

In a binary convolutional code, redundant bits for combating the channel noise are added to the information sequence when $k < n$, or $R < 1$. Typically, k and n are small integers, and more redundancy is added by increasing the memory order m of the code while holding k and n , and hence the code rate R , fixed. How to use the memory to achieve reliable transmission over a noisy channel is the major problem in designing the encoder of a convolutional code. An example of a binary feed-forward convolutional encoder with $k = 1$, $n = 2$, and $m = 2$ is shown in Figure 1.3. As an illustration of how codewords are generated, consider the information sequence

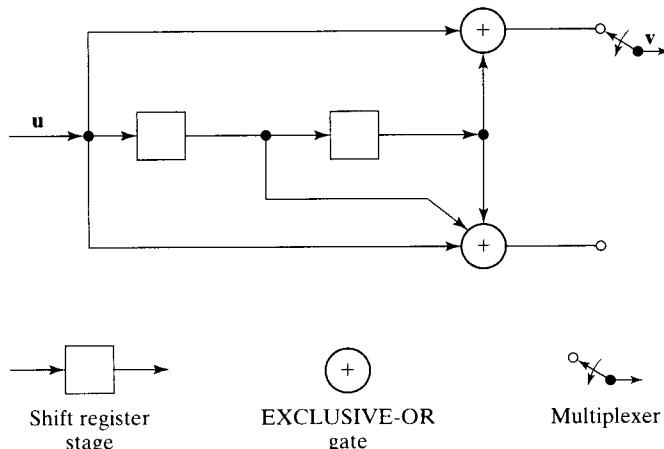


FIGURE 1.3: A binary feed-forward convolutional encoder with $k = 1$, $n = 2$, and $m = 2$.

$\mathbf{u} = (1\ 1\ 0\ 1\ 0\ 0\ 0\ \dots)$, where the leftmost bit is assumed to enter the encoder first. Using the rules of EXCLUSIVE-OR (or X-OR) addition, and assuming that the multiplexer takes the first encoded bit from the top output, it is easy to see that the encoded sequence is $\mathbf{v} = (1\ 1, 1\ 0, 1\ 0, 0\ 0, 0\ 1, 1\ 1, 0\ 0, 0\ 0, 0\ 0, \dots)$. Chapters 11 through 13, 16, 18, and 21 are devoted to the analysis, design, and decoding of convolutional codes for controlling errors in a noisy environment.

1.3 MODULATION AND CODING

The modulator in a communication system must select a waveform of duration T seconds that is suitable for transmission for each encoder output symbol. In the case of a binary code, the modulator must generate one of two signals, $s_1(t)$ for an encoded “1” or $s_2(t)$ for an encoded “0”. For a wideband channel, the optimum choice of signals is

$$\begin{aligned}
 s_1(t) &= \sqrt{\frac{2E_s}{T}} \cos 2\pi f_0 t, \quad 0 \leq t \leq T \\
 s_2(t) &= \sqrt{\frac{2E_s}{T}} \cos(2\pi f_0 t + \pi) \\
 &= -\sqrt{\frac{2E_s}{T}} \cos 2\pi f_0 t, \quad 0 \leq t \leq T,
 \end{aligned} \tag{1.1}$$

where the carrier frequency f_0 is a multiple of $1/T$, and E_s is the energy of each signal. This form of modulation is called *binary-phase-shift-keying* (BPSK), since the phase of the carrier $\cos 2\pi f_0 t$ is shifted between 0 and π , depending on the encoder output. The BPSK-modulated waveform corresponding to the codeword $\mathbf{v} = (1\ 1\ 0\ 1\ 0\ 0\ 0)$ in Table 1.1 is shown in Figure 1.4.

A common form of noise disturbance present in any communication system is *additive white Gaussian noise* (AWGN) [2, 6, 7]. If the transmitted signal is

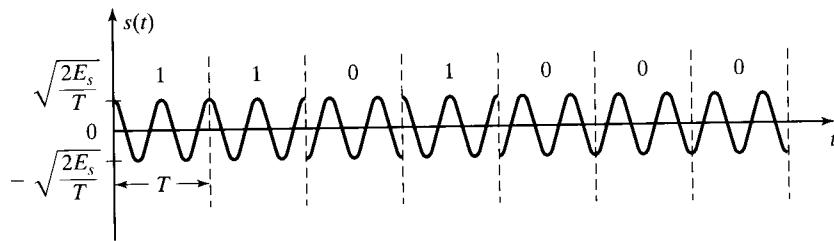


FIGURE 1.4: BPSK-modulated waveform corresponding to the codeword $v = (1101000)$.

$s(t)$ ($= s_1(t)$ or $s_2(t)$), then the received signal is

$$r(t) = s(t) + n(t), \quad (1.2)$$

where $n(t)$ is a Gaussian random process with one-sided *power spectral density* (PSD) N_0 . Other forms of noise are also present in many systems [7]. For example, in a communication system subject to multipath transmission, the received signal is observed to fade (lose strength) during certain time intervals. This fading can be modeled as a multiplicative noise scaling factor on the signal $s(t)$.

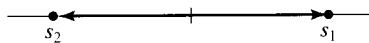
The demodulator must produce an output corresponding to the received signal in each T -second interval. This output may be a real number or one of a discrete set of preselected symbols depending on the demodulator design. An optimum demodulator always includes a matched filter or correlation detector followed by a switch that samples the output once every T seconds. For BPSK modulation with coherent detection the sampled output is a real number

$$y = \int_0^T r(t) \sqrt{\frac{2E_s}{T}} \cos 2\pi f_0 t \, dt. \quad (1.3)$$

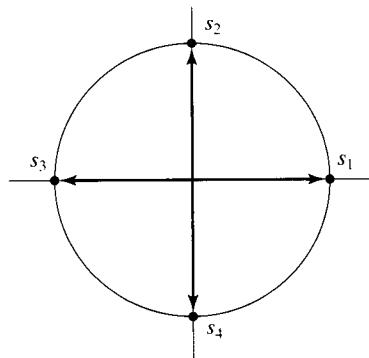
The sequence of unquantized demodulator outputs can be passed on directly to the channel decoder for processing. In this case, the channel decoder must be capable of handling unquantized inputs; that is, it must process real numbers. A much more common approach to decoding is to quantize the real-number detector output y into one of a finite number Q of discrete output symbols. In this case, the channel decoder has discrete inputs; that is, it must process discrete values. Most coded communication systems use some form of discrete processing.

To transmit information with $M = 2^l$ channel signals, the output sequence of the binary encoder is first segmented into a sequence of l -bit bytes. Each byte is called a symbol, and there are M distinct symbols. Each symbol is then mapped into one of the M signals in a signal set S for transmission. Each signal is a waveform pulse of duration T , which results in M -ary modulation. One example of M -ary modulation is *M-ary phase-shift-keying* (MPSK), for which the signal set consists of M sinusoidal pulses. These signals have the same energy E_s but M different equally spaced phases. Such a signal set is given by

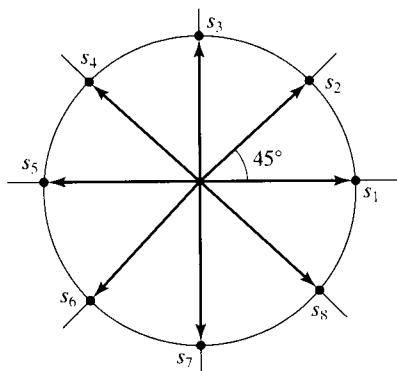
$$s_i(t) = \sqrt{\frac{2E_s}{T}} \cos(2\pi f_0 t + \phi_i), \quad 0 \leq t \leq T,$$



(a) BPSK



(b) QPSK



(c) 8-PSK

FIGURE 1.5: BPSK, QPSK, and 8-PSK signal constellations.

where $\phi_i = 2\pi(i - 1)/M$ for $1 \leq i \leq M$. Because the signals have constant envelope, MPSK is also called *constant-envelope modulation*. For $M = 2$, $M = 4$, and $M = 8$, we have BPSK, 4-PSK (also known as QPSK), and 8-PSK, respectively. These are commonly used modulations for digital communications. Their signal space constellations are depicted in Figure 1.5. Other types of M -ary modulation will be discussed in Chapters 18 and 19.

If the detector output in a given interval depends only on the transmitted signal in that interval, and not on any previous transmission, the channel is said to be *memoryless*. In this case, the combination of an M -ary input modulator, the physical channel, and a Q -ary output demodulator can be modeled as a *discrete memoryless channel* (DMC). A DMC is completely described by a set of *transition probabilities* $P(j|i)$, $0 \leq i \leq M - 1$, $0 \leq j \leq Q - 1$, where i represents a modulator input symbol, j represents a demodulator output symbol, and $P(j|i)$ is the probability of receiving j given that i was transmitted. As an example, consider a communication system in which (1) binary modulation is used ($M = 2$), (2) the amplitude distribution of the

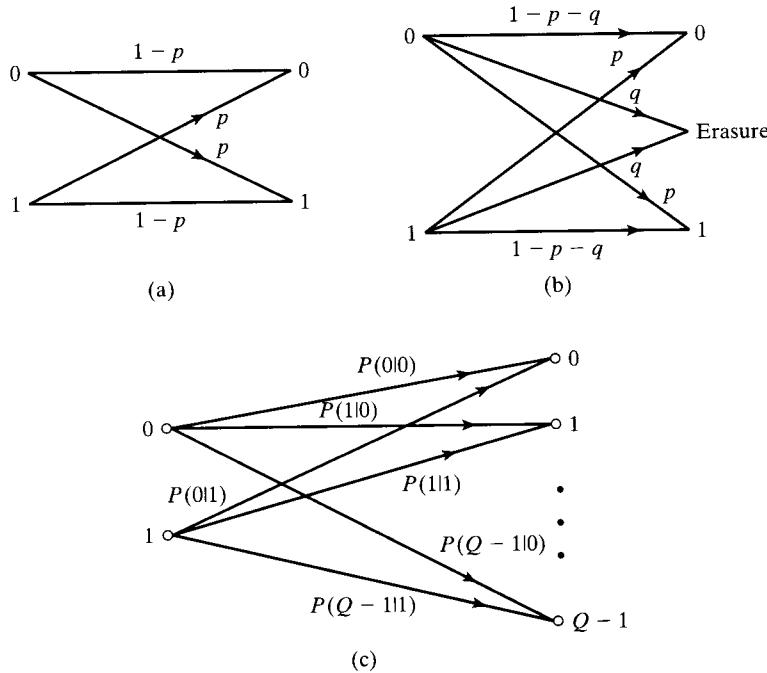


FIGURE 1.6: Transition probability diagrams for (a) binary symmetric channel (BSC); (b) binary symmetric erasure channel; and (c) binary-input, Q -ary-output discrete memoryless channel.

noise is symmetric, and (3) the demodulator output is quantized to $Q = 2$ levels. In this case a particularly simple and practically important channel model, called the *binary symmetric channel* (BSC), results. The transition probability diagram for a BSC is shown in Figure 1.6(a). Note that the transition probability p completely describes the channel.

The transition probability p can be calculated from a knowledge of the signals used, the probability distribution of the noise, and the output quantization threshold of the demodulator. When BPSK modulation is used on an AWGN channel with optimum coherent detection and binary output quantization, the BSC transition probability is just the uncoded BPSK bit error probability for equally likely signals given by

$$p = Q(\sqrt{2E_s/N_0}), \quad (1.4)$$

where $Q(x) \triangleq \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-y^2/2} dy$ is the *complementary error function* (or simply Q -function) of Gaussian statistics. An upper bound on $Q(x)$ that will be used later in evaluating the error performance of codes on a BSC is

$$Q(x) \leq \frac{1}{2} e^{-x^2/2}, \quad x \geq 0. \quad (1.5)$$

When binary coding is used, the modulator has only binary inputs ($M = 2$). Similarly, when binary demodulator output quantization is used ($Q = 2$), the

decoder has only binary inputs. In this case, the demodulator is said to make *hard decisions*. Many coded digital communication systems, whether block or convolutional, use binary coding with *hard-decision decoding* owing to the resulting simplicity of implementation; however, when $Q > 2$ (or the output is left unquantized) the demodulator is said to make *soft decisions*. In this case the decoder must accept multilevel (or continuous-valued) inputs. Although this makes the decoder more difficult to implement, *soft-decision decoding* offers significant performance improvement compared with hard-decision decoding, as is discussed in Chapters 10 and 12–19. A transition probability diagram for a soft-decision DMC with $M = 2$ and $Q > 2$ is shown in Figure 1.6(c). This is the appropriate model for a binary-input AWGN channel with finite output quantization. The transition probabilities can be calculated from a knowledge of the signals used, the probability distribution of the noise, and the output quantization thresholds of the demodulator in a manner similar to the calculation of the BSC transition probability p . The channel model given by Figure 1.6(b) is called the *binary symmetric erasure channel* (BSEC) where $Q = 3$ and one of the output symbols is an erasure (correcting errors and erasures will be discussed in Chapters 6 and 7). For a more thorough treatment of the calculation of DMC transition probabilities see references [6] and [7].

Suppose that the input to the modulator consists of symbols selected from a finite and discrete alphabet $X = \{x_0, x_1, \dots, x_{M-1}\}$ and the output of the demodulator is left unquantized. In this case, the combination of the modulator, the physical channel, and the demodulator results in a discrete-input, continuous-output channel. The channel output is a random variable that can assume any value y on the real line, i.e., $y \in (-\infty, +\infty)$. If the physical channel is subject only to AWGN with zero mean and one-sided PSD N_0 , then the channel output is a Gaussian random variable with zero mean and variance $\sigma^2 = N_0/2$. In this case, we obtain a discrete-input, continuous-output memoryless Gaussian channel. This channel is completely characterized by a set of M conditional (Gaussian) probability density functions, $p(y|x)$ for $x \in \{0, 1, \dots, M-1\}$ [7]. For the special case with $M = 2$ and $x \in \{0, 1\}$, we obtain a binary-input, continuous-output memoryless Gaussian channel. If BPSK modulation is used, the channel is completely characterized by the following conditional probability density functions:

$$\begin{aligned} p(y|x=0) &= \frac{1}{\sqrt{\pi N_o}} e^{-\frac{y+\sqrt{E_s}}{N_o}}, \\ p(y|x=1) &= \frac{1}{\sqrt{\pi N_o}} e^{-\frac{y-\sqrt{E_s}}{N_o}}, \end{aligned} \quad (1.6)$$

where E_s is the signal energy. Soft-decision decoding algorithms for this channel will be discussed in later chapters.

If the detector output in a given interval depends on the transmitted signal in previous intervals as well as the transmitted signal in the present interval, the channel is said to have *memory*. A fading channel is a good example of a channel with memory, since the multipath transmission destroys the independence from interval to interval. Appropriate models for channels with memory are difficult to construct, and coding for these channels is more of an art than a science.

Two important and related parameters in any digital communication system are the speed of information transmission and the bandwidth of the channel. Because one encoded symbol is transmitted every T seconds, the *symbol transmission rate*

(baud rate) is $1/T$. In a coded system, if the code rate is $R = k/n$, k information bits correspond to the transmission of n symbols, and the *information transmission rate* (data rate) is R/T bits per second (bps). In addition to signal modification due to the effects of noise, most communication channels are subject to signal distortion owing to bandwidth limitations. To minimize the effect of this distortion on the detection process, the channel should have a *bandwidth* W of at least $\frac{1}{2}T$ hertz (Hz).¹ In an uncoded system ($R = 1$), the data rate $1/T = 2W$ is therefore limited by the channel bandwidth. In a binary coded system, with a code rate $R < 1$, the data rate $R/T = 2RW$ is reduced by the factor R compared with an uncoded system. Hence, to maintain the same data rate as an uncoded system, the coded system requires *bandwidth expansion* by a factor of $1/R$. It is characteristic of binary coded systems to require some bandwidth expansion to maintain a constant data rate. If no additional bandwidth can be used without causing severe signal distortion, binary coding is not feasible, and bandwidth-efficient means of reliable communication must be sought. This is the subject of Chapters 18 and 19.

1.4 MAXIMUM LIKELIHOOD DECODING

A block diagram of a coded system on an AWGN channel with finite output quantization is shown in Figure 1.7. In a block-coded system, the source output \mathbf{u} represents a k -bit message, the encoder output \mathbf{v} represents an n -symbol codeword, the demodulator output \mathbf{r} represents the corresponding Q -ary received n -tuple, and the decoder output $\hat{\mathbf{u}}$ represents the k -bit estimate of the encoded message. In a

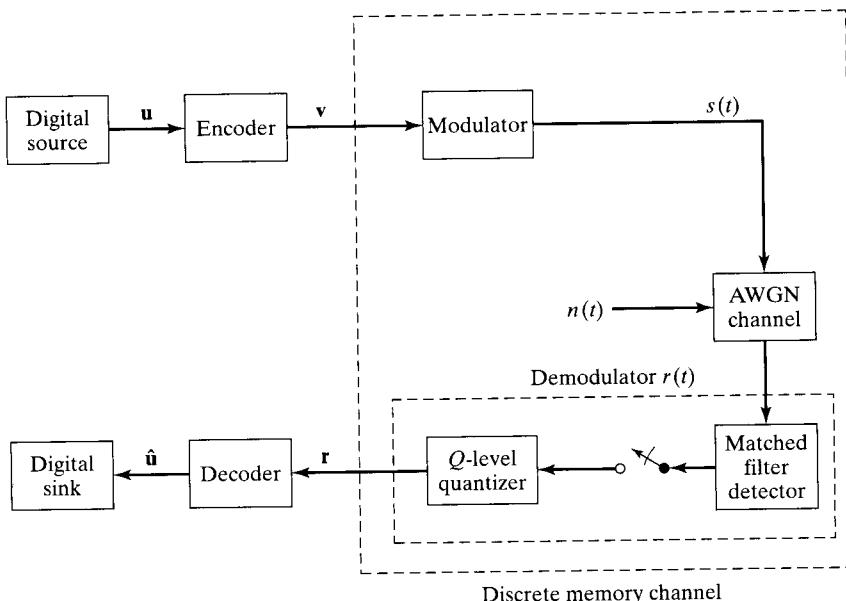


FIGURE 1.7: A coded system on an additive white Gaussian noise channel.

¹The exact bandwidth required depends on the shape of the signal waveform, the acceptable limits of distortion, and the definition of bandwidth.

convolutional-coded system, \mathbf{u} represents a sequence of kh information bits, and \mathbf{v} represents a codeword containing $N \triangleq nh + nm = n(h + m)$ symbols, where kh is the length of the information sequence, and N is the length of the codeword. The additional nm encoded symbols are produced after the last block of information bits has entered the encoder, owing to the m -time unit memory of the encoder. This topic is more fully discussed in Chapter 11. The demodulator output \mathbf{r} is a Q -ary received N -tuple, and the decoder output $\hat{\mathbf{u}}$ is a kh -bit estimate of the information sequence.

The decoder must produce an estimate $\hat{\mathbf{u}}$ of the information sequence \mathbf{u} based on the received sequence \mathbf{r} . Equivalently, since there is a one-to-one correspondence between the information sequence \mathbf{u} and the codeword \mathbf{v} , the decoder can produce an estimate $\hat{\mathbf{v}}$ of the codeword \mathbf{v} . Clearly, $\hat{\mathbf{u}} = \mathbf{u}$ if and only if $\hat{\mathbf{v}} = \mathbf{v}$. A *decoding rule* is a strategy for choosing an estimated codeword $\hat{\mathbf{v}}$ for each possible received sequence \mathbf{r} . If the codeword \mathbf{v} was transmitted, a *decoding error* occurs if and only if $\hat{\mathbf{v}} \neq \mathbf{v}$. Given that \mathbf{r} is received, the *conditional error probability of the decoder* is defined as

$$P(E|\mathbf{r}) \triangleq P(\hat{\mathbf{v}} \neq \mathbf{v}|\mathbf{r}). \quad (1.7)$$

The *error probability of the decoder* is then given by

$$P(E) = \sum_{\mathbf{r}} P(E|\mathbf{r})P(\mathbf{r}), \quad (1.8)$$

where $P(\mathbf{r})$ is the probability of the received sequence \mathbf{r} . $P(\mathbf{r})$ is independent of the decoding rule used, since \mathbf{r} is produced prior to decoding. Hence, an optimum decoding rule, that is, one that minimizes $P(E)$, must minimize $P(E|\mathbf{r}) = P(\hat{\mathbf{v}} \neq \mathbf{v}|\mathbf{r})$ for all \mathbf{r} . Because minimizing $P(\hat{\mathbf{v}} \neq \mathbf{v}|\mathbf{r})$ is equivalent to maximizing $P(\hat{\mathbf{v}} = \mathbf{v}|\mathbf{r})$, $P(E|\mathbf{r})$ is minimized for a given \mathbf{r} by choosing $\hat{\mathbf{v}}$ as the codeword \mathbf{v} that maximizes

$$P(\mathbf{v}|\mathbf{r}) = \frac{P(\mathbf{r}|\mathbf{v})P(\mathbf{v})}{P(\mathbf{r})}; \quad (1.9)$$

that is, $\hat{\mathbf{v}}$ is chosen as the most likely codeword given that \mathbf{r} is received. If all information sequences, and hence all codewords, are equally likely; that is, $P(\mathbf{v})$ is the same for all \mathbf{v} , maximizing (1.9) is equivalent to maximizing $P(\mathbf{r}|\mathbf{v})$. For a DMC,

$$P(\mathbf{r}|\mathbf{v}) = \prod_i P(r_i|v_i), \quad (1.10)$$

since for a memoryless channel each received symbol depends only on the corresponding transmitted symbol. A decoder that chooses its estimate to maximize (1.10) is called a *maximum likelihood decoder* (MLD). Because $\log x$ is a monotone increasing function of x , maximizing (1.10) is equivalent to maximizing the *log-likelihood function*,

$$\log P(\mathbf{r}|\mathbf{v}) = \sum_i \log P(r_i|v_i). \quad (1.11)$$

An MLD for a DMC then chooses $\hat{\mathbf{v}}$ as the codeword \mathbf{v} that maximizes the sum in (1.11). If the codewords are not equally likely, then an MLD is not necessarily optimum, since the conditional probabilities $P(\mathbf{r}|\mathbf{v})$ must be weighted by

the codeword probabilities $P(\mathbf{v})$ to determine which codeword maximizes $P(\mathbf{v}|\mathbf{r})$; however, in many cases, the codeword probabilities are not known exactly at the receiver, making optimum decoding impossible, and an MLD then becomes the best feasible decoding rule.

Now, consider specializing the MLD decoding rule to the BSC. In this case \mathbf{r} is a binary sequence that may differ from the transmitted codeword \mathbf{v} in some positions owing to the channel noise. When $r_i \neq v_i$, $P(r_i|v_i) = p$, and when $r_i = v_i$, $P(r_i|v_i) = 1 - p$. Let $d(\mathbf{r}, \mathbf{v})$, be the distance between \mathbf{r} and \mathbf{v} , that is, the number of positions in which \mathbf{r} and \mathbf{v} differ. This distance is called the *Hamming distance*. For a block code of length n , (1.11) becomes

$$\begin{aligned}\log P(\mathbf{r}|\mathbf{v}) &= d(\mathbf{r}, \mathbf{v}) \log p + [n - d(\mathbf{r}, \mathbf{v})] \log(1 - p) \\ &= d(\mathbf{r}, \mathbf{v}) \log \frac{p}{1-p} + n \log(1 - p).\end{aligned}\quad (1.12)$$

(For a convolutional code, n in (1.12) is replaced by N .) Since $\log \frac{p}{1-p} < 0$ for $p < \frac{1}{2}$, and $n \log(1 - p)$ is a constant for all \mathbf{v} , the MLD decoding rule for the BSC chooses $\hat{\mathbf{v}}$ as the codeword \mathbf{v} that minimizes the distance $d(\mathbf{r}, \mathbf{v})$ between \mathbf{r} and \mathbf{v} ; that is, it chooses the codeword that differs from the received sequence in the fewest number of positions. Hence, an MLD for the BSC is sometimes called a *minimum distance decoder*.

For a discrete-input, continuous-output memoryless channel, the received sequence \mathbf{r} is a sequence of real numbers. To compute the conditional probability $P(\mathbf{r}|\mathbf{v})$, we simply replace each symbol transition probability $p(r_i|v_i)$ in the product of (1.10) with the corresponding conditional probability density function, such as those given in (1.6). Then, the MLD decoding rule is to choose the codeword \mathbf{v} as the decoded codeword that maximizes the conditional probability $P(\mathbf{r}|\mathbf{v})$.

The capability of a noisy channel to transmit information reliably was determined by Shannon [1] in his original work. This result, called the *noisy channel coding theorem*, states that every channel has a *capacity* C , and that for any rate $R < C$, there exist codes of rate R that, with maximum likelihood decoding, have an arbitrarily small decoding error probability $P(E)$. In particular, for any $R < C$, there exist block codes with sufficiently large block length n such that

$$P(E) \leq 2^{-nE_b(R)}, \quad (1.13)$$

and there exist convolutional codes with sufficiently large memory order m such that

$$P(E) \leq 2^{-(m+1)nE_c(R)}. \quad (1.14)$$

$E_b(R)$ and $E_c(R)$ are positive functions of R for $R < C$ and are completely determined by the channel characteristics. The bound of (1.13) implies that arbitrarily small error probabilities are achievable with block coding for any fixed $R < C$ by increasing the block length n while holding the ratio k/n constant. The bound of (1.14) implies that arbitrarily small error probabilities are also achievable with convolutional coding for any fixed $R < C$ by increasing the memory order m while holding k and n constant.

The noisy channel coding theorem is based on an argument called *random coding*. The bound obtained is actually on the average error probability of the

ensemble of all codes. Because some codes must perform better than the average, the noisy channel coding theorem guarantees the existence of codes satisfying (1.13) and (1.14), but it does not indicate how to construct these codes. Furthermore, to achieve very low error probabilities for block codes of fixed rate $R < C$, long block lengths are needed. This requires that the number of codewords $2^k = 2^{nR}$ must be very large. Because an MLD must compute $\log P(\mathbf{r}|\mathbf{v})$ for each codeword, and then choose the codeword that gives the maximum, the number of computations that must be performed by an MLD becomes very large as n increases. For convolutional codes, low error probabilities require a large memory order m . As will be seen in Chapter 12, an MLD for convolutional codes requires approximately 2^{km} computations to decode each block of k information bits. This number, too, becomes very large as m increases. Hence, it is impractical to achieve very low error probabilities with maximum likelihood decoding. Therefore, two major problems are encountered when designing a coded system to achieve low error probabilities: (1) to construct good long codes whose performance with MLD satisfies (1.13) and (1.14), and (2) to find easily implementable methods of encoding and decoding these codes such that their actual performance is close to what could be achieved with MLD. The remainder of this book is devoted to finding solutions to these two problems.

1.5 TYPES OF ERRORS

On memoryless channels, the noise affects each transmitted symbol independently. As an example, consider the BSC whose transition diagram is shown in Figure 1.6(a). Each transmitted bit has a probability p of being received incorrectly and a probability $1 - p$ of being received correctly, independent of the other transmitted bits. Hence, transmission errors occur randomly in the received sequence, and memoryless channels are called *random-error channels*. Good examples of random-error channels are the deep-space channel and many satellite channels. Most line-of-sight transmission facilities, as well, are primarily affected by random errors. Codes designed to correct random errors are called *random-error correcting codes*. Most of the codes presented in this book are random-error correcting codes.

On channels with memory, the noise is not independent from transmission to transmission. A simplified model of a channel with memory is shown in Figure 1.8. This model contains two states, a “good state,” in which transmission errors occur infrequently, $p_1 \approx 0$, and a “bad state,” in which transmission errors are highly probable, $p_2 \approx .5$. The channel is in the good state most of the time but on occasion shifts to the bad state owing to a change in the transmission characteristic of the channel, for example, a “deep fade” caused by multipath transmission. As a consequence, transmission errors occur in clusters or bursts because of the high transition probability in the bad state, and channels with memory are called *burst-error channels*. Examples of burst-error channels are mobile telephony channels, where the error bursts are caused by signal fading owing to multipath transmission; cable transmission, which is affected by impulsive switching noise and crosstalk; and magnetic recording, which is subject to dropouts caused by surface defects and dust particles. Codes designed to correct burst errors are called *burst-error correcting codes*. Chapters 20 and 21 are devoted to codes of this type and coding techniques for correcting burst errors.

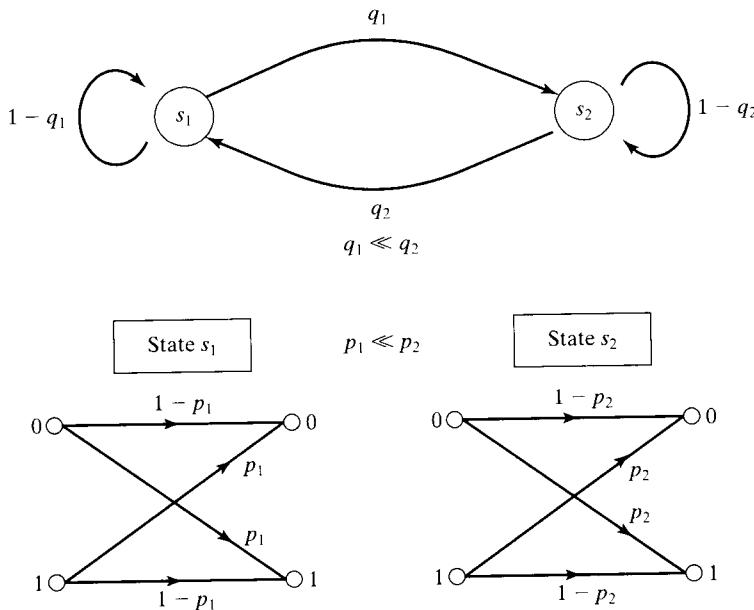


FIGURE 1.8: A simplified model of a channel with memory.

Finally, some channels contain a combination of both random and burst errors. These are called *compound channels*, and codes designed to correct errors on these channels are called *burst-and-random-error correcting codes*. Sections 20.5 and 21.4 are devoted to codes of this type.

1.6 ERROR CONTROL STRATEGIES

The block diagram shown in Figure 1.1 represents a one-way communication system. The transmission (or recording) is strictly in one direction, from transmitter to receiver. Error control strategies for a one-way system must use *forward error correction* (FEC); that is, they employ error-correcting codes that automatically correct errors detected at the receiver. Examples are digital storage systems, in which the information recorded on a storage medium may be replayed weeks or even months after it is recorded; and deep-space communication systems, where the relatively simple encoding equipment can be placed aboard the spacecraft, but the much more complex decoding procedure must be performed on Earth. Most of the coded systems in use today employ some form of FEC, even if the channel is not strictly one-way. This book is devoted mostly to the analysis and design of FEC systems.

In some cases, though, a transmission system can be two-way; that is, information can be sent in both directions, and the transmitter also acts as a receiver (a transceiver), and vice versa. Examples of two-way systems are some data networks and satellite communication systems. Error control strategies for a two-way system use error detection and retransmission, called *automatic repeat request* (ARQ). In an ARQ system, when errors are detected at the receiver, a request is sent for the

transmitter to repeat the message, and repeat requests continue to be sent until the message is correctly received.

There are two types of ARQ systems: stop-and-wait ARQ and continuous ARQ. With *stop-and-wait ARQ* the transmitter sends a codeword to the receiver and waits for a positive (ACK) or negative (NAK) acknowledgment from the receiver. If an ACK is received (no errors detected), the transmitter sends the next codeword; however, if a NAK is received (errors detected), the transmitter resends the previous codeword. When the noise is persistent, the same codeword may be retransmitted several times before it is correctly received and acknowledged.

With *continuous ARQ*, the transmitter sends codewords to the receiver continuously and receives acknowledgments continuously. When a NAK is received, the transmitter begins a retransmission. It may back up to the codeword in error and resend that codeword plus the $N - 1$ codewords that follow it. This is called *go-back-N ARQ*. Alternatively, the transmitter may simply resend only those codewords that are negatively acknowledged. This is known as *selective-repeat ARQ*. Selective-repeat ARQ is more efficient than go-back-N ARQ, but requires more logic and buffering.

Continuous ARQ is more efficient than stop-and-wait ARQ, but it is also more expensive to implement. In a satellite communication system, where the transmission rate is high and the round-trip delay is long, continuous ARQ is normally used. Stop-and-wait ARQ is used in systems where the time taken to transmit a codeword is long compared to the time taken to receive an acknowledgment. Stop-and-wait ARQ is designed for use on half-duplex channels (only one-way transmission at a time), whereas continuous ARQ is designed for use on full-duplex channels (simultaneous two-way transmission).

The major advantage of ARQ over FEC is that error detection requires much simpler decoding equipment than error correction. Also, ARQ is adaptive in the sense that information is retransmitted only when errors occur. In contrast, when the channel error rate is high, retransmissions must be sent too frequently, and the *system throughput*—the rate at which newly generated messages are correctly received—is lowered by ARQ. In this situation, a hybrid combination of FEC for the most frequent error patterns along with error detection and retransmission for the less likely error patterns is more efficient than ARQ alone. Various types of ARQ and hybrid ARQ schemes are presented in Chapter 22.

1.7 PERFORMANCE MEASURES

The performance of a coded communication system is in general measured by its probability of decoding error (called the *error probability*) and its *coding gain* over an uncoded system that transmits information at the same rate. There are two types of error probability, probability of word (or block) error and probability of bit error. The probability of word error is defined as the probability that a decoded word (or block) at the output of the decoder is in error. This error probability is commonly called the *word-error rate* (WER) or *block-error rate* (BLER). The probability of bit error, also called the *bit-error rate* (BER), is defined as the probability that a decoded information bit at the output of the decoder is in error. A coded communication system should be designed to keep these two error probabilities as

low as possible under certain system constraints, such as power, bandwidth, and decoding complexity.

In a coded communication system with code rate $R = k/n$, the number of transmitted symbols (or signals) per information bit is $1/R$. If the energy per transmitted symbol is E_s , then the energy-per-information bit is

$$E_b = E_s/R. \quad (1.15)$$

The error probability of a coded communication system is commonly expressed in terms of the ratio of energy-per-information bit E_b to the one-sided PSD N_0 of the channel noise.

Consider a coded communication system using the (23,12) binary Golay code [8] for error control (see Chapter 5 for a complete description of this code). Each codeword consists of 23 code digits, of which 12 are information bits. Therefore, there are 11 redundant bits, and the code rate is $R = 12/23 = 0.5217$. Suppose BPSK modulation with coherent detection is used and the channel is an AWGN channel with one-sided PSD N_0 . Let E_b/N_0 denote the ratio of energy-per-information-bit to noise power spectral density at the input to the receiver, which is called the *signal-to-noise ratio* (SNR) and is usually expressed in decibels (dBs). The bit-error performance of the (23,12) Golay code with both hard-decision decoding and (unquantized) soft-decision maximum likelihood decoding versus E_b/N_0 is depicted in Figure 1.9. For comparison, the bit-error performance of an uncoded BPSK system is also included. We see that the coded system, with either hard- or soft-decision decoding, provides a lower bit-error probability than the uncoded system for the same SNR when the SNR is above a certain threshold. For hard-decision decoding of the Golay code, this threshold is 3.7 dB. For SNR = 7 dB, the BER of the uncoded BPSK system is 8×10^{-4} , whereas the coded system achieves a BER of 2.9×10^{-5} . This is a significant improvement in error performance. For SNR = 5 dB, the performance improvement of the Golay-coded system over the uncoded system is small: 2.1×10^{-3} compared to 6.5×10^{-3} ; however, with soft-decision decoding, the coded system achieves a BER of 7×10^{-5} .

The other performance measure of a coded communication system is the *coding gain*. Coding gain is defined as the reduction in the E_b/N_0 required to achieve a specific error probability (BER or WER) for a coded communication system compared to an uncoded system. For example, in Figure 1.9, for a BER = 10^{-5} , the Golay-coded system with hard-decision decoding has a coding gain of 2.15 dB over uncoded BPSK, whereas with soft-decision decoding (MLD), a coding gain of more than 4.0 dB is achieved. This result shows that soft-decision decoding of the Golay code achieves 1.85 dB additional coding gain compared to hard-decision decoding at a BER of 10^{-5} . This additional coding gain is achieved at the expense of higher decoding complexity. Coding gain is important in communication applications where every decibel of improved performance results in substantial savings in overall system cost.

From Figure 1.9, we observe that at sufficiently low values of E_b/N_0 , the coding gain actually becomes negative. This threshold phenomenon is common to all coding schemes. There always exists an E_b/N_0 below which the code loses its effectiveness and actually makes the situation worse. This SNR is called the *coding threshold*. It is important to keep this threshold low and to maintain a coded communication

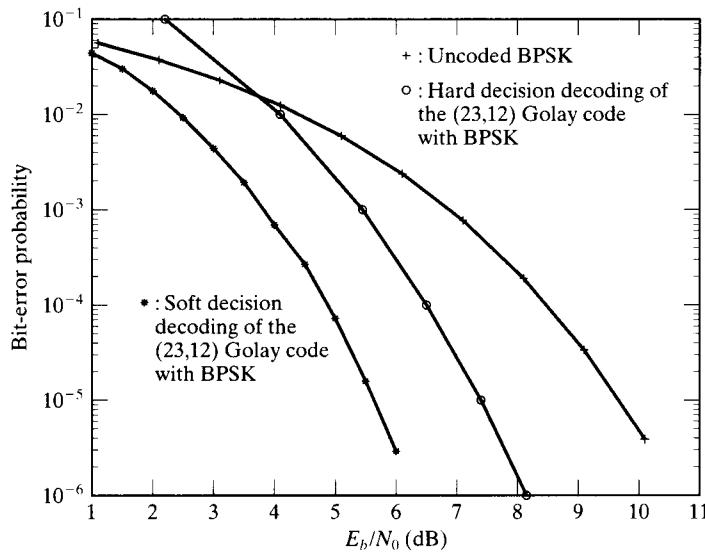


FIGURE 1.9: Bit-error performance of a coded communication system with the (23,12) Golay code.

system operating at an SNR well above its coding threshold. For the Golay-coded system with hard-decision decoding, the coding threshold is 3.7 dB.

Another quantity that is sometimes used as a performance measure for a particular code is the *asymptotic coding gain* (the coding gain for large SNR). This coding gain depends only on the code rate and the minimum distance d_{\min} of the code, which is defined as the minimum Hamming distance between any two codewords. For an AWGN channel, it follows from (1.4) and (1.5) that for high SNR (large E_b/N_0), the bit-error probability of an uncoded BPSK system with coherent detection can be approximated by

$$(P_b)_{\text{uncoded}} \cong 1/2e^{-(E_b/N_0)_{\text{uncoded}}}. \quad (1.16)$$

For high SNR with soft-decision (unquantized) MLD, the bit error probability of an (n, k) block code with minimum distance d_{\min} can be approximated by (see Chapter 10)

$$(P_b)_{\text{coded}} \cong Ke^{-d_{\min}R(E_b/N_0)_{\text{coded}}}, \quad (1.17)$$

where $R = k/n$ is the code rate, and K is a (typically small) constant that depends only on the code. One may compute the reduction in SNR by equating (1.16) and (1.17), which gives

$$Ke^{-d_{\min}R(E_b/N_0)_{\text{coded}}} = \frac{1}{2}e^{-(E_b/N_0)_{\text{uncoded}}}. \quad (1.18)$$

Taking the logarithm (base e) of both sides and noting that $\log K$ and $\log \frac{1}{2}$ are negligible for large E_b/N_0 , we obtain

$$\frac{(E_b/N_0)_{\text{uncoded}}}{(E_b/N_0)_{\text{coded}}} = Rd_{\min}. \quad (1.19)$$

Thus, for soft-decision MLD, the asymptotic coding gain of an (n, k) code with minimum distance d_{min} over uncoded BPSK is given by

$$\gamma_{asym} = 10 \log_{10} R d_{min}. \quad (1.20)$$

For hard-decision MLD, it can be shown [7] that the asymptotic coding gain is

$$\gamma_{asym} = 10 \log_{10} \left(\frac{1}{2} \right) R d_{min}. \quad (1.21)$$

From (1.20) and (1.21), we see that soft-decision MLD has a

$$10 \log_{10} 2 = 3 \text{ dB}$$

asymptotic coding gain advantage over hard-decision MLD. This 3 dB coding gain is achieved only for very large SNR. For more realistic values of SNR, the real coding gain of soft-decision decoding over hard-decision decoding is less than 3 dB, typically between 2 and 2.5 dB. Expressions similar to (1.20) and (1.21) are given in Chapter 12 for convolutional codes, with the minimum free distance d_{free} replacing d_{min} .

In designing a coding system for error control, it is desired to minimize the SNR required to achieve a specific error rate. This is equivalent to maximizing the coding gain of the coded system compared to an uncoded system using the same modulation signal set. A theoretical limit on the minimum SNR required for a coded system with code rate R to achieve error-free communication (or an arbitrarily small error probability) can be derived based on Shannon's noisy coding theorem [1]. This theoretical limit (often called the *Shannon limit*) simply says that for a coded system with code rate R , error-free communication is achievable only if the SNR exceeds this

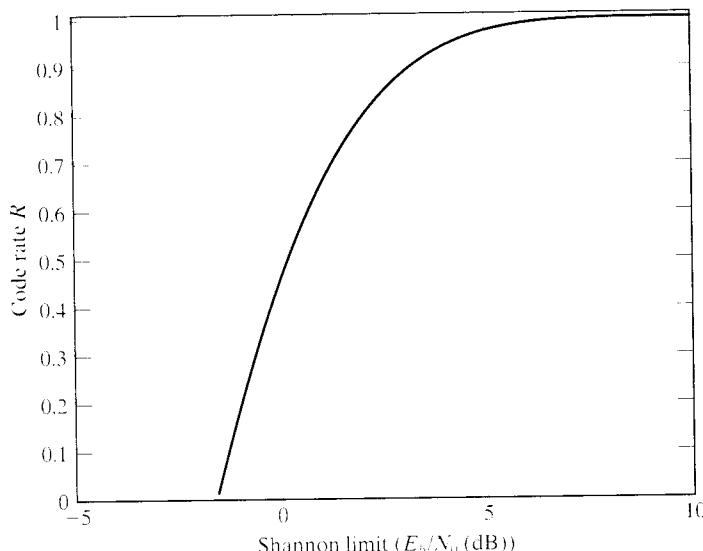


FIGURE 1.10: Shannon limit as a function of code rate R .

limit. As long as the SNR exceeds this limit, Shannon's coding theorem guarantees the existence of a (perhaps very complex) coded system capable of achieving error-free communication. For transmission over a binary-input, continuous-output AWGN channel with BPSK signaling, the Shannon limit (in terms of E_b/N_0) as a function of code rate R does not have a closed form; however, it can be evaluated numerically. Figure 1.10 shows the Shannon limit as a function of the code rate R for BPSK signaling on a continuous-output AWGN channel. From this figure we can determine the Shannon limit for a given code rate. For example, to achieve error-free communication with a coded system of rate $R = \frac{1}{2}$ over a continuous-output AWGN channel using BPSK signaling, the minimum required SNR (the Shannon limit) is 0.188 dB. Table 1.2 gives the values of the Shannon limit for various code rates from 0 to 0.999. The Shannon limit can be used as a yardstick to measure the maximum achievable coding gain for a coded system with a given rate R over an uncoded system with the same modulation signal set. For example, to achieve a BER of 10^{-5} , an uncoded BPSK system requires an SNR of 9.65 dB. For a coded BPSK system with code rate $R = \frac{1}{2}$, the Shannon limit is 0.188. Therefore, the maximum achievable (or potential) coding gain for a coded BPSK system with a code rate $R = \frac{1}{2}$ is 9.462 dB. For example, Figure 1.11 shows the bit-error performance of a rate $R = \frac{1}{2}$ convolutional code with memory order $m = 6$ decoded with soft-decision MLD (see Chapter 12). To achieve a BER of 10^{-5} , this code requires an SNR of 4.15 dB and achieves a 5.35 dB coding gain compared to uncoded BPSK; however, it is 3.962 dB away from the Shannon limit. This gap can be reduced by using a longer and more powerful code, block or convolutional, decoded with an efficient soft-decision MLD or near-MLD algorithm. As another example, consider Figure 1.12

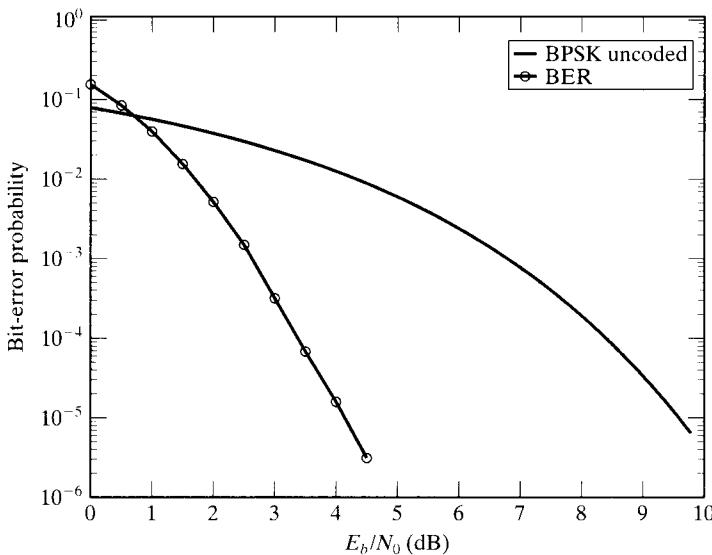


FIGURE 1.11: Bit-error performance of a rate $R = \frac{1}{2}$ convolutional code with memory order $m = 6$ decoded with soft-decision MLD.

TABLE 1.2: Shannon limit of a continuous-output AWGN channel with BPSK signaling for various code rates R .

R	E_b/N_0 (dB)	R	E_b/N_0 (dB)	R	E_b/N_0 (dB)	R	E_b/N_0 (dB)
0.01	-1.548	0.35	-0.432	0.69	1.208	0.954	4.304
0.02	-1.531	0.36	-0.394	0.70	1.275	0.958	4.425
0.03	-1.500	0.37	-0.355	0.71	1.343	0.961	4.521
0.04	-1.470	0.38	-0.314	0.72	1.412	0.964	4.618
0.05	-1.440	0.39	-0.276	0.73	1.483	0.967	4.725
0.06	-1.409	0.40	-0.236	0.74	1.554	0.970	4.841
0.07	-1.378	0.41	-0.198	0.75	1.628	0.972	4.922
0.08	-1.347	0.42	-0.156	0.76	1.708	0.974	5.004
0.09	-1.316	0.42	-0.118	0.77	1.784	0.976	5.104
0.10	-1.285	0.44	-0.074	0.78	1.867	0.978	5.196
0.11	-1.254	0.45	-0.032	0.79	1.952	0.980	5.307
0.12	-1.222	0.46	0.010	0.800	2.045	0.982	5.418
0.13	-1.190	0.47	0.055	0.807	2.108	0.983	5.484
0.14	-1.158	0.48	0.097	0.817	2.204	0.984	5.549
0.15	-1.126	0.49	0.144	0.827	2.302	0.985	5.615
0.16	-1.094	0.50	0.188	0.837	2.402	0.986	5.681
0.17	-1.061	0.51	0.233	0.846	2.503	0.987	5.756
0.18	-1.028	0.52	0.279	0.855	2.600	0.988	5.842
0.19	-0.995	0.53	0.326	0.864	2.712	0.989	5.927
0.20	-0.963	0.54	0.374	0.872	2.812	0.990	6.023
0.21	-0.928	0.55	0.424	0.880	2.913	0.991	6.119
0.22	-0.896	0.56	0.474	0.887	3.009	0.992	6.234
0.23	-0.861	0.57	0.526	0.894	3.114	0.993	6.360
0.24	-0.827	0.58	0.574	0.900	3.205	0.994	6.495
0.25	-0.793	0.59	0.628	0.907	3.312	0.995	6.651
0.26	-0.757	0.60	0.682	0.913	3.414	0.996	6.837
0.27	-0.724	0.61	0.734	0.918	3.500	0.997	7.072
0.28	-0.687	0.62	0.791	0.924	3.612	0.998	7.378
0.29	-0.651	0.63	0.844	0.929	3.709	0.999	7.864
0.30	-0.616	0.64	0.904	0.934	3.815		
0.31	-0.579	0.65	0.960	0.938	3.906		
0.32	-0.544	0.66	1.021	0.943	4.014		
0.33	-0.507	0.67	1.084	0.947	4.115		
0.34	-0.469	0.68	1.143	0.951	4.218		

which depicts the bit-error performance of a (65520,61425) low-density parity-check code (see Chapter 17) [12] decoded with a near-MLD soft-decision algorithm. The rate of this code is $R = 15/16 = 0.9375$. The Shannon limit for this rate is 3.91 dB. At the BER of 10^{-5} , we find that the error performance of this code is less than 0.5 dB away from the Shannon limit. This example demonstrates that the Shannon limit can be approached with a properly designed code of sufficient length decoded with an

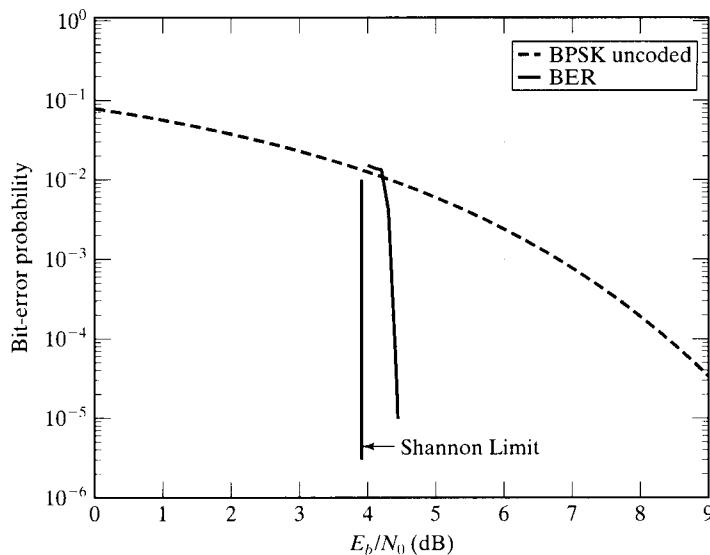


FIGURE 1.12: Bit-error performance of a (65520, 61425) low-density parity check code decoded with a soft-decision near-MLD algorithm.

efficient soft-decision MLD or near-MLD algorithm. Shannon limit–approaching codes are discussed in Chapters 16 and 17.

1.8 CODED MODULATION

Combining coding with binary modulation to achieve coding gain (or to improve error performance) results in channel bandwidth expansion; that is, the coded system requires a larger channel bandwidth for signal transmission than the uncoded system. This is because redundant symbols must be added to the transmitted information sequence to combat the channel noise (i.e., for error control). As shown in Section 1.3, to maintain the same information transmission rate using a code with rate $R = k/n < 1$ for error control requires bandwidth expansion by a factor of $1/R$. Therefore, coding gain is achieved at the expense of bandwidth expansion. For example, the Golay-coded system with BPSK modulation described in the previous section requires bandwidth expansion by a factor of $1/R = 23/12 = 1.917$. Therefore, coding gain is achieved at a cost of almost doubling the required bandwidth of the transmitted signal, plus the additional cost of decoder implementation. This type of coding provides an effective trade-off between channel bandwidth and transmitter power and is suitable for digital communication systems that are designed to operate on power-limited channels where $R/W < 1$ [7], such as deep-space, satellite, and other wideband communication systems.

For many communication channels, however, such as voiceband telephone, terrestrial microwave, and some satellite channels, bandwidth is limited, and hence bandwidth expansion is not desirable or even possible. For this reason, coding by adding extra redundant symbols for error control is rarely used on

bandlimited channels. To overcome this problem, coding must be used in conjunction with bandwidth-efficient multilevel modulation (M -ary modulation with $M > 2$). If coding and modulation are combined properly and designed as a single entity, coding gain can be achieved without bandwidth expansion [9]. Such a combination of coding and modulation is called *coded modulation*. The first coded modulation technique, known as *trellis coded modulation*, was devised by Ungerboeck [10].

The basic concept of coded modulation is to encode information symbols onto an expanded modulation signal set (relative to that needed for uncoded modulation). This signal set expansion provides the needed redundancy for error control without increasing the signaling rate and hence without increasing the bandwidth requirement. Coding is used to form structured signal sequences with large minimum *Euclidean distance* between any two transmitted signal sequences. For example, consider the simple coded modulation system shown in Figure 1.13, in which a rate $R = \frac{1}{2}$ feedback convolutional encoder with memory order $m = 2$ and an 8-PSK modulation signal set are used. During each unit of time, the two output code bits of the convolutional encoder together with an uncoded information bit are used to select one 8-PSK signal (properly known as *mapping by set partitioning* [10]) for transmission. As a result, each 8-PSK signal carries two information bits, and hence the information transmission rate (called the *spectral efficiency*) for this coded modulation system is 2 bits per transmitted signal. Now, consider an uncoded QPSK (or 4-PSK) system. With this system, information is also transmitted at a rate of 2 bits per transmitted signal. Therefore, the coded 8-PSK and uncoded QPSK systems both have the same spectral efficiency, 2 bits per transmitted signal, and thus they require the same bandwidth for signal transmission. Ungerboeck showed that this coded 8-PSK system with soft-decision MLD achieves a 3 dB asymptotic coding gain over an uncoded QPSK system without bandwidth expansion [10, 11; also see Chapter 18]. This simple coded 8-PSK system can be soft-decision decoded in a straightforward manner (see Chapter 18). Its bit-error performance is shown in Figure 1.14. At the BER of 10^{-5} , it achieves a 2.4 dB real coding gain over the uncoded QPSK system.

Coded modulation can be classified into two categories based on the code structure:

1. Trellis coded modulation (TCM), in which convolutional (or trellis) codes are combined with multilevel modulation signal sets, and

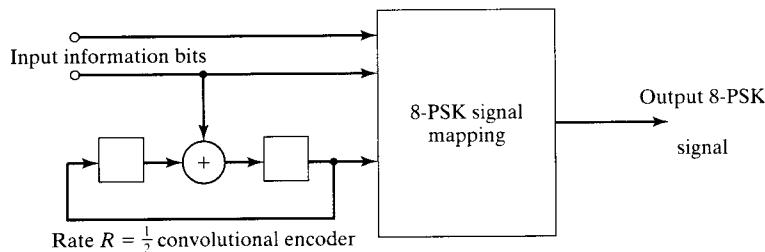


FIGURE 1.13: A trellis coded 8-PSK modulation system.

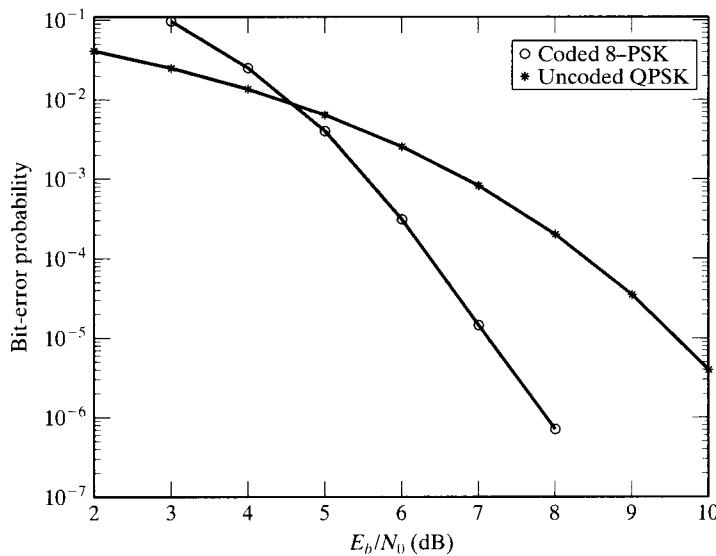


FIGURE 1.14: Bit error performance of the coded 8-PSK system shown in Figure 1.13.

2. Block coded modulation (BCM) [13], in which block codes are combined with multilevel modulation signal sets.

TCM is the subject of Chapter 18, and BCM is discussed in Chapter 19.

Basically, coding for error control is achieved by adding properly designed redundancy to the transmitted information sequence. The redundancy is obtained either by adding extra symbols to the transmitted information sequence or by channel signal-set expansion in which a larger signal set is used for mapping the transmitted information sequence into a signal sequence. Coding by adding redundant symbols results in bandwidth expansion and is suitable for error control in power-limited communication systems. Coding by channel signal set expansion allows coding gain without bandwidth expansion and is suitable for error control in bandwidth-limited communication systems.

BIBLIOGRAPHY

1. C. E. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. J.*, pp. 379–423 (Part 1); pp. 623–56 (Part 2), July 1948.
2. R. G. Gallager, *Information Theory and Reliable Communication*, John Wiley, New York, 1968.
3. T. Berger, *Rate Distortion Theory*, Prentice Hall, Englewood Cliffs, N.J., 1971.
4. R. Gray, *Source Coding Theory*, Kluwer Academic, Boston, Mass., 1990.
5. T. Cover and J. Thomas, *Elements of Information Theory*, Wiley Inter-Science, New York, 1991.

6. J. M. Wozencraft and I. M. Jacob, *Principles of Communication Engineering*, John Wiley, New York, 1965.
7. J. G. Proakis, *Digital Communications*, 4th ed., McGraw-Hill, New York, 2000.
8. M. J. E. Golay, "Notes on Digital Coding," *Proc. IRE*, p. 657, June 1949.
9. J. L. Massey, "Coding and Modulation in Digital Communications," in Proc. 1974 Int. Zurich Seminar on Digital Communications, Zurich, Switzerland, March 1974.
10. G. Ungerboeck, "Channel Coding with Multilevel/Phase Signals," *IEEE Trans. Inform. Theory*, IT-28 (no. 1): 55–67, January 1982.
11. ———, "Trellis-Coded Modulation with Redundant Signal Sets: Parts I and II," *IEEE Commun. Mag.*, 25: 5–21, February 1987.
12. R. G. Gallager, "Low Density Parity Check Codes," *IRE Trans. Inform. Theory*, IT-8: 21–28, January 1962.
13. H. Imai and S. Hirakawa, "A New Multilevel Coding Method Using Error-Correcting Codes," *IEEE Trans. Inform. Theory*, 23 (no. 3): 371–76, May 1977.

Introduction to Algebra

The purpose of this chapter is to provide the reader with an elementary knowledge of algebra that will aid in the understanding of the material in the following chapters. The treatment is basically descriptive, and no attempt is made to be mathematically rigorous. There are many good textbooks on algebra. The reader who is interested in more advance algebraic coding theory is referred to the textbooks listed at the end of the chapter.

2.1 GROUPS

Let G be a set of elements. A *binary operation* $*$ on G is a *rule* that assigns to each pair of elements a and b a uniquely defined third element $c = a * b$ in G . When such a binary operation $*$ is defined on G , we say that G is *closed* under $*$. For example, let G be the set of all integers and let the binary operation on G be real addition $+$. We all know that, for any two integers i and j in G , $i + j$ is a uniquely defined integer in G . Hence, the set of integers is closed under real addition. A binary operation $*$ on G is said to be *associative* if, for any a , b , and c in G ,

$$a * (b * c) = (a * b) * c.$$

Now, we introduce a useful algebraic system called a *group*.

DEFINITION 2.1 A set G on which a binary operation $*$ is defined is called a *group* if the following conditions are satisfied:

- i. The binary operation $*$ is associative.
- ii. G contains an element e such that, for any a in G ,

$$a * e = e * a = a.$$

This element e is called an *identity element* of G .

- iii. For any element a in G , there exists another element a' in G such that

$$a * a' = a' * a = e.$$

The element a' is called an *inverse* of a (a is also an inverse of a').

A group G is said to be *commutative* if its binary operation $*$ also satisfies the following condition: For any a and b in G ,

$$a * b = b * a.$$

THEOREM 2.1 The identity element in a group G is unique.

Proof. Suppose that there exist two identity elements e and e' in G . Then $e' = e' * e = e$. This implies that e and e' are identical. Therefore, there is one and only one identity element. **Q.E.D.**

THEOREM 2.2 The inverse of a group element is unique.

Proof. Suppose that there exist two inverses a' and a'' for a group element a . Then

$$a' = a' * e = a' * (a * a'') = (a' * a) * a'' = e * a'' = a''.$$

This implies that a' and a'' are identical and there is only one inverse for a .

Q.E.D.

The set of all integers is a commutative group under real addition. In this case, the integer 0 is the identity element, and the integer $-i$ is the inverse of integer i . The set of all rational numbers excluding zero is a commutative group under real multiplication. The integer 1 is the identity element with respect to real multiplication, and the rational number b/a is the multiplicative inverse of a/b . The groups just noted contain infinite numbers of elements. Groups with finite numbers of elements do exist, as we shall see in the next example.

EXAMPLE 2.1

Consider the set of two integers $G = \{0, 1\}$. Let us define a binary operation, denoted by \oplus , on G as follows:

$$0 \oplus 0 = 0, \quad 0 \oplus 1 = 1, \quad 1 \oplus 0 = 1, \quad 1 \oplus 1 = 0.$$

This binary operation is called *modulo-2* addition. The set $G = \{0, 1\}$ is a group under modulo-2 addition. It follows from the definition of modulo-2 addition \oplus that G is closed under \oplus , and \oplus is commutative. We can easily check that \oplus is associative. The element 0 is the identity element. The inverse of 0 is itself, and the inverse of 1 is also itself. Thus, G together with \oplus is a commutative group.

The number of elements in a group is called the *order* of the group. A group of finite order is called a *finite* group. For any positive integer m , it is possible to construct a group of order m under a binary operation that is very similar to real addition, as is shown in the next example.

EXAMPLE 2.2

Let m be a positive integer. Consider the set of integers $G = \{0, 1, 2, \dots, m - 1\}$. Let $+$ denote real addition. Define a binary operation \boxplus on G as follows: For any integers i and j in G ,

$$i \boxplus j = r,$$

where r is the *remainder* resulting from dividing $i + j$ by m . The remainder r is an integer between 0 and $m - 1$ (Euclid's division algorithm) and is therefore in G . Hence, G is closed under the binary operation \boxplus , which is called *modulo- m*

addition. The set $G = \{0, 1, \dots, m - 1\}$ is a group under modulo- m addition. First, we see that 0 is the identity element. For $0 < i < m$, i and $m - i$ are both in G . Since

$$i + (m - i) = (m - i) + i = m,$$

it follows from the definition of modulo- m addition that

$$i \boxplus (m - i) = (m - i) \boxplus i = 0.$$

Therefore, i and $m - i$ are inverses of each other with respect to \boxplus . It is also clear that the inverse of 0 is itself. Because real addition is commutative, it follows from the definition of modulo- m addition that, for any i and j in G , $i \boxplus j = j \boxplus i$. Therefore, modulo- m addition is commutative. Next, we show that modulo- m addition is also associative. Let i , j , and k be three integers in G . Since real addition is associative, we have

$$i + j + k = (i + j) + k = i + (j + k).$$

Dividing $i + j + k$ by m , we obtain

$$i + j + k = qm + r,$$

where q and r are the quotient and the remainder, respectively, and $0 \leq r < m$. Now, dividing $i + j$ by m , we have

$$i + j = q_1 m + r_1 \tag{2.1}$$

with $0 \leq r_1 < m$. Therefore, $i \boxplus j = r_1$. Dividing $r_1 + k$ by m , we obtain

$$r_1 + k = q_2 m + r_2 \tag{2.2}$$

with $0 \leq r_2 < m$. Hence, $r_1 \boxplus k = r_2$, and

$$(i \boxplus j) \boxplus k = r_2.$$

Combining (2.1) and (2.2), we have

$$i + j + k = (q_1 + q_2)m + r_2.$$

This implies that r_2 is also the remainder when $i + j + k$ is divided by m . Because the remainder resulting from dividing an integer by another integer is unique, we must have $r_2 = r$. As a result, we have

$$(i \boxplus j) \boxplus k = r.$$

Similarly, we can show that

$$i \boxplus (j \boxplus k) = r.$$

Therefore, $(i \boxplus j) \boxplus k = i \boxplus (j \boxplus k)$, and modulo- m addition is associative. This concludes our proof that the set $G = \{0, 1, 2, \dots, m - 1\}$ is a group under modulo- m addition. We shall call this group an *additive group*. For $m = 2$, we obtain the binary group given in Example 2.1.

TABLE 2.1: Modulo-5 addition.

\oplus	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

The additive group under modulo-5 addition is given in Table 2.1.

Finite groups with a binary operation similar to real multiplication also can be constructed.

EXAMPLE 2.3

Let p be a prime (e.g., $p = 2, 3, 5, 7, 11, \dots$). Consider the set of integers, $G = \{1, 2, 3, \dots, p - 1\}$. Let \cdot denote real multiplication. Define a binary operation \square on G as follows: For i and j in G ,

$$i \square j = r,$$

where r is the remainder resulting from dividing $i \cdot j$ by p . First, we note that $i \cdot j$ is not divisible by p . Hence, $0 < r < p$, and r is an element in G . Therefore, the set G is closed under the binary operation \square , which is referred to as *modulo- p multiplication*. The set $G = \{1, 2, \dots, p - 1\}$ is a group under modulo- p multiplication. We can easily check that modulo- p multiplication is commutative and associative. The identity element is 1. The only thing left to be proved is that every element in G has an inverse. Let i be an element in G . Because p is a prime, and $i < p$, i and p must be relatively prime (i.e., i and p do not have any common factor greater than 1). It is well known that there exist two integers a and b such that

$$a \cdot i + b \cdot p = 1 \tag{2.3}$$

and a and p are relatively prime (Euclid's theorem). Rearranging (2.3), we have

$$a \cdot i = -b \cdot p + 1. \tag{2.4}$$

This says that when $a \cdot i$ is divided by p , the remainder is 1. If $0 < a < p$, a is in G , and it follows from (2.4) and the definition of modulo- p multiplication that

$$a \square i = i \square a = 1.$$

Therefore, a is the inverse of i . However, if a is not in G , we divide a by p ,

$$a = q \cdot p + r. \tag{2.5}$$

Because a and p are relatively prime, the remainder r cannot be 0, and r must be between 1 and $p - 1$. Therefore, r is in G . Now, combining (2.4) and (2.5), we obtain

$$r \cdot i = -(b + qi)p + 1.$$

TABLE 2.2: Modulo-5 multiplication.

\square	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

Therefore, $r \square i = i \square r = 1$ and r is the inverse of i . Hence, any element i in G has an inverse with respect to modulo- p multiplication. The group $G = \{1, 2, \dots, p-1\}$ under modulo- p multiplication is called a *multiplicative group*. For $p = 2$, we obtain a group $G = \{1\}$ with only one element under modulo-2 multiplication.

If p is not a prime, the set $G = \{1, 2, \dots, p-1\}$ is not a group under modulo- p multiplication (see Problem 2.3). Table 2.2 illustrates the group $G = \{1, 2, 3, 4\}$ under modulo-5 multiplication.

Let H be a nonempty subset of G . The subset H is said to be a *subgroup* of G if H is closed under the group operation of G and satisfies all the conditions of a group. For example, the set of all rational numbers is a group under real addition. The set of all integers is a subgroup of the group of rational numbers under real addition. A subgroup of G that is not identical to G is called a *proper subgroup* of G .

THEOREM 2.3 Let G be a group under the binary operation $*$. Let H be a nonempty subset of G . Then H is a subgroup of G if the following conditions hold:

- i. H is closed under the binary operation $*$.
- ii. For any element a in H , the inverse of a is also in H .

Proof. Condition (ii) says that every element of H has an inverse in H . Conditions (i) and (ii) ensure that the identity element of G is also in H . Because the elements in H are elements in G , the associative condition on $*$ holds automatically. Hence, H satisfies all the conditions of a group and is a subgroup of G . Q.E.D.

DEFINITION 2.2 Let H be a subgroup of a group G with binary operation $*$. Let a be an element of G . Then the set of elements $a * H \stackrel{\Delta}{=} \{a * h : h \in H\}$ is called a *left coset* of H ; the set of elements $H * a \stackrel{\Delta}{=} \{h * a : h \in H\}$ is called a *right coset* of H .

It is clear that if the group G is commutative, then every left coset $a * H$ is identical to every right coset $H * a$; that is, $a * H = H * a$ for any $a \in G$. In this text, we are primarily interested in commutative groups, so, we will make no further distinction between left and right cosets. We will simply refer to them as cosets.

EXAMPLE 2.4

Consider the additive group $G = \{0, 1, 2, \dots, 15\}$ under modulo-16 addition. We can readily check that $H = \{0, 4, 8, 12\}$ forms a subgroup of G . The coset $3 \boxplus H$ is

$$\begin{aligned} 3 \boxplus H &= \{3 \boxplus 0, 3 \boxplus 4, 3 \boxplus 8, 3 \boxplus 12\} \\ &= \{3, 7, 11, 15\}. \end{aligned}$$

The coset $7 \boxplus H$ is

$$\begin{aligned} 7 \boxplus H &= \{7 \boxplus 0, 7 \boxplus 4, 7 \boxplus 8, 7 \boxplus 12\} \\ &= \{7, 11, 15, 3\}. \end{aligned}$$

We find that $3 \boxplus H = 7 \boxplus H$. There are only four distinct cosets of H . Besides $3 \boxplus H$, the other three distinct cosets are

$$\begin{aligned} 0 \boxplus H &= \{0, 4, 8, 12\}, \\ 1 \boxplus H &= \{1, 5, 9, 13\}, \\ 2 \boxplus H &= \{2, 6, 10, 14\}. \end{aligned}$$

The four distinct cosets of H are disjoint, and their union forms the entire group G .

In the following theorems, we prove some important properties of cosets of a subgroup of a group.

THEOREM 2.4 Let H be a subgroup of a group G with binary operation $*$. No two elements in a coset of H are identical.

Proof. The proof is based on the fact that all the elements in the subgroup H are distinct. Consider the coset $a * H = \{a * h : h \in H\}$ with $a \in G$. Suppose two elements, say $a * h$ and $a * h'$, in $a * H$ are identical, where h and h' are two distinct elements in H . Let a^{-1} denote the inverse of a with respect to the binary operation $*$. Then,

$$\begin{aligned} a^{-1} * (a * h) &= a^{-1} * (a * h'), \\ (a^{-1} * a) * h &= (a^{-1} * a) * h', \\ e * h &= e * h', \\ h &= h'. \end{aligned}$$

This result is a contradiction to the fact that all the elements of H are distinct. Therefore, no two elements in a coset are identical. **Q.E.D.**

THEOREM 2.5 No two elements in two different cosets of a subgroup H of a group G are identical.

Proof. Let $a * H$ and $b * H$ be two distinct cosets of H , with a and b in G . Let $a * h$ and $b * h'$ be two elements in $a * H$ and $b * H$, respectively. Suppose $a * h = b * h'$. Let h^{-1} be the inverse of h . Then

$$\begin{aligned}(a * h) * h^{-1} &= (b * h') * h^{-1}, \\ a * (h * h^{-1}) &= b * (h' * h^{-1}), \\ a * e &= b * h'', \\ a &= b * h'',\end{aligned}$$

where $h'' = h' * h^{-1}$ is an element in H . The equality $a = b * h''$ implies that

$$\begin{aligned}a * H &= (b * h'') * H, \\ &= \{(b * h'') * h : h \in H\}, \\ &= \{b * (h'' * h) : h \in H\}, \\ &= \{b * h''' : h''' \in H\}, \\ &= b * H.\end{aligned}$$

This result says that $a * H$ and $b * H$ are identical, which is a contradiction to the given condition that $a * H$ and $b * H$ are two distinct cosets of H . Therefore, no two elements in two distinct cosets of H are identical. **Q.E.D.**

From Theorems 2.4 and 2.5, we obtain the following properties of cosets of a subgroup H of a group G :

- i. Every element in G appears in one and only one coset of H ;
- ii. All the distinct cosets of H are disjoint; and
- iii. The union of all the distinct cosets of H forms the group G .

Based on the preceding structural properties of cosets, we say that all the distinct cosets of a subgroup H of a group G form a *partition* of G , denoted by G/H .

THEOREM 2.6 (LAGRANGE'S THEOREM) Let G be a group of order n , and let H be a subgroup of order m . Then m divides n , and the partition G/H consists of n/m cosets of H .

Proof. It follows from Theorem 2.4 that every coset of H consists of m elements of G . Let i be the number of distinct cosets of H . Then, it follows from the preceding structural properties of cosets that $n = i \cdot m$. Therefore, m divides n , and $i = n/m$. **Q.E.D.**

2.2 FIELDS

Now, we use group concepts to introduce another algebraic system, called a *field*. Roughly speaking, a field is a set of elements in which we can perform addition, subtraction, multiplication, and division without leaving the set. Addition and multiplication must satisfy the commutative, associative, and distributive laws. A formal definition of a field is given next.

DEFINITION 2.3 Let F be a set of elements on which two binary operations, called addition “+” and multiplication “·”, are defined. The set F together with the two binary operations + and · is a field if the following conditions are satisfied:

- i. F is a commutative group under addition +. The identity element with respect to addition is called the *zero element* or the *additive identity* of F and is denoted by 0.
- ii. The set of nonzero elements in F is a commutative group under multiplication ·. The identity element with respect to multiplication is called the *unit element* or the *multiplicative identity* of F and is denoted by 1.
- iii. Multiplication is *distributive* over addition; that is, for any three elements a , b , and c in F ,

$$a \cdot (b + c) = a \cdot b + a \cdot c.$$

It follows from the definition that a field consists of at least two elements, the additive identity and the multiplicative identity. Later, we will show that a field of two elements does exist. The number of elements in a field is called the *order* of the field. A field with a finite number of elements is called a *finite field*. In a field, the additive inverse of an element a is denoted by $-a$, and the multiplicative inverse of a is denoted by a^{-1} , provided that $a \neq 0$. Subtracting a field element b from another field element a is defined as adding the additive inverse, $-b$, of b to a [i.e., $a - b \triangleq a + (-b)$]. If b is a nonzero element, dividing a by b is defined as multiplying a by the multiplicative inverse, b^{-1} , of b [i.e., $a \div b \triangleq a \cdot b^{-1}$].

A number of basic properties of fields can be derived from the definition of a field.

Property I For every element a in a field, $a \cdot 0 = 0 \cdot a = 0$.

Proof. First, we note that

$$a = a \cdot 1 = a \cdot (1 + 0) = a + a \cdot 0.$$

Adding $-a$ to both sides of the preceding equality, we have

$$\begin{aligned} -a + a &= -a + a + a \cdot 0 \\ 0 &= 0 + a \cdot 0 \\ 0 &= a \cdot 0. \end{aligned}$$

Similarly, we can show that $0 \cdot a = 0$. Therefore, we obtain $a \cdot 0 = 0 \cdot a = 0$. **Q.E.D.**

Property II For any two nonzero elements a and b in a field, $a \cdot b \neq 0$.

Proof. This is a direct consequence of the fact that the nonzero elements of a field are closed under multiplication. **Q.E.D.**

Property III $a \cdot b = 0$ and $a \neq 0$ imply that $b = 0$.

Proof. This is a direct consequence of Property II.

Q.E.D.

Property IV For any two elements a and b in a field,

$$-(a \cdot b) = (-a) \cdot b = a \cdot (-b).$$

Proof. $0 = 0 \cdot b = (a + (-a)) \cdot b = a \cdot b + (-a) \cdot b$. Therefore, $(-a) \cdot b$ must be the additive inverse of $a \cdot b$, and $-(a \cdot b) = (-a) \cdot b$. Similarly, we can prove that $-(a \cdot b) = a \cdot (-b)$.

Q.E.D.

Property V For $a \neq 0$, $a \cdot b = a \cdot c$ implies that $b = c$.

Proof. Because a is a nonzero element in the field, it has a multiplicative inverse, a^{-1} . Multiplying both sides of $a \cdot b = a \cdot c$ by a^{-1} , we obtain

$$\begin{aligned} a^{-1} \cdot (a \cdot b) &= a^{-1} \cdot (a \cdot c) \\ (a^{-1} \cdot a) \cdot b &= (a^{-1} \cdot a) \cdot c \\ 1 \cdot b &= 1 \cdot c. \end{aligned}$$

Thus, $b = c$.

Q.E.D.

We can readily verify that the set of real numbers is a field under real-number addition and multiplication. This field has an infinite number of elements. Fields with finite number of elements can be constructed and are illustrated in the next two examples and in Section 2.4.

EXAMPLE 2.5

Consider the set $\{0, 1\}$ together with modulo-2 addition and multiplication, defined in Tables 2.3 and 2.4. In Example 2.1 we showed that $\{0, 1\}$ is a commutative group under modulo-2 addition; and in Example 2.3, we showed that $\{1\}$ is a group under modulo-2 multiplication. We can easily check that modulo-2 multiplication is distributive over modulo-2 addition by simply computing $a \cdot (b + c)$ and $a \cdot b + a \cdot c$ for eight possible combinations of a, b and c ($a = 0$ or 1 , $b = 0$ or 1 , and $c = 0$ or 1). Therefore, the set $\{0, 1\}$ is a field of two elements under modulo-2 addition and modulo-2 multiplication.

The field given in Example 2.5 is usually called a *binary field* and is denoted by $GF(2)$. The binary field $GF(2)$ plays an important role in coding theory and is widely used in digital computers and digital data transmission (or storage) systems.

TABLE 2.3: Modulo-2 addition.

+	0	1
0	0	1
1	1	0

TABLE 2.4: Modulo-2 multiplication.

.	0	1
0	0	0
1	0	1

EXAMPLE 2.6

Let p be a prime. We showed in Example 2.2 that the set of integers $\{0, 1, 2, \dots, p-1\}$ is a commutative group under modulo- p addition. We also showed in Example 2.3 that the nonzero elements $\{1, 2, \dots, p-1\}$ form a commutative group under modulo- p multiplication. Following the definitions of modulo- p addition and multiplication and the fact that real-number multiplication is distributive over real-number addition, we can show that modulo- p multiplication is distributive over modulo- p addition. Therefore, the set $\{0, 1, 2, \dots, p-1\}$ is a field of order p under modulo- p addition and multiplication. Because this field is constructed from a prime, p , it is called a *prime field* and is denoted by $GF(p)$. For $p = 2$, we obtain the binary field $GF(2)$.

Let $p = 7$. Modulo-7 addition and multiplication are given by Tables 2.5 and 2.6, respectively. The set of integers $\{0, 1, 2, 3, 4, 5, 6\}$ is a field of seven elements, denoted by $GF(7)$, under modulo-7 addition and multiplication. The addition table is also used for subtraction. For example, if we want to subtract 6 from 3, we first use the addition table to find the additive inverse of 6, which is 1. Then we add 1 to 3 to obtain the result [i.e., $3 - 6 = 3 + (-6) = 3 + 1 = 4$]. For division, we use the multiplication table. Suppose that we divide 3 by 2. We first find the multiplicative inverse of 2, which is 4, and then we multiply 3 by 4 to obtain the result [i.e., $3 \div 2 = 3 \cdot (2^{-1}) = 3 \cdot 4 = 5$]. Here we have demonstrated that in a finite field, addition, subtraction, multiplication, and division can be carried out much like ordinary arithmetic, with which we are quite familiar.

In Example 2.6, we showed that, for any prime p , there exists a finite field of p elements. In fact, for any positive integer m , it is possible to extend the prime field $GF(p)$ to a field of p^m elements, which is called an *extension field* of $GF(p)$ and is denoted by $GF(p^m)$. Furthermore, it has been proved that the order of any finite field is a power of a prime. Finite fields are also called *Galois fields*, in honor of their discoverer. A large portion of algebraic coding theory, code construction, and decoding is built around finite fields. In the rest of this section and in the next two sections we examine some basic structures of finite fields, their arithmetic, and the construction of extension fields from prime fields. Our presentation is mainly descriptive, and no attempt is made to be mathematically rigorous. Because finite-field arithmetic is very similar to ordinary arithmetic, most of the rules of ordinary

TABLE 2.5: Modulo-7 addition.

+	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

TABLE 2.6: Modulo-7 multiplication.

·	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

arithmetic apply to finite-field arithmetic. Therefore, it is possible to utilize most of the techniques of algebra in the computations over finite fields.

Consider a finite field of q elements, $GF(q)$. Let us form the following sequence of sums of the unit element 1 in $GF(q)$:

$$\begin{aligned}\sum_{i=1}^1 1 &= 1, \quad \sum_{i=1}^2 1 = 1 + 1, \quad \sum_{i=1}^3 1 = 1 + 1 + 1, \dots, \\ \sum_{i=1}^k 1 &= 1 + 1 + \dots + 1(k \text{ times}), \dots\end{aligned}$$

Because the field is closed under addition, these sums must be elements in the field; and because the field has finite number of elements, these sums cannot be all distinct. Therefore, at some point in the sequence of sums, there must be a repetition; that is, there must exist two positive integers m and n such that $m < n$ and

$$\sum_{i=1}^m 1 = \sum_{i=1}^n 1.$$

This equality implies that $\sum_{i=1}^{n-m} 1 = 0$. Therefore, there must exist a *smallest positive integer* λ such that $\sum_{i=1}^\lambda 1 = 0$. This integer λ is called the *characteristic* of the field $GF(q)$. The characteristic of the binary field $GF(2)$ is 2, since $1 + 1 = 0$. The characteristic of the prime field $GF(p)$ is p , since $\sum_{i=1}^k 1 = k \neq 0$ for $1 \leq k < p$ and $\sum_{i=1}^p 1 = 0$.

THEOREM 2.7 The characteristic λ of a finite field is prime.

Proof. Suppose that λ is not a prime and is equal to the product of two smaller integers k and m (i.e., $\lambda = km$). Because the field is closed under multiplication,

$$\left(\sum_{i=1}^k 1\right) \cdot \left(\sum_{i=1}^m 1\right)$$

is also a field element. It follows from the distributive law that

$$\left(\sum_{i=1}^k 1\right) \cdot \left(\sum_{i=1}^m 1\right) = \sum_{i=1}^{km} 1.$$

Because $\sum_{i=1}^{km} 1 = 0$, then either $\sum_{i=1}^k 1 = 0$ or $\sum_{i=1}^m 1 = 0$; however, this contradicts the definition that λ is the smallest positive integer such that $\sum_{i=1}^\lambda 1 = 0$. Therefore, we conclude that λ is prime. **Q.E.D.**

It follows from the definition of the characteristic of a finite field that for any two distinct positive integers k and m less than λ ,

$$\sum_{i=1}^k 1 \neq \sum_{i=1}^m 1.$$

Suppose that $\sum_{i=1}^k 1 = \sum_{i=1}^m 1$. Then, we have

$$\sum_{i=1}^{m-k} 1 = 0$$

(assuming that $m > k$); however, this is impossible, since $m - k < \lambda$. Therefore, the sums

$$1 = \sum_{i=1}^1 1, \quad \sum_{i=1}^2 1, \quad \sum_{i=1}^3 1, \quad \dots, \quad \sum_{i=1}^{\lambda-1} 1, \quad \sum_{i=1}^{\lambda} 1 = 0$$

are λ distinct elements in $GF(q)$. In fact, this set of sums itself is a field of λ elements, $GF(\lambda)$, under the addition and multiplication of $GF(q)$ (see Problem 2.7). Because $GF(\lambda)$ is a subset of $GF(q)$, $GF(\lambda)$ is called a *subfield* of $GF(q)$. Therefore, any finite field $GF(q)$ of characteristic λ contains a subfield of λ elements. It can be proved that if $q \neq \lambda$, then q is a power of λ .

Now, let a be a nonzero element in $GF(q)$. Since the set of nonzero elements of $GF(q)$ is closed under multiplication, the following powers of a ,

$$a^1 = a, \quad a^2 = a \cdot a, \quad a^3 = a \cdot a \cdot a, \dots$$

must also be nonzero elements in $GF(q)$. Because $GF(q)$ has only a finite number of elements, the powers of a given cannot all be distinct. Therefore, at some point in the sequence of powers of a there must be a repetition; that is, there must exist two positive integers k and m such that $m > k$ and $a^k = a^m$. Let a^{-1} be the multiplicative inverse of a . Then $(a^{-1})^k = a^{-k}$ is the multiplicative inverse of a^k . Multiplying both sides of $a^k = a^m$ by a^{-k} , we obtain

$$1 = a^{m-k}.$$

This equality implies that there must exist a *smallest positive integer* n such that $a^n = 1$. This integer n is called the *order* of the field element a . Therefore, the sequence a^1, a^2, a^3, \dots repeats itself after $a^n = 1$. Also, the powers $a^1, a^2, \dots, a^{n-1}, a^n = 1$ are all distinct. In fact, they form a group under the multiplication of $GF(q)$. First, we see that they contain the unit element 1. Consider $a^i \cdot a^j$. If $i + j \leq n$,

$$a^i \cdot a^j = a^{i+j}.$$

If $i + j > n$, we have $i + j = n + r$, where $0 < r \leq n$. Hence,

$$a^i \cdot a^j = a^{i+j} = a^n \cdot a^r = a^r.$$

Therefore, the powers $a^1, a^2, \dots, a^{n-1}, a^n = 1$ are closed under the multiplication of $GF(q)$. For $1 \leq i < n$, a^{n-i} is the multiplicative inverse of a^i . Because the powers of a are nonzero elements in $GF(q)$, they satisfy the associative and commutative laws. Therefore, we conclude that $a^n = 1, a^1, a^2, \dots, a^{n-1}$ form a commutative group under the multiplication of $GF(q)$. A group is said to be *cyclic* if there exists an element in the group whose powers constitute the whole group.

THEOREM 2.8 Let a be a nonzero element of a finite field $GF(q)$. Then $a^{q-1} = 1$.

Proof. Let b_1, b_2, \dots, b_{q-1} be the $q - 1$ nonzero elements of $GF(q)$. Clearly, the $q - 1$ elements $a \cdot b_1, a \cdot b_2, \dots, a \cdot b_{q-1}$ are nonzero and distinct. Thus,

$$(a \cdot b_1) \cdot (a \cdot b_2) \cdots \cdots (a \cdot b_{q-1}) = b_1 \cdot b_2 \cdots \cdots b_{q-1}$$

$$a^{q-1} \cdot (b_1 \cdot b_2 \cdots \cdots b_{q-1}) = b_1 \cdot b_2 \cdots \cdots b_{q-1}.$$

Since $a \neq 0$ and $(b_1 \cdot b_2 \cdots b_{q-1}) \neq 0$, we must have $a^{q-1} = 1$.

Q.E.D.

THEOREM 2.9 Let a be a nonzero element in a finite field $GF(q)$. Let n be the order of a . Then n divides $q - 1$.

Proof. Suppose that n does not divide $q - 1$. Dividing $q - 1$ by n , we obtain

$$q - 1 = kn + r,$$

where $0 < r < n$. Then

$$a^{q-1} = a^{kn+r} = a^{kn} \cdot a^r = (a^n)^k \cdot a^r.$$

Because $a^{q-1} = 1$, and $a^n = 1$, we must have $a^r = 1$. This is impossible, since $0 < r < n$, and n is the smallest integer such that $a^n = 1$. Therefore, n must divide $q - 1$.

Q.E.D.

In a finite field $GF(q)$, a nonzero element a is said to be *primitive* if the order of a is $q - 1$. Therefore, the powers of a primitive element generate all the nonzero elements of $GF(q)$. Every finite field has a primitive element (see Problem 2.8).

Consider the prime field $GF(7)$ illustrated by Tables 2.5 and 2.6. The characteristic of this field is 7. If we take the powers of the integer 3 in $GF(7)$ using the multiplication table, we obtain

$$3^1 = 3, \quad 3^2 = 3 \cdot 3 = 2, \quad 3^3 = 3 \cdot 3^2 = 6,$$

$$3^4 = 3 \cdot 3^3 = 4, \quad 3^5 = 3 \cdot 3^4 = 5, \quad 3^6 = 3 \cdot 3^5 = 1.$$

Therefore, the order of the integer 3 is 6, and the integer 3 is a primitive element of $GF(7)$. The powers of the integer 4 in $GF(7)$ are

$$4^1 = 4, \quad 4^2 = 4 \cdot 4 = 2, \quad 4^3 = 4 \cdot 4^2 = 1.$$

Clearly, the order of the integer 4 is 3, which is a factor of 6.

2.3 BINARY FIELD ARITHMETIC

In general, we can construct codes with symbols from any Galois field $GF(q)$, where q is either a prime p or a power of p ; however, codes with symbols from the binary field $GF(2)$ or its extension $GF(2^m)$ are most widely used in digital data transmission and storage systems because information in these systems is universally coded in binary form for practical reasons. In this text we are concerned only with binary codes and codes with symbols from the field $GF(2^m)$. Most of the results presented in this text can be generalized to codes with symbols from any finite field $GF(q)$.

with $q \neq 2$ or 2^m . In this section we discuss arithmetic over the binary field $GF(2)$, which will be used in the remainder text of this book.

In binary arithmetic we use modulo-2 addition and multiplication, which are defined by Tables 2.3 and 2.4, respectively. This arithmetic is actually equivalent to ordinary arithmetic, except that we consider 2 to be equal to 0 (i.e., $1 + 1 = 2 = 0$). Note that since $1 + 1 = 0$, $1 = -1$. Hence, in binary arithmetic, subtraction is the same as addition. To illustrate how the ideas of ordinary algebra can be used with binary arithmetic, we consider the following set of equations:

$$X + Y = 1,$$

$$X + Z = 0,$$

$$X + Y + Z = 1.$$

These can be solved by adding the first equation to the third, giving $Z = 0$. Then, from the second equation, since $Z = 0$ and $X + Z = 0$, we obtain $X = 0$. From the first equation, since $X = 0$ and $X + Y = 1$, we have $Y = 1$. We can substitute these solutions into the original set of equations and verify that they are correct.

Because we were able to solve the preceding equations, they must be linearly independent, and the determinant of the coefficients on the left side must be nonzero. If the determinant is nonzero, it must be 1. This result can be verified as follows.

$$\begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix} = 1 \cdot \begin{vmatrix} 0 & 1 \\ 1 & 1 \end{vmatrix} - 1 \cdot \begin{vmatrix} 1 & 1 \\ 1 & 1 \end{vmatrix} + 0 \cdot \begin{vmatrix} 1 & 0 \\ 1 & 1 \end{vmatrix}$$

$$= 1 \cdot 1 - 1 \cdot 0 + 0 \cdot 1 = 1.$$

We could have solved the equations by Cramer's rule:

$$X = \frac{\begin{vmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{vmatrix}} = \frac{0}{1} = 0, \quad Y = \frac{\begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix}} = \frac{1}{1} = 1, \quad Z = \frac{\begin{vmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix}} = \frac{0}{1} = 0.$$

Next, we consider computations with polynomials whose coefficients are from the binary field $GF(2)$. A polynomial $f(X)$ with one variable X and with coefficients from $GF(2)$ is of the following form:

$$f(X) = f_0 + f_1 X + f_2 X^2 + \cdots + f_n X^n,$$

where $f_i = 0$ or 1 for $0 \leq i \leq n$. The *degree* of a polynomial is the largest power of X with a nonzero coefficient. For the preceding polynomial, if $f_n = 1$, $f(X)$ is a polynomial of degree n ; if $f_n = 0$, $f(X)$ is a polynomial of degree less than n . The degree of $f(X) = f_0$ is zero. In the following discussion we use the phrase “a polynomial over $GF(2)$ ” to mean “a polynomial with coefficients from $GF(2)$.”

There are two polynomials over $GF(2)$ with degree 1: X and $1 + X$. There are four polynomials over $GF(2)$ with degree 2: X^2 , $1 + X^2$, $X + X^2$, and $1 + X + X^2$. In general, there are 2^n polynomials over $GF(2)$ with degree n .

Polynomials over $GF(2)$ can be added (or subtracted), multiplied, and divided in the usual way. Let

$$g(X) = g_0 + g_1X + g_2X^2 + \cdots + g_mX^m$$

be another polynomial over $GF(2)$. To add $f(X)$ and $g(X)$, we simply add the coefficients of the same power of X in $f(X)$ and $g(X)$ as follows (assuming that $m \leq n$):

$$\begin{aligned} f(X) + g(X) &= (f_0 + g_0) + (f_1 + g_1)X + \cdots \\ &\quad + (f_m + g_m)X^m + f_{m+1}X^{m+1} + \cdots + f_nX^n, \end{aligned}$$

where $f_i + g_i$ is carried out in modulo-2 addition. For example, adding $a(X) = 1 + X + X^3 + X^5$ and $b(X) = 1 + X^2 + X^3 + X^4 + X^7$, we obtain the following sum:

$$\begin{aligned} a(X) + b(X) &= (1 + 1) + X + X^2 + (1 + 1)X^3 + X^4 + X^5 + X^7 \\ &= X + X^2 + X^4 + X^5 + X^7. \end{aligned}$$

When we multiply $f(X)$ and $g(X)$, we obtain the following product:

$$f(X) \cdot g(X) = c_0 + c_1X + c_2X^2 + \cdots + c_{n+m}X^{n+m},$$

where

$$\begin{aligned} c_0 &= f_0g_0, \\ c_1 &= f_0g_1 + f_1g_0, \\ c_2 &= f_0g_2 + f_1g_1 + f_2g_0, \\ &\vdots \\ c_i &= f_0g_i + f_1g_{i-1} + f_2g_{i-2} + \cdots + f_ig_0, \\ &\vdots \\ c_{n+m} &= f_ng_m. \end{aligned} \tag{2.6}$$

(Multiplication and addition of coefficients are modulo-2.) It is clear from (2.6) that if $g(X) = 0$, then

$$f(X) \cdot 0 = 0. \tag{2.7}$$

We can readily verify that the polynomials over $GF(2)$ satisfy the following conditions:

i. Commutative:

$$a(X) + b(X) = b(X) + a(X),$$

$$a(X) \cdot b(X) = b(X) \cdot a(X).$$

ii. Associative:

$$\begin{aligned} a(X) + [b(X) + c(X)] &= [a(X) + b(X)] + c(X), \\ a(X) \cdot [b(X) \cdot c(X)] &= [a(X) \cdot b(X)] \cdot c(X). \end{aligned}$$

iii. Distributive:

$$a(X) \cdot [b(X) + c(X)] = [a(X) \cdot b(X)] + [a(X) \cdot c(X)]. \quad (2.8)$$

Suppose that the degree of $g(X)$ is *not* zero. When $f(X)$ is divided by $g(X)$, we obtain a unique pair of polynomials over $GF(2)$ — $q(X)$, called the quotient, and $r(X)$, called the remainder—such that

$$f(X) = q(X)g(X) + r(X),$$

and the degree of $r(X)$ is less than that of $g(X)$. This expression is known as *Euclid's division algorithm*. As an example, we divide $f(X) = 1 + X + X^4 + X^5 + X^6$ by $g(X) = 1 + X + X^3$. Using the long-division technique, we have

$$\begin{array}{r} X^3 + X^2 \quad (\text{quotient}) \\ \hline X^3 + X + 1 | X^6 + X^5 + X^4 \quad + X + 1 \\ \quad X^6 \quad + X^4 + X^3 \\ \hline \quad X^5 \quad + X^3 \quad + X + 1 \\ \quad X^5 \quad + X^3 \quad + X^2 \\ \hline \quad \quad \quad X^2 + X + 1 \quad (\text{remainder}). \end{array}$$

We can easily verify that

$$X^6 + X^5 + X^4 + X + 1 = (X^3 + X^2)(X^3 + X + 1) + X^2 + X + 1.$$

When $f(X)$ is divided by $g(X)$, if the remainder $r(X)$ is identical to zero [$r(X) = 0$], we say that $f(X)$ is divisible by $g(X)$, and $g(X)$ is a factor of $f(X)$.

For real numbers, if a is a *root* of a polynomial $f(X)$ [i.e., $f(a) = 0$], $f(X)$ is divisible by $X - a$. (This fact follows from Euclid's division algorithm.) This statement is still true for $f(X)$ over $GF(2)$. For example, let $f(X) = 1 + X^2 + X^3 + X^4$. Substituting $X = 1$, we obtain

$$f(1) = 1 + 1^2 + 1^3 + 1^4 = 1 + 1 + 1 + 1 = 0.$$

Thus, $f(X)$ has 1 as a root, and it should be divisible by $X + 1$, as shown:

$$\begin{array}{r} X^3 + X + 1 \\ \hline X + 1 | X^4 + X^3 + X^2 \quad + 1 \\ \quad X^4 + X^3 \\ \hline \quad \quad \quad X^2 \quad + 1 \\ \quad \quad \quad X^2 + X \\ \hline \quad \quad \quad X + 1 \\ \quad \quad \quad X + 1 \\ \hline \quad \quad \quad 0. \end{array}$$

For a polynomial $f(X)$ over $GF(2)$, if the polynomial has an even number of terms, it is divisible by $X + 1$. A polynomial $p(X)$ over $GF(2)$ of degree m is said to be *irreducible* over $GF(2)$ if $p(X)$ is not divisible by any polynomial over $GF(2)$ of degree less than m but greater than zero. Among the four polynomials of degree 2, X^2 , $X^2 + 1$, and $X^2 + X$ are not irreducible, since they are either divisible by X or $X + 1$; however, $X^2 + X + 1$ does not have either 0 or 1 as a root and so is not divisible by any polynomial of degree 1. Therefore, $X^2 + X + 1$ is an irreducible polynomial of degree 2. The polynomial $X^3 + X + 1$ is an irreducible polynomial of degree 3. First, we note that $X^3 + X + 1$ does not have either 0 or 1 as a root. Therefore, $X^3 + X + 1$ is not divisible by X or $X + 1$. Because the polynomial is not divisible by any polynomial of degree 1, it cannot be divisible by a polynomial of degree 2. Consequently, $X^3 + X + 1$ is irreducible over $GF(2)$. We may verify that $X^4 + X + 1$ is an irreducible polynomial of degree 4. It has been proved that for any $m \geq 1$ there exists an irreducible polynomial of degree m . An important theorem regarding irreducible polynomials over $GF(2)$ is given next without a proof.

THEOREM 2.10 Any irreducible polynomial over $GF(2)$ of degree m divides $X^{2^m - 1} + 1$.

As an example of Theorem 2.10, we can check that $X^3 + X + 1$ divides $X^{2^3 - 1} + 1 = X^7 + 1$:

$$\begin{array}{r}
 & X^4 + X^2 + X + 1 \\
 \hline
 X^3 + X + 1 | & X^7 & & + 1 \\
 & X^7 & + X^5 + X^4 \\
 \hline
 & X^5 + X^4 & & + 1 \\
 & X^5 & + X^3 + X^2 & \\
 \hline
 & X^4 + X^3 + X^2 & + 1 \\
 & X^4 & + X^2 + X \\
 \hline
 & X^3 & + X + 1 \\
 & X^3 & + X + 1 \\
 \hline
 & 0.
 \end{array}$$

An irreducible polynomial $p(X)$ of degree m is said to be *primitive* if the smallest positive integer n for which $p(X)$ divides $X^n + 1$ is $n = 2^m - 1$. We may check that $p(X) = X^4 + X + 1$ divides $X^{15} + 1$ but does not divide any $X^n + 1$ for $1 \leq n < 15$. Hence, $X^4 + X + 1$ is a primitive polynomial. The polynomial $X^4 + X^3 + X^2 + X + 1$ is irreducible but it is not primitive, since it divides $X^5 + 1$. It is not easy to recognize a primitive polynomial; however, there are tables of irreducible polynomials in which primitive polynomials are indicated [6, 8]. For a given m , there may be more than one primitive polynomial of degree m . A list of primitive polynomials is given in Table 2.7. For each degree m , we list only a primitive polynomial with the smallest number of terms.

TABLE 2.7: List of primitive polynomials.

m	m
3	$1 + X + X^3$
4	$1 + X + X^4$
5	$1 + X^2 + X^5$
6	$1 + X + X^6$
7	$1 + X^3 + X^7$
8	$1 + X^2 + X^3 + X^4 + X^8$
9	$1 + X^4 + X^9$
10	$1 + X^3 + X^{10}$
11	$1 + X^2 + X^{11}$
12	$1 + X + X^4 + X^6 + X^{12}$
13	$1 + X + X^3 + X^4 + X^{13}$
14	$1 + X + X^6 + X^{10} + X^{14}$
15	$1 + X + X^{15}$
16	$1 + X + X^3 + X^{12} + X^{16}$
17	$1 + X^3 + X^{17}$
18	$1 + X^7 + X^{18}$
19	$1 + X + X^2 + X^5 + X^{19}$
20	$1 + X^3 + X^{20}$
21	$1 + X^2 + X^{21}$
22	$1 + X + X^{22}$
23	$1 + X^5 + X^{23}$
24	$1 + X + X^2 + X^7 + X^{24}$

Before leaving this section, we derive another useful property of polynomials over $GF(2)$. Consider

$$\begin{aligned}
 f^2(X) &= (f_0 + f_1X + \cdots + f_nX^n)^2 \\
 &= [f_0 + (f_1X + f_2X^2 + \cdots + f_nX^n)]^2 \\
 &= f_0^2 + f_0 \cdot (f_1X + f_2X^2 + \cdots + f_nX^n) \\
 &\quad + f_0 \cdot (f_1X + f_2X^2 + \cdots + f_nX^n) + (f_1X + f_2X^2 + \cdots + f_nX^n)^2 \\
 &= f_0^2 + (f_1X + f_2X^2 + \cdots + f_nX^n)^2.
 \end{aligned}$$

Expanding the preceding equation repeatedly, we eventually obtain

$$f^2(X) = f_0^2 + (f_1X)^2 + (f_2X^2)^2 + \cdots + (f_nX^n)^2.$$

Since $f_i = 0$ or 1 , $f_i^2 = f_i$. Hence, we have

$$\begin{aligned}
 f^2(X) &= f_0 + f_1X^2 + f_2(X^2)^2 + \cdots + f_n(X^2)^n \\
 &= f(X^2).
 \end{aligned} \tag{2.9}$$

It follows from (2.9) that, for any $i \geq 0$,

$$[f(X)]^{2^i} = f(X^{2^i}). \tag{2.10}$$

2.4 CONSTRUCTION OF GALOIS FIELD $GF(2^m)$

In this section we present a method for constructing the Galois field of 2^m elements ($m > 1$) from the binary field $GF(2)$. We begin with the two elements 0 and 1 from

$GF(2)$ and a new symbol α . Then, we define a multiplication “.” to introduce a sequence of powers of α as follows:

$$\begin{aligned}
 0 \cdot 0 &= 0, \\
 0 \cdot 1 &= 1 \cdot 0 = 0, \\
 1 \cdot 1 &= 1, \\
 0 \cdot \alpha &= \alpha \cdot 0 = 0, \\
 1 \cdot \alpha &= \alpha \cdot 1 = \alpha, \\
 \alpha^2 &= \alpha \cdot \alpha, \\
 \alpha^3 &= \alpha \cdot \alpha \cdot \alpha, \\
 &\vdots \\
 \alpha^j &= \alpha \cdot \alpha \cdots \alpha \text{ (} j \text{ times)}, \\
 &\vdots
 \end{aligned} \tag{2.11}$$

It follows from the preceding definition of multiplication that

$$\begin{aligned}
 0 \cdot \alpha^j &= \alpha^j \cdot 0 = 0, \\
 1 \cdot \alpha^j &= \alpha^j \cdot 1 = \alpha^j, \\
 \alpha^i \cdot \alpha^j &= \alpha^j \cdot \alpha^i = \alpha^{i+j}.
 \end{aligned} \tag{2.12}$$

Now, we have the following set of elements on which a multiplication operation “.” is defined:

$$F = \{0, 1, \alpha, \alpha^2, \dots, \alpha^j, \dots\}.$$

The element 1 is sometimes denoted by α^0 .

Next, we put a condition on the element α so that the set F contains only 2^m elements and is closed under the multiplication “.” defined by (2.11). Let $p(X)$ be a primitive polynomial of degree m over $GF(2)$. We assume that $p(\alpha) = 0$ (i.e., α is a root of $p(X)$). Since $p(X)$ divides $X^{2^m-1} + 1$ (Theorem 2.10) we have

$$X^{2^m-1} + 1 = q(X)p(X). \tag{2.13}$$

If we replace X with α in (2.13), we obtain

$$\alpha^{2^m-1} + 1 = q(\alpha)p(\alpha).$$

Because $p(\alpha) = 0$, we have

$$\alpha^{2^m-1} + 1 = q(\alpha) \cdot 0.$$

If we regard $q(\alpha)$ as a polynomial of α over $GF(2)$, it follows from (2.7) that $q(\alpha) \cdot 0 = 0$. As a result, we obtain the following equality:

$$\alpha^{2^m-1} + 1 = 0.$$

Adding 1 to both sides of $\alpha^{2^m-1} + 1 = 0$ (using modulo-2 addition), we obtain the following equality:

$$\alpha^{2^m-1} = 1. \quad (2.14)$$

Therefore, under the condition that $p(\alpha) = 0$, the set F becomes finite and contains the following elements:

$$F^* = \{0, 1, \alpha, \alpha^2, \dots, \alpha^{2^m-2}\}.$$

The nonzero elements of F^* are closed under the multiplication operation “.” defined by (2.11). To see this, let i and j be two integers such that $0 \leq i, j < 2^m - 1$. If $i + j < 2^m - 1$, then $\alpha^i \cdot \alpha^j = \alpha^{i+j}$, which is obviously a nonzero element in F^* . If $i + j \geq 2^m - 1$, we can express $i + j$ as follows: $i + j = (2^m - 1) + r$, where $0 \leq r < 2^m - 1$. Then,

$$\alpha^i \cdot \alpha^j = \alpha^{i+j} = \alpha^{(2^m-1)+r} = \alpha^{2^m-1} \cdot \alpha^r = \alpha^r,$$

which is also a nonzero element in F^* . Hence, we conclude that the nonzero elements of F^* are closed under the multiplication “.” defined by (2.11). In fact, these nonzero elements form a commutative group under “.”. First, we see that the element 1 is the unit element. From (2.11) and (2.12) we see readily that the multiplication operation “.” is commutative and associative. For $0 < i < 2^m - 1$, α^{2^m-i-1} is the multiplicative inverse of α^i , since

$$\alpha^{2^m-i-1} \cdot \alpha^i = \alpha^{2^m-1} = 1.$$

(Note that $\alpha^0 = \alpha^{2^m-1} = 1$.) It will be clear in the discussion that follows that $1, \alpha, \alpha^2, \dots, \alpha^{2^m-2}$ represent $2^m - 1$ distinct elements. Therefore, the nonzero elements of F^* form a commutative group of order $2^m - 1$ under the multiplication operation “.” defined by (2.11).

Our next step is to define an addition operation “+” on F^* so that F^* forms a commutative group under “+”. For $0 \leq i < 2^m - 1$, we divide the polynomial X^i by $p(X)$ and obtain the following:

$$X^i = q_i(X)p(X) + a_i(X), \quad (2.15)$$

where $q_i(X)$ and $a_i(X)$ are the quotient and the remainder, respectively. The remainder $a_i(X)$ is a polynomial of degree $m - 1$ or less over $GF(2)$ and is of the following form:

$$a_i(X) = a_{i0} + a_{i1}X + a_{i2}X^2 + \dots + a_{i,m-1}X^{m-1}.$$

Because X and $p(X)$ are relatively prime (i.e., they do not have any common factor except 1), X^i is not divisible by $p(X)$. Therefore, for any $i \geq 0$,

$$a_i(X) \neq 0. \quad (2.16)$$

For $0 \leq i, j < 2^m - 1$, and $i \neq j$, we can also show that

$$a_i(X) \neq a_j(X). \quad (2.17)$$

Suppose that $a_i(X) = a_j(X)$. Then, it follows from (2.15) that

$$\begin{aligned} X^i + X^j &= [q_i(X) + q_j(X)]p(X) + a_i(X) + a_j(X) \\ &= [q_i(X) + q_j(X)]p(X). \end{aligned}$$

This implies that $p(X)$ divides $X^i + X^j = X^i(1 + X^{j-i})$ (assuming that $j > i$). Because X^i and $p(X)$ are relatively prime, $p(X)$ must divide $X^{j-i} + 1$; however, this is impossible, since $j - i < 2^m - 1$, and $p(X)$ is a primitive polynomial of degree m that does not divide $X^n + 1$ for $n < 2^m - 1$. Therefore, our hypothesis that $a_i(X) = a_j(X)$ is invalid. As a result, for $0 \leq i, j < 2^m - 1$, and $i \neq j$, we must have $a_i(X) \neq a_j(X)$. Hence, for $i = 0, 1, 2, \dots, 2^m - 2$, we obtain $2^m - 1$ distinct nonzero polynomials $a_i(X)$ of degree $m - 1$ or less. Now, replacing X with α in (2.15) and using the equality that $q_i(\alpha) \cdot 0 = 0$ [see (2.7)], we obtain the following polynomial expression for α^i :

$$\alpha^i = a_i(\alpha) = a_{i0} + a_{i1}\alpha + a_{i2}\alpha^2 + \dots + a_{i,m-1}\alpha^{m-1}. \quad (2.18)$$

From (2.16), (2.17), and (2.18), we see that the $2^m - 1$ nonzero elements, $\alpha^0, \alpha^1, \dots, \alpha^{2^m-2}$ in F^* , are represented by $2^m - 1$ distinct nonzero polynomials of α over $GF(2)$ with degree $m - 1$ or less. The zero element 0 in F^* may be represented by the zero polynomial. As a result, the 2^m elements in F^* are represented by 2^m distinct polynomials of α over $GF(2)$ with degree $m - 1$ or less and are regarded as 2^m distinct elements.

Now, we define an addition “+” on F^* as follows:

$$0 + 0 = 0 \quad (2.19a)$$

and, for $0 \leq i, j < 2^m - 1$,

$$0 + \alpha^i = \alpha^i + 0 = \alpha^i. \quad (2.19b)$$

$$\alpha^i + \alpha^j = (a_{i0} + a_{i1}\alpha + \dots + a_{i,m-1}\alpha^{m-1}) + (a_{j0} + a_{j1}\alpha + \dots + a_{j,m-1}\alpha^{m-1})$$

$$= (a_{i0} + a_{j0}) + (a_{i1} + a_{j1})\alpha + \dots + (a_{i,m-1} + a_{j,m-1})\alpha^{m-1}, \quad (2.19c)$$

where $a_{i,k} + a_{j,k}$ is carried out in modulo-2 addition for $0 \leq k < m$. From (2.19c) we see that, for $i = j$,

$$\alpha^i + \alpha^i = 0 \quad (2.20)$$

and for $i \neq j$,

$$(a_{i0} + a_{j0}) + (a_{i1} + a_{j1})\alpha + \dots + (a_{i,m-1} + a_{j,m-1})\alpha^{m-1}$$

is nonzero and must be the polynomial expression for some α^k in F^* . Hence, the set F^* is closed under the addition “+” defined by (2.19). We can immediately verify that F^* is a commutative group under “+”. First, we see that 0 is the additive identity. Because modulo-2 addition is commutative and associative, the addition defined on F^* is also commutative and associative. From (2.19a) and (2.20) we see that the additive inverse of any element in F^* is itself.

Up to this point we have shown that the set $F^* = \{0, 1, \alpha, \alpha^2, \dots, \alpha^{2^m-2}\}$ is a commutative group under an addition operation “+”, and the nonzero elements of

F^* form a commutative group under a multiplication operation “.”. Using the polynomial representation for the elements in F^* and (2.8) (polynomial multiplication satisfies distributive law), we readily see that the multiplication on F^* is distributive over the addition on F^* . Therefore, the set $F^* = \{0, 1, \alpha, \alpha^2, \dots, \alpha^{2^m-2}\}$ is a Galois field of 2^m elements, $GF(2^m)$. We notice that the addition and multiplication defined on $F^* = GF(2^m)$ imply modulo-2 addition and multiplication. Hence, the subset $\{0, 1\}$ forms a subfield of $GF(2^m)$ [i.e., $GF(2)$ is a subfield of $GF(2^m)$]. The binary field $GF(2)$ is usually called the *ground* field of $GF(2^m)$. The characteristic of $GF(2^m)$ is 2.

In our process of constructing $GF(2^m)$ from $GF(2)$, we have developed two representations for the nonzero elements of $GF(2^m)$: the power representation and the polynomial representation. The power representation is convenient for multiplication, and the polynomial representation is convenient for addition.

EXAMPLE 2.7

Let $m = 4$. The polynomial $p(X) = 1 + X + X^4$ is a primitive polynomial over $GF(2)$. Set $p(\alpha) = 1 + \alpha + \alpha^4 = 0$. Then, $\alpha^4 = 1 + \alpha$. Using this relation, we can construct $GF(2^4)$. The elements of $GF(2^4)$ are given in Table 2.8. The identity $\alpha^4 = 1 + \alpha$ is used repeatedly to form the polynomial representations for the elements of $GF(2^4)$. For example,

$$\alpha^5 = \alpha \cdot \alpha^4 = \alpha(1 + \alpha) = \alpha + \alpha^2,$$

$$\alpha^6 = \alpha \cdot \alpha^5 = \alpha(\alpha + \alpha^2) = \alpha^2 + \alpha^3,$$

$$\alpha^7 = \alpha \cdot \alpha^6 = \alpha(\alpha^2 + \alpha^3) = \alpha^3 + \alpha^4 = \alpha^3 + 1 + \alpha = 1 + \alpha + \alpha^3.$$

To multiply two elements α^i and α^j , we simply add their exponents and use the fact that $\alpha^{15} = 1$. For example, $\alpha^5 \cdot \alpha^7 = \alpha^{12}$, and $\alpha^{12} \cdot \alpha^7 = \alpha^{19} = \alpha^4$. Dividing α^j by α^i , we simply multiply α^j by the multiplicative inverse α^{15-i} of α^i . For example, $\alpha^4/\alpha^{12} = \alpha^4 \cdot \alpha^3 = \alpha^7$, and $\alpha^{12}/\alpha^5 = \alpha^{12} \cdot \alpha^{10} = \alpha^{22} = \alpha^7$. To add α^i and α^j , we use their polynomial representations given in Table 2.8. Thus,

$$\alpha^5 + \alpha^7 = (\alpha + \alpha^2) + (1 + \alpha + \alpha^3) = 1 + \alpha^2 + \alpha^3 = \alpha^{13},$$

$$1 + \alpha^5 + \alpha^{10} = 1 + (\alpha + \alpha^2) + (1 + \alpha + \alpha^2) = 0.$$

There is another useful representation for the field elements in $GF(2^m)$. Let $a_0 + a_1\alpha + a_2\alpha^2 + \dots + a_{m-1}\alpha^{m-1}$ be the polynomial representation of a field element β . Then, we can represent β by an ordered sequence of m components called an *m-tuple*, as follows:

$$(a_0, a_1, a_2, \dots, a_{m-1}),$$

where the m components are simply the m coefficients of the polynomial representation of β . Clearly, we see that there is one-to-one correspondence between this *m-tuple* and the polynomial representation of β . The zero element 0 of $GF(2^m)$ is represented by the zero *m-tuple* $(0, 0, \dots, 0)$. Let $(b_0, b_1, \dots, b_{m-1})$ be the *m-tuple*

TABLE 2.8: Three representations for the elements of $GF(2^4)$ generated by $p(X) = 1 + X + X^4$.

Power representation	Polynomial representation	4-Tuple representation
0	0	(0 0 0 0)
1	1	(1 0 0 0)
α	α	(0 1 0 0)
α^2	α^2	(0 0 1 0)
α^3	α^3	(0 0 0 1)
α^4	$1 + \alpha$	(1 1 0 0)
α^5	$\alpha + \alpha^2$	(0 1 1 0)
α^6	$\alpha^2 + \alpha^3$	(0 0 1 1)
α^7	$1 + \alpha + \alpha^3$	(1 1 0 1)
α^8	$1 + \alpha^2$	(1 0 1 0)
α^9	$\alpha + \alpha^3$	(0 1 0 1)
α^{10}	$1 + \alpha + \alpha^2$	(1 1 1 0)
α^{11}	$\alpha + \alpha^2 + \alpha^3$	(0 1 1 1)
α^{12}	$1 + \alpha + \alpha^2 + \alpha^3$	(1 1 1 1)
α^{13}	$1 + \alpha^2 + \alpha^3$	(1 0 1 1)
α^{14}	$1 + \alpha^3$	(1 0 0 1)

representation of γ in $GF(2^m)$. Adding β and γ , we simply add the corresponding components of their m -tuple representations as follows:

$$(a_0 + b_0, a_1 + b_1, \dots, a_{m-1} + b_{m-1}),$$

where $a_i + b_i$ is carried out in modulo-2 addition. Obviously, the components of the resultant m -tuple are the coefficients of the polynomial representation for $\beta + \gamma$. All three representations for the elements of $GF(2^4)$ are given in Table 2.8.

Galois fields of 2^m elements with $m = 3$ to 10 are given in Appendix A.

2.5 BASIC PROPERTIES OF A GALOIS FIELD $GF(2^m)$

In ordinary algebra we often see that a polynomial with real coefficients has roots not from the field of real numbers but from the field of complex numbers that contains the field of real numbers as a subfield. For example, the polynomial $X^2 + 6X + 25$ does not have roots from the field of real numbers but has two complex-conjugate roots, $-3 + 4i$ and $-3 - 4i$, where $i = \sqrt{-1}$. This situation is also true for polynomials with coefficients from $GF(2)$. In this case, a polynomial with coefficients from $GF(2)$ may not have roots from $GF(2)$ but has roots from an extension field of $GF(2)$. For example, $X^4 + X^3 + 1$ is irreducible over $GF(2)$ and therefore it does not have roots from $GF(2)$; however, it has four roots from the field $GF(2^4)$. If we substitute the elements of $GF(2^4)$ given by Table 2.8 into $X^4 + X^3 + 1$, we find that $\alpha^7, \alpha^{11}, \alpha^{13}$, and α^{14} are the roots of $X^4 + X^3 + 1$. We may verify this result as follows:

$$(\alpha^7)^4 + (\alpha^7)^3 + 1 = \alpha^{28} + \alpha^{21} + 1 = (1 + \alpha^2 + \alpha^3) + (\alpha^2 + \alpha^3) + 1 = 0.$$

Indeed, α^7 is a root for $X^4 + X^3 + 1$. Similarly, we may verify that α^{11} , α^{13} , and α^{14} are the other three roots. Since α^7 , α^{11} , α^{13} , and α^{14} are all roots of $X^4 + X^3 + 1$, then $(X + \alpha^7)(X + \alpha^{11})(X + \alpha^{13})(X + \alpha^{14})$ must be equal to $X^4 + X^3 + 1$. To see this, we multiply out the preceding product using Table 2.8:

$$\begin{aligned} & (X + \alpha^7)(X + \alpha^{11})(X + \alpha^{13})(X + \alpha^{14}) \\ &= [X^2 + (\alpha^7 + \alpha^{11})X + \alpha^{18}][X^2 + (\alpha^{13} + \alpha^{14})X + \alpha^{27}] \\ &= (X^2 + \alpha^8X + \alpha^3)(X^2 + \alpha^2X + \alpha^{12}) \\ &= X^4 + (\alpha^8 + \alpha^2)X^3 + (\alpha^{12} + \alpha^{10} + \alpha^3)X^2 + (\alpha^{20} + \alpha^5)X + \alpha^{15} \\ &= X^4 + X^3 + 1. \end{aligned}$$

Let $f(X)$ be a polynomial with coefficients from $GF(2)$. If β , an element in $GF(2^m)$, is a root of $f(X)$, the polynomial $f(X)$ may have other roots from $GF(2^m)$. Then, what are these roots? This question is answered by the following theorem.

THEOREM 2.11 Let $f(X)$ be a polynomial with coefficients from $GF(2)$. Let β be an element in an extension field of $GF(2)$. If β is a root of $f(X)$, then for any $l \geq 0$, β^{2^l} is also a root of $f(X)$.

Proof. From (2.10), we have

$$[f(X)]^{2^l} = f(X^{2^l}).$$

Substituting β into the preceding equation, we obtain

$$[f(\beta)]^{2^l} = f(\beta^{2^l}).$$

Since $f(\beta) = 0$, $f(\beta^{2^l}) = 0$. Therefore, β^{2^l} is also a root of $f(X)$. Q.E.D.

The element β^{2^l} is called a *conjugate* of β . Theorem 2.11 says that if β , an element in $GF(2^m)$, is a root of a polynomial $f(X)$ over $GF(2)$, then all the distinct conjugates of β , also elements in $GF(2^m)$, are roots of $f(X)$. For example, the polynomial $f(X) = 1 + X^3 + X^4 + X^5 + X^6$ has α^4 , an element in $GF(2^4)$ given by Table 2.8, as a root. To verify this, we use Table 2.8 and the fact that $\alpha^{15} = 1$:

$$\begin{aligned} f(\alpha^4) &= 1 + \alpha^{12} + \alpha^{16} + \alpha^{20} + \alpha^{24} = 1 + \alpha^{12} + \alpha + \alpha^5 + \alpha^9 \\ &= 1 + (1 + \alpha + \alpha^2 + \alpha^3) + \alpha + (\alpha + \alpha^2) + (\alpha + \alpha^3) = 0. \end{aligned}$$

The conjugates of α^4 are

$$(\alpha^4)^2 = \alpha^8, \quad (\alpha^4)^{2^2} = \alpha^{16} = \alpha, \quad (\alpha^4)^{2^3} = \alpha^{32} = \alpha^2.$$

[Note that $(\alpha^4)^{2^4} = \alpha^{64} = \alpha^4$.] It follows from Theorem 2.11 that α^8 , α , and α^2 must also be roots of $f(X) = 1 + X^3 + X^4 + X^5 + X^6$. We can check that α^5 and its conjugate, α^{10} , are roots of $f(X) = 1 + X^3 + X^4 + X^5 + X^6$. Therefore, $f(X) = 1 + X^3 + X^4 + X^5 + X^6$ has six distinct roots in $GF(2^4)$.

Let β be a nonzero element in the field $GF(2^m)$. It follows from Theorem 2.8 that

$$\beta^{2^m-1} = 1.$$

Adding 1 to both sides of $\beta^{2^m-1} = 1$, we obtain

$$\beta^{2^m-1} + 1 = 0.$$

This says that β is a root of the polynomial $X^{2^m-1} + 1$. Hence, every nonzero element of $GF(2^m)$ is a root of $X^{2^m-1} + 1$. Because the degree of $X^{2^m-1} + 1$ is $2^m - 1$, the $2^m - 1$ nonzero elements of $GF(2^m)$ form all the roots of $X^{2^m-1} + 1$. Summarizing the preceding result, we obtain Theorem 2.12.

THEOREM 2.12 The $2^m - 1$ nonzero elements of $GF(2^m)$ form all the roots of $X^{2^m-1} + 1$.

Since the zero element 0 of $GF(2^m)$ is the root of X , Theorem 2.12 has the following corollary:

COROLLARY 2.12.1 The elements of $GF(2^m)$ form all the roots of $X^{2^m} + X$.

Because any element β in $GF(2^m)$ is a root of the polynomial $X^{2^m} + X$, β may be a root of a polynomial over $GF(2)$ with a degree less than 2^m . Let $\phi(X)$ be the polynomial of *smallest degree* over $GF(2)$ such that $\phi(\beta) = 0$. [We can easily prove that $\phi(X)$ is unique.] This polynomial $\phi(X)$ is called the *minimal polynomial* of β . For example, the minimal polynomial of the zero element 0 of $GF(2^m)$ is X , and the minimal polynomial of the unit element 1 is $X + 1$. Next, we derive a number of properties of minimal polynomials.

THEOREM 2.13 The minimal polynomial $\phi(X)$ of a field element β is irreducible.

Proof. Suppose that $\phi(X)$ is not irreducible and that $\phi(X) = \phi_1(X)\phi_2(X)$, where both $\phi_1(X)$ and $\phi_2(X)$ have degrees greater than 0 and less than the degree of $\phi(X)$. Since $\phi(\beta) = \phi_1(\beta)\phi_2(\beta) = 0$, either $\phi_1(\beta) = 0$ or $\phi_2(\beta) = 0$. This result contradicts the hypothesis that $\phi(X)$ is a polynomial of smallest degree such that $\phi(\beta) = 0$. Therefore, $\phi(X)$ must be irreducible. **Q.E.D.**

THEOREM 2.14 Let $f(X)$ be a polynomial over $GF(2)$. Let $\phi(X)$ be the minimal polynomial of a field element β . If β is a root of $f(X)$, then $f(X)$ is divisible by $\phi(X)$.

Proof. Dividing $f(X)$ by $\phi(X)$, we obtain

$$f(X) = a(X)\phi(X) + r(X),$$

where the degree of the remainder $r(X)$ is less than the degree of $\phi(X)$. Substituting β into the preceding equation and using the fact that $f(\beta) = \phi(\beta) = 0$, we have $r(\beta) = 0$. If $r(X) \neq 0$, $r(X)$ would be a polynomial of lower degree than $\phi(X)$, which has β as a root. This is a contradiction to the fact that $\phi(X)$ is the minimal polynomial of β . Hence, $r(X)$ must be identical to 0 and $\phi(X)$ divides $f(X)$. **Q.E.D.**

The following result follows from Corollary 2.12.1 and Theorem 2.14.

THEOREM 2.15 The minimal polynomial $\phi(X)$ of an element β in $GF(2^m)$ divides $X^{2^m} + X$.

Theorem 2.15 says that all the roots of $\phi(X)$ are from $GF(2^m)$. Then, what are the roots of $\phi(X)$? This question is answered by the next two theorems.

THEOREM 2.16 Let $f(X)$ be an irreducible polynomial over $GF(2)$. Let β be an element in $GF(2^m)$. Let $\phi(X)$ be the minimal polynomial of β . If $f(\beta) = 0$, then $\phi(X) = f(X)$.

Proof. It follows from Theorem 2.14 that $\phi(X)$ divides $f(X)$. Since $\phi(X) \neq 1$ and $f(X)$ is irreducible, we must have $\phi(X) = f(X)$. **Q.E.D.**

Theorem 2.16 says that if an irreducible polynomial has β as a root, it is the minimal polynomial $\phi(X)$ of β . It follows from Theorem 2.11 that β and its conjugates $\beta^2, \beta^{2^2}, \dots, \beta^{2^e}, \dots$ are roots of $\phi(X)$. Let e be the smallest integer such that $\beta^{2^e} = \beta$. Then, $\beta^2, \beta^{2^2}, \dots, \beta^{2^{e-1}}$ are all the distinct conjugates of β (see Problem 2.15). Since $\beta^{2^m} = \beta$, $e \leq m$ (in fact e divides m).

THEOREM 2.17 Let β be an element in $GF(2^m)$, and let e be the smallest nonnegative integer such that $\beta^{2^e} = \beta$. Then,

$$f(X) = \prod_{i=0}^{e-1} (X + \beta^{2^i})$$

is an irreducible polynomial over $GF(2)$.

Proof. Consider

$$[f(X)]^2 = \left[\prod_{i=0}^{e-1} (X + \beta^{2^i}) \right]^2 = \prod_{i=0}^{e-1} (X + \beta^{2^i})^2.$$

Since $(X + \beta^{2^i})^2 = X^2 + (\beta^{2^i} + \beta^{2^i})X + \beta^{2^{i+1}} = X^2 + \beta^{2^{i+1}}$,

$$\begin{aligned} [f(X)]^2 &= \prod_{i=0}^{e-1} (X^2 + \beta^{2^{i+1}}) = \prod_{i=1}^e (X^2 + \beta^{2^i}) \\ &= \left[\prod_{i=1}^{e-1} (X^2 + \beta^{2^i}) \right] (X^2 + \beta^{2^e}). \end{aligned}$$

Since $\beta^{2^e} = \beta$, then

$$[f(X)]^2 = \prod_{i=0}^{e-1} (X^2 + \beta^{2^i}) = f(X^2). \quad (2.21)$$

Let $f(X) = f_0 + f_1X + \cdots + f_eX^e$, where $f_e = 1$. Expand

$$\begin{aligned} [f(X)]^2 &= (f_0 + f_1X + \cdots + f_eX^e)^2 \\ &= \sum_{i=0}^e f_i^2 X^{2i} + (1+1) \sum_{\substack{i=0 \\ i \neq j}}^e \sum_{j=0}^e f_i f_j X^{i+j} = \sum_{i=0}^e f_i^2 X^{2i}. \end{aligned} \quad (2.22)$$

From (2.21) and (2.22), we obtain

$$\sum_{i=0}^e f_i X^{2i} = \sum_{i=0}^e f_i^2 X^{2i}.$$

Then, for $0 \leq i \leq e$, we must have

$$f_i = f_i^2.$$

This result holds only when $f_i = 0$ or 1 . Therefore, $f(X)$ has coefficients from $GF(2)$.

Now, suppose that $f(X)$ is not irreducible over $GF(2)$, and $f(X) = a(X)b(X)$. Since $f(\beta) = 0$, either $a(\beta) = 0$ or $b(\beta) = 0$. If $a(\beta) = 0$, $a(X)$ has $\beta, \beta^2, \dots, \beta^{2^{e-1}}$ as roots, so $a(X)$ has degree e , and $a(X) = f(X)$. Similarly, if $b(\beta) = 0$, $b(X) = f(X)$. Therefore, $f(X)$ must be irreducible. **Q.E.D.**

A direct consequence of Theorems 2.16 and 2.17 is Theorem 2.18.

THEOREM 2.18 Let $\phi(X)$ be the minimal polynomial of an element β in $GF(2^m)$. Let e be the smallest integer such that $\beta^{2^e} = \beta$. Then

$$\phi(X) = \prod_{i=0}^{e-1} (X + \beta^{2^i}). \quad (2.23)$$

EXAMPLE 2.8

Consider the Galois field $GF(2^4)$ given by Table 2.8. Let $\beta = \alpha^3$. The conjugates of β are

$$\beta^2 = \alpha^6, \quad \beta^{2^2} = \alpha^{12}, \quad \beta^{2^3} = \alpha^{24} = \alpha^9.$$

The minimal polynomial of $\beta = \alpha^3$ is then

$$\phi(X) = (X + \alpha^3)(X + \alpha^6)(X + \alpha^{12})(X + \alpha^9).$$

Multiplying out the right-hand side of the preceding equation with the aid of Table 2.8, we obtain

$$\begin{aligned} \phi(X) &= [X^2 + (\alpha^3 + \alpha^6)X + \alpha^9][X^2 + (\alpha^{12} + \alpha^9)X + \alpha^{21}] \\ &= (X^2 + \alpha^2 X + \alpha^9)(X^2 + \alpha^8 X + \alpha^6) \\ &= X^4 + (\alpha^2 + \alpha^8)X^3 + (\alpha^6 + \alpha^{10} + \alpha^9)X^2 + (\alpha^{17} + \alpha^8)X + \alpha^{15} \\ &= X^4 + X^3 + X^2 + X + 1. \end{aligned}$$

There is another way of finding the minimal polynomial of a field element, which is illustrated by the following example.

EXAMPLE 2.9

Suppose that we want to determine the minimal polynomial $\phi(X)$ of $\gamma = \alpha^7$ in $GF(2^4)$. The distinct conjugates of γ are

$$\gamma^2 = \alpha^{14}, \quad \gamma^3 = \alpha^{28} = \alpha^{13}, \quad \gamma^{2^3} = \alpha^{56} = \alpha^{11}.$$

Hence, $\phi(X)$ has degree 4 and must be of the following form:

$$\phi(X) = a_0 + a_1X + a_2X^2 + a_3X^3 + X^4.$$

Substituting γ into $\phi(X)$, we have

$$\phi(\gamma) = a_0 + a_1\gamma + a_2\gamma^2 + a_3\gamma^3 + \gamma^4 = 0.$$

Using the polynomial representations for γ , γ^2 , γ^3 , and γ^4 in the preceding equation, we obtain the following:

$$\begin{aligned} a_0 + a_1(1 + \alpha + \alpha^3) + a_2(1 + \alpha^3) + a_3(\alpha^2 + \alpha^3) + (1 + \alpha^2 + \alpha^3) &= 0 \\ (a_0 + a_1 + a_2 + 1) + a_1\alpha + (a_3 + 1)\alpha^2 + (a_1 + a_2 + a_3 + 1)\alpha^3 &= 0. \end{aligned}$$

For the preceding equality to be true, the coefficients must equal zero:

$$\begin{aligned} a_0 + a_1 + a_2 + 1 &= 0, \\ a_1 &= 0, \\ a_3 + 1 &= 0, \\ a_1 + a_2 + a_3 + 1 &= 0. \end{aligned}$$

Solving the preceding linear equations, we obtain $a_0 = 1$, $a_1 = a_2 = 0$, and $a_3 = 1$. Therefore, the minimal polynomial of $\gamma = \alpha^7$ is $\phi(X) = 1 + X^3 + X^4$. All the minimal polynomials of the elements in $GF(2^4)$ are given by Table 2.9.

TABLE 2.9: Minimal polynomials of the elements in $GF(2^4)$ generated by $p(X) = X^4 + X + 1$.

Conjugate roots	Minimal polynomials
0	X
1	$X + 1$
$\alpha, \alpha^2, \alpha^4, \alpha^8$	$X^4 + X + 1$
$\alpha^3, \alpha^6, \alpha^9, \alpha^{12}$	$X^4 + X^3 + X^2 + X + 1$
α^5, α^{10}	$X^2 + X + 1$
$\alpha^7, \alpha^{11}, \alpha^{13}, \alpha^{14}$	$X^4 + X^3 + 1$

A direct consequence of Theorem 2.18 is Theorem 2.19.

THEOREM 2.19 Let $\phi(X)$ be the minimal polynomial of an element β in $GF(2^m)$. Let e be the degree of $\phi(X)$. Then e is the smallest integer such that $\beta^{2^e} = \beta$. Moreover, $e \leq m$.

In particular, the degree of the minimal polynomial of any element in $GF(2^m)$ divides m . The proof of this property is omitted here. Table 2.9 shows that the degree of the minimal polynomial of each element in $GF(2^4)$ divides by 4. Minimal polynomials of the elements in $GF(2^m)$ for $m = 2$ to 10 are given in Appendix B.

In the construction of the Galois field $GF(2^m)$ we use a primitive polynomial $p(X)$ of degree m and require that the element α be a root of $p(X)$. Because the powers of α generate all the nonzero elements of $GF(2^m)$, α is a primitive element. In fact, all the conjugates of α are primitive elements of $GF(2^m)$. To see this, let n be the order of α^{2^l} for $l > 0$. Then

$$(\alpha^{2^l})^n = \alpha^{n2^l} = 1.$$

Also, it follows from Theorem 2.9 that n divides $2^m - 1$:

$$2^m - 1 = k \cdot n. \quad (2.24)$$

Because α is a primitive element of $GF(2^m)$, its order is $2^m - 1$. For $\alpha^{n2^l} = 1$, $n2^l$ must be a multiple of $2^m - 1$. Since 2^l and $2^m - 1$ are relatively prime, n must be divisible by $2^m - 1$, say

$$n = q \cdot (2^m - 1). \quad (2.25)$$

From (2.24) and (2.25) we conclude that $n = 2^m - 1$. Consequently, α^{2^l} is also a primitive element of $GF(2^m)$. In general, we have the following theorem.

THEOREM 2.20 If β is a primitive element of $GF(2^m)$, all its conjugates $\beta^2, \beta^{2^2}, \dots$ are also primitive elements of $GF(2^m)$.

EXAMPLE 2.10

Consider the field $GF(2^4)$ given by Table 2.8. The powers of $\beta = \alpha^7$ are

$$\begin{aligned} \beta^0 &= 1, \quad \beta^1 = \alpha^7, \quad \beta^2 = \alpha^{14}, \quad \beta^3 = \alpha^{21} = \alpha^6, \quad \beta^4 = \alpha^{28} = \alpha^{13}, \\ \beta^5 &= \alpha^{35} = \alpha^5, \quad \beta^6 = \alpha^{42} = \alpha^{12}, \quad \beta^7 = \alpha^{49} = \alpha^4, \quad \beta^8 = \alpha^{56} = \alpha^{11}, \\ \beta^9 &= \alpha^{63} = \alpha^3, \quad \beta^{10} = \alpha^{70} = \alpha^{10}, \quad \beta^{11} = \alpha^{77} = \alpha^2, \quad \beta^{12} = \alpha^{84} = \alpha^9, \\ \beta^{13} &= \alpha^{91} = \alpha, \quad \beta^{14} = \alpha^{98} = \alpha^8, \quad \beta^{15} = \alpha^{105} = 1. \end{aligned}$$

Clearly, the powers of $\beta = \alpha^7$ generate all the nonzero elements of $GF(2^4)$, so $\beta = \alpha^7$ is a primitive element of $GF(2^7)$. The conjugates of $\beta = \alpha^7$ are

$$\beta^2 = \alpha^{14}, \quad \beta^{2^2} = \alpha^{13}, \quad \beta^{2^3} = \alpha^{11}.$$

We may readily check that they are all primitive elements of $GF(2^m)$.

A more general form of Theorem 2.20 is Theorem 2.21.

THEOREM 2.21 If β is an element of order n in $GF(2^m)$, all its conjugates have the same order n . (The proof is left as an exercise.)

EXAMPLE 2.11

Consider the element α^5 in $GF(2^4)$ given by Table 2.8. Since $(\alpha^5)^2 = \alpha^{20} = \alpha^5$, the only conjugate of α^5 is α^{10} . Both α^5 and α^{10} have order $n = 3$. The minimal polynomial of α^5 and α^{10} is $X^2 + X + 1$, whose degree is a factor of $m = 4$. The conjugates of α^3 are α^6 , α^9 , and α^{12} . They all have order $n = 5$.

2.6 COMPUTATIONS USING GALOIS FIELD $GF(2^m)$ ARITHMETIC

Here we perform some example computations using arithmetic over $GF(2^m)$. Consider the following linear equations over $GF(2^4)$ (see Table 2.8):

$$\begin{aligned} X + \alpha^7 Y &= \alpha^2, \\ \alpha^{12} X + \alpha^8 Y &= \alpha^4. \end{aligned} \tag{2.26}$$

Multiplying the second equation by α^3 gives

$$\begin{aligned} X + \alpha^7 Y &= \alpha^2, \\ X + \alpha^{11} Y &= \alpha^7. \end{aligned}$$

By adding the two preceding equations, we get

$$\begin{aligned} (\alpha^7 + \alpha^{11})Y &= \alpha^2 + \alpha^7, \\ \alpha^8 Y &= \alpha^{12}, \\ Y &= \alpha^4. \end{aligned}$$

Substituting $Y = \alpha^4$ into the first equation of (2.26), we obtain $X = \alpha^9$. Thus, the solution for the equations of (2.26) is $X = \alpha^9$ and $Y = \alpha^4$.

Alternatively, the equations of (2.26) could be solved by using Cramer's rule:

$$\begin{aligned} X &= \frac{\left| \begin{array}{cc} \alpha^2 & \alpha^7 \\ \alpha^4 & \alpha^8 \end{array} \right|}{\left| \begin{array}{cc} 1 & \alpha^7 \\ \alpha^{12} & \alpha^8 \end{array} \right|} = \frac{\alpha^{10} + \alpha^{11}}{\alpha^8 + \alpha^{19}} = \frac{1 + \alpha^3}{\alpha + \alpha^2} = \frac{\alpha^{14}}{\alpha^5} = \alpha^9, \\ Y &= \frac{\left| \begin{array}{cc} 1 & \alpha^2 \\ \alpha^{12} & \alpha^4 \end{array} \right|}{\left| \begin{array}{cc} 1 & \alpha^7 \\ \alpha^{12} & \alpha^8 \end{array} \right|} = \frac{\alpha^4 + \alpha^{14}}{\alpha^8 + \alpha^{19}} = \frac{\alpha + \alpha^3}{\alpha + \alpha^2} = \frac{\alpha^9}{\alpha^5} = \alpha^4. \end{aligned}$$

As one more example, suppose that we want to solve the equation

$$f(X) = X^2 + \alpha^7 X + \alpha = 0$$

over $GF(2^4)$. The quadratic formula will not work because it requires dividing by 2, and in this field, 2 = 0. If $f(X) = 0$ has any solutions in $GF(2^4)$, the solutions can be found simply by substituting all the elements of Table 2.8 for X . By doing so, we would find that $f(\alpha^6) = 0$ and $f(\alpha^{10}) = 0$, since

$$f(\alpha^6) = (\alpha^6)^2 + \alpha^7 \cdot \alpha^6 + \alpha = \alpha^{12} + \alpha^{13} + \alpha = 0,$$

$$f(\alpha^{10}) = (\alpha^{10})^2 + \alpha^7 \cdot \alpha^{10} + \alpha = \alpha^5 + \alpha^2 + \alpha = 0.$$

Thus, α^6 and α^{10} are the roots of $f(X)$, and $f(X) = (X + \alpha^6)(X + \alpha^{10})$.

The preceding computations are typical of those required for decoding codes such as BCH and Reed–Solomon codes, and they can be programmed quite easily on a general-purpose computer. It is also a simple matter to build a computer that can do this kind of arithmetic.

2.7 VECTOR SPACES

Let V be a set of elements on which a binary operation called addition, $+$, is defined. Let F be a field. A multiplication operation, denoted by \cdot , between the elements in F and elements in V is also defined. The set V is called a *vector space* over the field F if it satisfies the following conditions:

- i. V is a commutative group under addition.
- ii. For any element a in F and any element \mathbf{v} in V , $a \cdot \mathbf{v}$ is an element in V .
- iii. (Distributive Laws) For any elements \mathbf{u} and \mathbf{v} in V and any elements a and b in F ,

$$a \cdot (\mathbf{u} + \mathbf{v}) = a \cdot \mathbf{u} + a \cdot \mathbf{v},$$

$$(a + b) \cdot \mathbf{v} = a \cdot \mathbf{v} + b \cdot \mathbf{v}.$$

- iv. (Associative Law) For any \mathbf{v} in V and any a and b in F ,

$$(a \cdot b) \cdot \mathbf{v} = a \cdot (b \cdot \mathbf{v}).$$

- v. Let 1 be the unit element of F . Then, for any \mathbf{v} in V , $1 \cdot \mathbf{v} = \mathbf{v}$.

The elements of V are called *vectors*, and the elements of the field F are called *scalars*. The addition on V is called a *vector addition*, and the multiplication that combines a scalar in F and a vector in V into a vector in V is referred to as *scalar multiplication* (or *product*). The additive identity of V is denoted by $\mathbf{0}$.

Some basic properties of a vector space V over a field F can be derived from the preceding definition.

Property I Let 0 be the zero element of the field F . For any vector \mathbf{v} in V , $0 \cdot \mathbf{v} = \mathbf{0}$.

Proof. Because $1 + 0 = 1$ in F , we have $1 \cdot \mathbf{v} = (1 + 0) \cdot \mathbf{v} = 1 \cdot \mathbf{v} + 0 \cdot \mathbf{v}$. Using condition (v) of the preceding definition of a vector space, we obtain

$\mathbf{v} = \mathbf{v} + 0 \cdot \mathbf{v}$. Let $-\mathbf{v}$ be the additive inverse of \mathbf{v} . Adding $-\mathbf{v}$ to both sides of $\mathbf{v} = \mathbf{v} + 0 \cdot \mathbf{v}$, we have

$$\mathbf{0} = \mathbf{0} + 0 \cdot \mathbf{v}$$

$$\mathbf{0} = 0 \cdot \mathbf{v}.$$

Property II For any scalar c in F , $c \cdot \mathbf{0} = \mathbf{0}$. (The proof is left as an exercise.)

Property III For any scalar c in F and any vector \mathbf{v} in V ,

$$(-c) \cdot \mathbf{v} = c \cdot (-\mathbf{v}) = -(c \cdot \mathbf{v})$$

That is, $(-c) \cdot \mathbf{v}$ or $c \cdot (-\mathbf{v})$ is the additive inverse of the vector $c \cdot \mathbf{v}$. (The proof is left as an exercise.)

Next, we present a very useful vector space over $GF(2)$ that plays a central role in coding theory. Consider an ordered sequence of n components,

$$(a_0, a_1, \dots, a_{n-1}),$$

where each component a_i is an element from the binary field $GF(2)$ (i.e., $a_i = 0$ or 1). This sequence is generally called an n -tuple over $GF(2)$. Because there are two choices for each a_i , we can construct 2^n distinct n -tuples. Let V_n denote this set of 2^n distinct n -tuples over $GF(2)$. Now, we define an addition, $+$, on V_n as the following: For any $\mathbf{u} = (u_0, u_1, \dots, u_{n-1})$ and $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ in V_n ,

$$\mathbf{u} + \mathbf{v} = (u_0 + v_0, u_1 + v_1, \dots, u_{n-1} + v_{n-1}), \quad (2.27)$$

where $u_i + v_i$ is carried out in modulo-2 addition. Clearly, $\mathbf{u} + \mathbf{v}$ is also an n -tuple over $GF(2)$. Hence, V_n is closed under the addition defined by (2.27). We can readily verify that V_n is a commutative group under the addition defined by (2.27). First, we note that the all-zero n -tuple $\mathbf{0} = (0, 0, \dots, 0)$ is the additive identity. For any \mathbf{v} in V_n ,

$$\begin{aligned} \mathbf{v} + \mathbf{v} &= (v_0 + v_0, v_1 + v_1, \dots, v_{n-1} + v_{n-1}) \\ &= (0, 0, \dots, 0) = \mathbf{0}. \end{aligned}$$

Hence, the additive inverse of each n -tuple in V_n is itself. Because modulo-2 addition is commutative and associative, we can easily check that the addition defined by (2.27) is also commutative and associative. Therefore, V_n is a commutative group under the addition defined by (2.27).

Next, we define scalar multiplication of an n -tuple \mathbf{v} in V_n by an element a from $GF(2)$ as follows:

$$a \cdot (v_0, v_1, \dots, v_{n-1}) = (a \cdot v_0, a \cdot v_1, \dots, a \cdot v_{n-1}), \quad (2.28)$$

where $a \cdot v_i$ is carried out in modulo-2 multiplication. Clearly, $a \cdot (v_0, v_1, \dots, v_{n-1})$ is also an n -tuple in V_n . If $a = 1$,

$$\begin{aligned} 1 \cdot (v_0, v_1, \dots, v_{n-1}) &= (1 \cdot v_0, 1 \cdot v_1, \dots, 1 \cdot v_{n-1}) \\ &= (v_0, v_1, \dots, v_{n-1}). \end{aligned}$$

We can easily show that the vector addition and scalar multiplication defined by (2.27) and (2.28), respectively, satisfy the distributive and associative laws. Therefore, the set V_n of all n -tuples over $GF(2)$ forms a vector space over $GF(2)$.

EXAMPLE 2.12

Let $n = 5$. The vector space V_5 of all 5-tuples over $GF(2)$ consists of the following 32 vectors:

$$\begin{aligned} & (00000), (00001), (00010), (00011), \\ & (00100), (00101), (00110), (00111), \\ & (01000), (01001), (01010), (01011), \\ & (01100), (01101), (01110), (01111), \\ & (10000), (10001), (10010), (10011), \\ & (10100), (10101), (10110), (10111), \\ & (11000), (11001), (11010), (11011), \\ & (11100), (11101), (11110), (11111). \end{aligned}$$

The vector sum of (10111) and (11001) is

$$(10111) + (11001) = (1+1, 0+1, 1+0, 1+0, 1+1) = (01110).$$

Using the rule of scalar multiplication defined by (2.28), we obtain

$$0 \cdot (11010) = (0 \cdot 1, 0 \cdot 1, 0 \cdot 0, 0 \cdot 1, 0 \cdot 0) = (00000),$$

$$1 \cdot (11010) = (1 \cdot 1, 1 \cdot 1, 1 \cdot 0, 1 \cdot 1, 1 \cdot 0) = (11010).$$

The vector space of all n -tuples over any field F can be constructed in a similar manner; however, in this text we are mostly concerned with the vector space of all n -tuples over $GF(2)$ or over an extension field of $GF(2)$ [e.g., $GF(2^m)$].

Because V is a vector space over a field F , it may happen that a subset S of V is also a vector space over F . Such a subset is called a *subspace* of V .

THEOREM 2.22 Let S be a nonempty subset of a vector space V over a field F . Then, S is a subspace of V if the following conditions are satisfied:

- i. For any two vectors \mathbf{u} and \mathbf{v} in S , $\mathbf{u} + \mathbf{v}$ is also a vector in S .
- ii. For any element a in F and any vector \mathbf{u} in S , $a \cdot \mathbf{u}$ is also in S .

Proof. Conditions (i) and (ii) simply say that S is closed under vector addition and scalar multiplication of V . Condition (ii) ensures that for any vector \mathbf{v} in S its additive inverse $(-1) \cdot \mathbf{v}$ is also in S . Then, $\mathbf{v} + (-1) \cdot \mathbf{v} = \mathbf{0}$ is also in S . Therefore, S is a subgroup of V . Because the vectors of S are also vectors of V , the associative and distributive laws must hold for S . Hence, S is a vector space over F and is a subspace of V . Q.E.D.

EXAMPLE 2.13

Consider the vector space V_5 of all 5-tuples over $GF(2)$ given in Example 2.12. The set

$$\{(00000), (00111), (11010), (11101)\}$$

satisfies both conditions of Theorem 2.22, so it is a subspace of V_5 .

Let $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ be k vectors in a vector space V over a field F . Let a_1, a_2, \dots, a_k be k scalars from F . The sum

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k$$

is called a *linear combination* of $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$. Clearly, the sum of two linear combinations of $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$,

$$\begin{aligned} & (a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k) + (b_1\mathbf{v}_1 + b_2\mathbf{v}_2 + \cdots + b_k\mathbf{v}_k) \\ &= (a_1 + b_1)\mathbf{v}_1 + (a_2 + b_2)\mathbf{v}_2 + \cdots + (a_k + b_k)\mathbf{v}_k, \end{aligned}$$

is also a linear combination of $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$, and the product of a scalar c in F and a linear combination of $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$,

$$c \cdot (a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k) = (c \cdot a_1)\mathbf{v}_1 + (c \cdot a_2)\mathbf{v}_2 + \cdots + (c \cdot a_k)\mathbf{v}_k,$$

is also a linear combination of $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$. It follows from Theorem 2.22 that we have the following result.

THEOREM 2.23 Let $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ be k vectors in a vector space V over a field F . The set of all linear combinations of $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ forms a subspace of V .

EXAMPLE 2.14

Consider the vector space V_5 of all 5-tuples over $GF(2)$ given by Example 2.12. The linear combinations of (00111) and (11101) are

$$\begin{aligned} 0 \cdot (00111) + 0 \cdot (11101) &= (00000), \\ 0 \cdot (00111) + 1 \cdot (11101) &= (11101), \\ 1 \cdot (00111) + 0 \cdot (11101) &= (00111), \\ 1 \cdot (00111) + 1 \cdot (11101) &= (11010). \end{aligned}$$

These four vectors form the same subspace given by Example 2.13.

A set of vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ in a vector space V over a field F is said to be *linearly dependent* if and only if there exist k scalars a_1, a_2, \dots, a_k from F , *not all zero*, such that

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k = \mathbf{0}.$$

A set of vectors $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ is said to be *linearly independent* if it is not linearly dependent. That is, if $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ are linearly independent, then

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_k\mathbf{v}_k \neq \mathbf{0}$$

unless $a_1 = a_2 = \cdots = a_k = 0$.

EXAMPLE 2.15

The vectors (10110) , (01001) , and (11111) are linearly dependent, since

$$1 \cdot (10110) + 1 \cdot (01001) + 1 \cdot (11111) = (00000);$$

however, (10110) , (01001) , and (11011) are linearly independent. All eight linear combinations of these three vectors are given here:

$$0 \cdot (10110) + 0 \cdot (01001) + 0 \cdot (11011) = (00000),$$

$$0 \cdot (10110) + 0 \cdot (01001) + 1 \cdot (11011) = (11011),$$

$$0 \cdot (10110) + 1 \cdot (01001) + 0 \cdot (11011) = (01001),$$

$$0 \cdot (10110) + 1 \cdot (01001) + 1 \cdot (11011) = (10010),$$

$$1 \cdot (10110) + 0 \cdot (01001) + 0 \cdot (11011) = (10110),$$

$$1 \cdot (10110) + 0 \cdot (01001) + 1 \cdot (11011) = (01101),$$

$$1 \cdot (10110) + 1 \cdot (01001) + 0 \cdot (11011) = (11111),$$

$$1 \cdot (10110) + 1 \cdot (01001) + 1 \cdot (11011) = (00100).$$

A set of vectors is said to *span* a vector space V if every vector in V is a linear combination of the vectors in the set. In any vector space or subspace there exists at least one set B of linearly independent vectors that span the space. This set is called a *basis* (or *base*) of the vector space. The number of vectors in a basis of a vector space is called the *dimension* of the vector space. (Note that the number of vectors in any two bases are the same.)

Consider the vector space V_n of all n -tuples over $GF(2)$. Let us form the following n n -tuples:

$$\mathbf{e}_0 = (1, 0, 0, 0, \dots, 0, 0),$$

$$\mathbf{e}_1 = (0, 1, 0, 0, \dots, 0, 0),$$

$$\vdots$$

$$\mathbf{e}_{n-1} = (0, 0, 0, 0, \dots, 0, 1),$$

where the n -tuple \mathbf{e}_i has only one nonzero component at the i th position. Then, every n -tuple $(a_0, a_1, a_2, \dots, a_{n-1})$ in V_n can be expressed as a linear combination of $\mathbf{e}_0, \mathbf{e}_1, \dots, \mathbf{e}_{n-1}$ as follows:

$$(a_0, a_1, a_2, \dots, a_{n-1}) = a_0\mathbf{e}_0 + a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + \cdots + a_{n-1}\mathbf{e}_{n-1}.$$

Therefore, $\mathbf{e}_0, \mathbf{e}_1, \dots, \mathbf{e}_{n-1}$ span the vector space V_n of all n -tuples over $GF(2)$. From the preceding equation we also see that $\mathbf{e}_0, \mathbf{e}_1, \dots, \mathbf{e}_{n-1}$ are linearly independent. Hence, they form a basis for V_n , and the dimension of V_n is n . If $k < n$ and $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ are k linearly independent vectors in V_n , then all the linear combinations of $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$ of the form

$$\mathbf{u} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_k\mathbf{v}_k$$

form a k -dimensional subspace S of V_n . Because each c_i has two possible values, 0 or 1, there are 2^k possible distinct linear combinations of $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_k$. Thus, S consists of 2^k vectors and is a k -dimensional subspace of V_n .

Let $\mathbf{u} = (u_0, u_1, \dots, u_{n-1})$ and $\mathbf{v} = (v_1, v_2, \dots, v_{n-1})$ be two n -tuples in V_n . We define the *inner product* (or *dot product*) of \mathbf{u} and \mathbf{v} as

$$\mathbf{u} \cdot \mathbf{v} = u_0 \cdot v_0 + u_1 \cdot v_1 + \cdots + u_{n-1} \cdot v_{n-1}, \quad (2.29)$$

where $u_i \cdot v_i$ and $u_i \cdot v_i + u_{i+1} \cdot v_{i+1}$ are carried out in modulo-2 multiplication and addition. Hence, the inner product $\mathbf{u} \cdot \mathbf{v}$ is a scalar in $GF(2)$. If $\mathbf{u} \cdot \mathbf{v} = 0$, \mathbf{u} and \mathbf{v} are said to be *orthogonal* to each other. The inner product has the following properties:

- i. $\mathbf{u} \cdot \mathbf{v} = \mathbf{v} \cdot \mathbf{u}$.
- ii. $\mathbf{u} \cdot (\mathbf{v} + \mathbf{w}) = \mathbf{u} \cdot \mathbf{v} + \mathbf{u} \cdot \mathbf{w}$.
- iii. $(a\mathbf{u}) \cdot \mathbf{v} = a(\mathbf{u} \cdot \mathbf{v})$.

(The concept of inner product can be generalized to any Galois field.)

Let S be a k -dimensional subspace of V_n and let S_d be the set of vectors in V_n such that for any \mathbf{u} in S and \mathbf{v} in S_d , $\mathbf{u} \cdot \mathbf{v} = 0$. The set S_d contains at least the all-zero n -tuple $\mathbf{0} = (0, 0, \dots, 0)$, since for any \mathbf{u} in S , $\mathbf{0} \cdot \mathbf{u} = 0$. Thus, S_d is nonempty. For any element a in $GF(2)$ and any \mathbf{v} in S_d ,

$$a \cdot \mathbf{v} = \begin{cases} \mathbf{0} & \text{if } a = 0, \\ \mathbf{v} & \text{if } a = 1. \end{cases}$$

Therefore, $a \cdot \mathbf{v}$ is also in S_d . Let \mathbf{v} and \mathbf{w} be any two vectors in S_d . For any vector \mathbf{u} in S , $\mathbf{u} \cdot (\mathbf{v} + \mathbf{w}) = \mathbf{u} \cdot \mathbf{v} + \mathbf{u} \cdot \mathbf{w} = 0 + 0 = 0$. This says that if \mathbf{v} and \mathbf{w} are orthogonal to \mathbf{u} , the vector sum $\mathbf{v} + \mathbf{w}$ is also orthogonal to \mathbf{u} . Consequently, $\mathbf{v} + \mathbf{w}$ is a vector in S_d . It follows from Theorem 2.22 that S_d is also a subspace of V_n . This subspace S_d is called the *null (or dual) space* of S . Conversely, S is also the null space of S_d . The dimension of S_d is given by Theorem 2.24, whose proof is omitted here [1].

THEOREM 2.24 Let S be a k -dimensional subspace of the vector space V_n of all n -tuples over $GF(2)$. The dimension of its null space S_d is $n - k$. In other words, $\dim(S) + \dim(S_d) = n$.

EXAMPLE 2.16

Consider the vector space V_5 of all 5-tuples over $GF(2)$ given by Example 2.12. The following eight vectors form a three-dimensional subspace S of V_5 :

$$(00000), \quad (11100), \quad (01010), \quad (10001), \\ (10110), \quad (01101), \quad (11011), \quad (00111).$$

The null space S_d of S consists of the following four vectors:

$$(00000), \quad (10101), \quad (01110), \quad (11011).$$

S_d is spanned by (10101) and (01110) , which are linearly independent. Thus, the dimension of S_d is 2.

All the results presented in this section can be generalized in a straightforward manner to the vector space of all n -tuples over $GF(q)$, where q is a power of prime (see Section 7.1).

2.8 MATRICES

A $k \times n$ matrix over $GF(2)$ (or over any other field) is a rectangular array with k rows and n columns,

$$\mathbf{G} = \begin{bmatrix} g_{00} & g_{01} & g_{02} & \cdots & g_{0,n-1} \\ g_{10} & g_{11} & g_{12} & \cdots & g_{1,n-1} \\ \vdots & & & & \\ g_{k-1,0} & g_{k-1,1} & g_{k-1,2} & \cdots & g_{k-1,n-1} \end{bmatrix}, \quad (2.30)$$

where each entry g_{ij} with $0 \leq i < k$ and $0 \leq j < n$ is an element from the binary field $GF(2)$. Observe that the first index, i , indicates the row containing g_{ij} , and the second index, j , tells which column g_{ij} is in. We shall sometimes abbreviate the matrix of (2.30) by the notation $[g_{ij}]$. We also observe that each row of \mathbf{G} is an n -tuple over $GF(2)$, and each column is a k -tuple over $GF(2)$. The matrix \mathbf{G} can also be represented by its k rows $\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{k-1}$ as follows:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix}.$$

If the k ($k \leq n$) rows of \mathbf{G} are linearly independent, then the 2^k linear combinations of these rows form a k -dimensional subspace of the vector space V_n of all the n -tuples over $GF(2)$. This subspace is called the *row space* of \mathbf{G} . We may interchange any two rows of \mathbf{G} or add one row to another. These are called *elementary row operations*. Performing elementary row operations on \mathbf{G} , we obtain another matrix \mathbf{G}' over $GF(2)$; however, both \mathbf{G} and \mathbf{G}' give the same row space.

EXAMPLE 2.17

Consider a 3×6 matrix \mathbf{G} over $GF(2)$,

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Adding the third row to the first row and interchanging the second and third rows, we obtain the following matrix:

$$\mathbf{G}' = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

Both \mathbf{G} and \mathbf{G}' give the following row space:

$$(000000), \quad (100101), \quad (010011), \quad (001110), \\ (110110), \quad (101011), \quad (011101), \quad (111000).$$

This is a three-dimensional subspace of the vector space V_6 of all the 6-tuples over $GF(2)$.

Let S be the row space of a $k \times n$ matrix \mathbf{G} over $GF(2)$ whose k rows $\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{k-1}$ are linearly independent. Let S_d be the null space of S . Then, the dimension of S_d is $n - k$. Let $\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_{n-k-1}$ be $n - k$ linearly independent vectors in S_d . Clearly, these vectors span S_d . We may form an $(n - k) \times n$ matrix \mathbf{H} using $\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_{n-k-1}$ as rows:

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_{n-k-1} \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & \cdots & h_{0,n-1} \\ h_{10} & h_{11} & \cdots & h_{1,n-1} \\ \vdots & \vdots & & \vdots \\ h_{n-k-1,0} & h_{n-k-1,1} & \cdots & h_{n-k-1,n-1} \end{bmatrix}.$$

The row space of \mathbf{H} is S_d . Because each row \mathbf{g}_i of \mathbf{G} is a vector in S , and each row \mathbf{h}_j of \mathbf{H} is a vector of S_d , the inner product of \mathbf{g}_i and \mathbf{h}_j must be zero (i.e., $\mathbf{g}_i \cdot \mathbf{h}_j = 0$). Because the row space S of \mathbf{G} is the null space of the row space S_d of \mathbf{H} , we call S the null (or dual) space of \mathbf{H} . Summarizing the preceding results, we have Theorem 2.25.

THEOREM 2.25 For any $k \times n$ matrix \mathbf{G} over $GF(2)$ with k linearly independent rows, there exists an $(n - k) \times n$ matrix \mathbf{H} over $GF(2)$ with $n - k$ linearly independent rows such that for any row \mathbf{g}_i in \mathbf{G} and any \mathbf{h}_j in \mathbf{H} , $\mathbf{g}_i \cdot \mathbf{h}_j = 0$. The row space of \mathbf{G} is the null space of \mathbf{H} , and vice versa.

EXAMPLE 2.18

Consider the following 3×6 matrix over $GF(2)$:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

The row space of this matrix is the null space

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

We can easily check that each row of \mathbf{G} is orthogonal to each row of \mathbf{H} .

Two matrices can be added if they have the same number of rows and the same number of columns. To add two $k \times n$ matrices $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$, we simply add their corresponding entries a_{ij} and b_{ij} as follows:

$$[a_{ij}] + [b_{ij}] = [a_{ij} + b_{ij}].$$

Hence, the resultant matrix is also a $k \times n$ matrix. Two matrices can be multiplied provided that the number of columns in the first matrix is equal to the number of rows in the second matrix. Multiplying a $k \times n$ matrix $\mathbf{A} = [a_{ij}]$ by an $n \times l$ matrix $\mathbf{B} = [b_{ij}]$, we obtain the product

$$\mathbf{C} = \mathbf{A} \times \mathbf{B} = [c_{ij}].$$

In the resultant $k \times l$ matrix the entry c_{ij} is equal to the inner product of the i th row \mathbf{a}_i in \mathbf{A} and the j th column \mathbf{b}_j in \mathbf{B} ; that is,

$$c_{ij} = \mathbf{a}_i \cdot \mathbf{b}_j = \sum_{t=0}^{n-1} a_{it} b_{tj}.$$

Let \mathbf{G} be a $k \times n$ matrix over $GF(2)$. The *transpose* of \mathbf{G} , denoted by \mathbf{G}^T , is an $n \times k$ matrix whose rows are columns of \mathbf{G} and whose columns are rows of \mathbf{G} . A $k \times k$ matrix is called an *identity* matrix if it has 1's on the main diagonal and 0's elsewhere. This matrix is usually denoted by \mathbf{I}_k . A *submatrix* of a matrix \mathbf{G} is a matrix that is obtained by striking out given rows or columns of \mathbf{G} .

It is straightforward to generalize the concepts and results presented in this section to matrices with entries from $GF(q)$ with q as a power of a prime.

PROBLEMS

- 2.1 Construct the group under modulo-6 addition.
- 2.2 Construct the group under modulo-3 multiplication.
- 2.3 Let m be a positive integer. If m is not a prime, prove that the set $\{1, 2, \dots, m-1\}$ is not a group under modulo- m multiplication.
- 2.4 Construct the prime field $GF(11)$ with modulo-11 addition and multiplication. Find all the primitive elements, and determine the orders of other elements.
- 2.5 Let m be a positive integer. If m is not prime, prove that the set $\{0, 1, 2, \dots, m-1\}$ is not a field under modulo- m addition and multiplication.
- 2.6 Consider the integer group $G = \{0, 1, 2, \dots, 31\}$ under modulo-32 addition. Show that $H = \{0, 4, 8, 12, 16, 20, 24, 28\}$ forms a subgroup of G . Decompose G into cosets with respect to H (or modulo H).
- 2.7 Let λ be the characteristic of a Galois field $GF(q)$. Let 1 be the unit element of $GF(q)$. Show that the sums

$$1, \quad \sum_{i=1}^2 1, \quad \sum_{i=1}^3 1, \quad \dots, \quad \sum_{i=1}^{\lambda-1} 1, \quad \sum_{i=1}^{\lambda} 1 = 0$$

form a subfield of $GF(q)$.

- 2.8 Prove that every finite field has a primitive element.

- 2.9** Solve the following simultaneous equations of X, Y, Z , and W with modulo-2 arithmetic:

$$\begin{array}{l} X + Y + W = 1, \\ X + Z + W = 0, \\ X + Y + Z + W = 1, \\ Y + Z + W = 0. \end{array}$$

- 2.10** Show that $X^5 + X^3 + 1$ is irreducible over $GF(2)$.

- 2.11** Let $f(X)$ be a polynomial of degree n over $GF(2)$. The reciprocal of $f(X)$ is defined as

$$f^*(X) = X^n f\left(\frac{1}{X}\right).$$

- a. Prove that $f^*(X)$ is irreducible over $GF(2)$ if and only if $f(X)$ is irreducible over $GF(2)$.

- b. Prove that $f^*(X)$ is primitive if and only if $f(X)$ is primitive.

- 2.12** Find all the irreducible polynomials of degree 5 over $GF(2)$.

- 2.13** Construct a table for $GF(2^3)$ based on the primitive polynomial $p(X) = 1 + X + X^3$. Display the power, polynomial, and vector representations of each element. Determine the order of each element.

- 2.14** Construct a table for $GF(2^5)$ based on the primitive polynomial $p(X) = 1 + X^2 + X^5$. Let α be a primitive element of $GF(2^5)$. Find the minimal polynomials of α^3 and α^7 .

- 2.15** Let β be an element in $GF(2^m)$. Let e be the smallest nonnegative integer such that $\beta^{2^e} = \beta$. Prove that $\beta^2, \beta^{2^2}, \dots, \beta^{2^{e-1}}$, are all the distinct conjugates of β .

- 2.16** Prove Theorem 2.21.

- 2.17** Let α be a primitive element in $GF(2^4)$. Use Table 2.8 to find the roots of $f(X) = X^3 + \alpha^6 X^2 + \alpha^9 X + \alpha^9$.

- 2.18** Let α be a primitive element in $GF(2^4)$. Divide the polynomial $f(X) = \alpha^3 X^7 + \alpha X^6 + \alpha^7 X^4 + \alpha^2 X^2 + \alpha^{11} X + 1$ over $GF(2^4)$ by the polynomial $g(X) = X^4 + \alpha^3 X^2 + \alpha^5 X + 1$ over $GF(2^4)$. Find the quotient and the remainder (use Table 2.8).

- 2.19** Let α be a primitive element in $GF(2^4)$. Use Table 2.8 to solve the following simultaneous equations for X, Y , and Z :

$$\begin{array}{l} X + \alpha^5 Y + Z = \alpha^7, \\ X + \alpha Y + \alpha^7 Z = \alpha^9, \\ \alpha^2 X + Y + \alpha^6 Z = \alpha. \end{array}$$

- 2.20** Let V be a vector space over a field F . For any element c in F , prove that $c \cdot \mathbf{0} = \mathbf{0}$.

- 2.21** Let V be a vector space over a field F . Prove that, for any c in F and any \mathbf{v} in V , $(-c) \cdot \mathbf{v} = c \cdot (-\mathbf{v}) = -(c \cdot \mathbf{v})$.

- 2.22** Let S be a subset of the vector space V_n of all n -tuples over $GF(2)$. Prove that S is a subspace of V_n if for any \mathbf{u} and \mathbf{v} in S , $\mathbf{u} + \mathbf{v}$ is in S .

- 2.23** Prove that the set of polynomials over $GF(2)$ with degree $n - 1$ or less forms a vector space $GF(2)$ with dimension n .

- 2.24** Prove that $GF(2^m)$ is a vector space over $GF(2)$.

- 2.25** Construct the vector space V_5 of all 5-tuples over $GF(2)$. Find a three-dimensional subspace and determine its null space.

2.26 Given the matrices

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix},$$

show that the row space of \mathbf{G} is the null space of \mathbf{H} , and vice versa.

- 2.27** Let S_1 and S_2 be two subspaces of a vector V . Show that the intersection of S_1 and S_2 is also a subspace in V .
- 2.28** Construct the vector space of all 3-tuples over $GF(3)$. Form a two-dimensional subspace and its dual space.

BIBLIOGRAPHY

1. G. Birkhoff and S. Maclane, *A Survey of Modern Algebra*, Macmillan, New York, 1953.
2. R. D. Carmichael, *Introduction to the Theory of Groups of Finite Order*, Ginn & Co., Boston, 1937.
3. A. Clark, *Elements of Abstract Algebra*, Dover, New York, 1984.
4. R. A. Dean, *Classical Abstract Algebra*, Harper & Row, New York, 1990.
5. T. W. Hungerford, *Abstract Algebra: An Introduction*, 2d ed., Saunders College Publishing, New York, 1997.
6. R. W. Marsh, *Table of Irreducible Polynomials over GF(2) through Degree 19*, NSA, Washington, D.C., 1957.
7. J. E. Maxfield and M. W. Maxfield, *Abstract Algebra and Solution by Radicals*, Dover, New York, 1992.
8. W. W. Peterson, *Error-Correcting Codes*, MIT Press, Cambridge, 1961.
9. B. L. Van der Waerden, *Modern Algebra*, Vols. 1 and 2, Ungar, New York, 1949.

In fact, a binary block code is linear if and only if the modulo-2 sum of two codewords is also a codeword. The block code given in Table 3.1 is a $(7, 4)$ linear code. One can easily check that the sum of any two codewords in this code is also a codeword.

Because an (n, k) linear code C is a k -dimensional subspace of the vector space V_n of all the binary n -tuples, it is possible to find k linearly independent codewords, $\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{k-1}$, in C such that every codeword \mathbf{v} in C is a linear combination of these k codewords; that is,

$$\mathbf{v} = u_0\mathbf{g}_0 + u_1\mathbf{g}_1 + \dots + u_{k-1}\mathbf{g}_{k-1}, \quad (3.1)$$

where $u_i = 0$ or 1 for $0 \leq i < k$. We arrange these k linearly independent codewords as the rows of a $k \times n$ matrix as follows:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{00} & g_{01} & g_{02} & \cdots & g_{0,n-1} \\ g_{10} & g_{11} & g_{12} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ g_{k-1,0} & g_{k-1,1} & g_{k-1,2} & \cdots & g_{k-1,n-1} \end{bmatrix}, \quad (3.2)$$

where $\mathbf{g}_i = (g_{i0}, g_{i1}, \dots, g_{i,n-1})$ for $0 \leq i < k$. If $\mathbf{u} = (u_0, u_1, \dots, u_{k-1})$ is the message to be encoded, the corresponding codeword can be given as follows:

$$\begin{aligned} \mathbf{v} &= \mathbf{u} \cdot \mathbf{G} \\ &= (u_0, u_1, \dots, u_{k-1}) \cdot \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} \\ &= u_0\mathbf{g}_0 + u_1\mathbf{g}_1 + \dots + u_{k-1}\mathbf{g}_{k-1}. \end{aligned} \quad (3.3)$$

Clearly, the rows of \mathbf{G} generate (or span) the (n, k) linear code C . For this reason, the matrix \mathbf{G} is called a *generator matrix* for C . Note that any k linearly independent codewords of an (n, k) linear code can be used to form a generator matrix for the code. It follows from (3.3) that an (n, k) linear code is completely specified by the k rows of a generator matrix \mathbf{G} . Therefore, the encoder has only to store the k rows of \mathbf{G} and to form a linear combination of these k rows based on the input message $\mathbf{u} = (u_0, u_1, \dots, u_{k-1})$.

EXAMPLE 3.1

The $(7, 4)$ linear code given in Table 3.1 has the following matrix as a generator matrix:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \mathbf{g}_2 \\ \mathbf{g}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

In fact, a binary block code is linear if and only if the modulo-2 sum of two codewords is also a codeword. The block code given in Table 3.1 is a $(7, 4)$ linear code. One can easily check that the sum of any two codewords in this code is also a codeword.

Because an (n, k) linear code C is a k -dimensional subspace of the vector space V_n of all the binary n -tuples, it is possible to find k linearly independent codewords, $\mathbf{g}_0, \mathbf{g}_1, \dots, \mathbf{g}_{k-1}$, in C such that every codeword \mathbf{v} in C is a linear combination of these k codewords; that is,

$$\mathbf{v} = u_0\mathbf{g}_0 + u_1\mathbf{g}_1 + \dots + u_{k-1}\mathbf{g}_{k-1}, \quad (3.1)$$

where $u_i = 0$ or 1 for $0 \leq i < k$. We arrange these k linearly independent codewords as the rows of a $k \times n$ matrix as follows:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{00} & g_{01} & g_{02} & \cdots & g_{0,n-1} \\ g_{10} & g_{11} & g_{12} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ g_{k-1,0} & g_{k-1,1} & g_{k-1,2} & \cdots & g_{k-1,n-1} \end{bmatrix}, \quad (3.2)$$

where $\mathbf{g}_i = (g_{i0}, g_{i1}, \dots, g_{i,n-1})$ for $0 \leq i < k$. If $\mathbf{u} = (u_0, u_1, \dots, u_{k-1})$ is the message to be encoded, the corresponding codeword can be given as follows:

$$\begin{aligned} \mathbf{v} &= \mathbf{u} \cdot \mathbf{G} \\ &= (u_0, u_1, \dots, u_{k-1}) \cdot \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} \\ &= u_0\mathbf{g}_0 + u_1\mathbf{g}_1 + \dots + u_{k-1}\mathbf{g}_{k-1}. \end{aligned} \quad (3.3)$$

Clearly, the rows of \mathbf{G} generate (or span) the (n, k) linear code C . For this reason, the matrix \mathbf{G} is called a *generator* matrix for C . Note that any k linearly independent codewords of an (n, k) linear code can be used to form a generator matrix for the code. It follows from (3.3) that an (n, k) linear code is completely specified by the k rows of a generator matrix \mathbf{G} . Therefore, the encoder has only to store the k rows of \mathbf{G} and to form a linear combination of these k rows based on the input message $\mathbf{u} = (u_0, u_1, \dots, u_{k-1})$.

EXAMPLE 3.1

The $(7, 4)$ linear code given in Table 3.1 has the following matrix as a generator matrix:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \mathbf{g}_2 \\ \mathbf{g}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

TABLE 3.1: Linear block code with $k = 4$ and $n = 7$.

Messages	Codewords
(0 0 0 0)	(0 0 0 0 0 0 0)
(1 0 0 0)	(1 1 0 1 0 0 0)
(0 1 0 0)	(0 1 1 0 1 0 0)
(1 1 0 0)	(1 0 1 1 1 0 0)
(0 0 1 0)	(1 1 1 0 0 1 0)
(1 0 1 0)	(0 0 1 1 0 1 0)
(0 1 1 0)	(1 0 0 0 1 1 0)
(1 1 1 0)	(0 1 0 1 1 1 0)
(0 0 0 1)	(1 0 1 0 0 0 1)
(1 0 0 1)	(0 1 1 1 0 0 1)
(0 1 0 1)	(1 1 0 0 1 0 1)
(1 1 0 1)	(0 0 0 1 1 0 1)
(0 0 1 1)	(0 1 0 0 0 1 1)
(1 0 1 1)	(1 0 0 1 0 1 1)
(0 1 1 1)	(0 0 1 0 1 1 1)
(1 1 1 1)	(1 1 1 1 1 1 1)

If $\mathbf{u} = (1 \ 1 \ 0 \ 1)$ is the message to be encoded, its corresponding codeword, according to (3.3), will be

$$\begin{aligned}\mathbf{v} &= 1 \cdot \mathbf{g}_0 + 1 \cdot \mathbf{g}_1 + 0 \cdot \mathbf{g}_2 + 1 \cdot \mathbf{g}_3 \\ &= (1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0) + (0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0) + (1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1) \\ &= (0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1).\end{aligned}$$

A desirable property for a linear block code to possess is the *systematic structure* of the codewords, as shown in Figure 3.1, in which a codeword is divided into two parts, the message part and the redundant checking part. The message part consists of k unaltered information (or message) digits, and the redundant checking part consists of $n - k$ parity-check digits, which are *linear sums* of the information digits. A linear block code with this structure is referred to as a *linear systematic block code*. The $(7, 4)$ code given in Table 3.1 is a linear systematic block

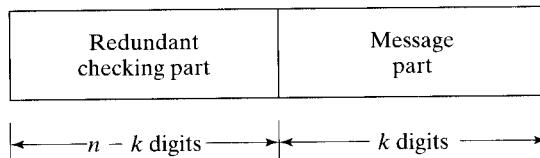


FIGURE 3.1: Systematic format of a codeword.

code; the rightmost four digits of each codeword are identical to the corresponding information digits.

A linear systematic (n, k) code is completely specified by a $k \times n$ matrix \mathbf{G} of the following form:

$$\mathbf{G} = \left[\begin{array}{c} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_{k-1} \end{array} \right] = \left[\begin{array}{ccccc} p_{00} & p_{01} & \cdots & p_{0,n-k-1} & \\ p_{10} & p_{11} & \cdots & p_{1,n-k-1} & \\ p_{20} & p_{21} & \cdots & p_{2,n-k-1} & \\ \vdots & \vdots & \ddots & \vdots & \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} & \end{array} \right] \left[\begin{array}{cccc} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{array} \right]. \quad (3.4)$$

where $p_{ij} = 0$ or 1 . Let \mathbf{I}_k denote the $k \times k$ identity matrix. Then, $\mathbf{G} = [\mathbf{P} \ \mathbf{I}_k]$. Let $\mathbf{u} = (u_0, u_1, \dots, u_{k-1})$ be the message to be encoded. The corresponding codeword is

$$\begin{aligned} \mathbf{v} &= (v_0, v_1, v_2, \dots, v_{n-1}) \\ &= (u_0, u_1, \dots, u_{k-1}) \cdot \mathbf{G}. \end{aligned} \quad (3.5)$$

It follows from (3.4) and (3.5) that the components of \mathbf{v} are

$$v_{n-k+i} = u_i \quad \text{for } 0 \leq i < k \quad (3.6a)$$

and

$$v_j = u_0 p_{0j} + u_1 p_{1j} + \cdots + u_{k-1} p_{kj} \quad (3.6b)$$

for $0 \leq j < n - k$. Equation (3.6a) shows that the rightmost k digits of a codeword \mathbf{v} are identical to the information digits u_0, u_1, \dots, u_{k-1} to be encoded, and (3.6b) shows that the leftmost $n - k$ redundant digits are linear sums of the information digits. The $n - k$ equations given by (3.6b) are called *parity-check equations* of the code.

EXAMPLE 3.2

The matrix \mathbf{G} given in Example 3.1 is in systematic form. Let $\mathbf{u} = (u_0, u_1, u_2, u_3)$ be the message to be encoded, and let $\mathbf{v} = (v_0, v_1, v_2, v_3, v_4, v_5, v_6)$ be the corresponding codeword. Then,

$$\mathbf{v} = (u_0, u_1, u_2, u_3) \cdot \left[\begin{array}{cccccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array} \right].$$

By matrix multiplication, we obtain the following digits of the codeword \mathbf{v} :

$$v_6 = u_3$$

$$v_5 = u_2$$

$$v_4 = u_1$$

$$v_3 = u_0$$

$$v_2 = u_1 + u_2 + u_3$$

$$v_1 = u_0 + u_1 + u_2$$

$$v_0 = u_0 + u_2 + u_3.$$

The codeword corresponding to the message $(1 \ 0 \ 1 \ 1)$ is $(1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1)$.

There is another useful matrix associated with every linear block code. As stated in Chapter 2, for any $k \times n$ matrix \mathbf{G} with k linearly independent rows, there exists an $(n - k) \times n$ matrix \mathbf{H} with $n - k$ linearly independent rows such that any vector in the row space of \mathbf{G} is orthogonal to the rows of \mathbf{H} , and any vector that is orthogonal to the rows of \mathbf{H} is in the row space of \mathbf{G} . Hence, we can describe the (n, k) linear code C generated by \mathbf{G} in an alternative way as follows: *An n -tuple \mathbf{v} is a codeword in the code C generated by \mathbf{G} if and only if $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$.* The code is said to be the null space of \mathbf{H} . This matrix \mathbf{H} is called a *parity-check matrix* of the code. The 2^{n-k} linear combinations of the rows of matrix \mathbf{H} form an $(n, n - k)$ linear code C_d . This code is the null space of the (n, k) linear code C generated by matrix \mathbf{G} (i.e., for any $\mathbf{v} \in C$ and any $\mathbf{w} \in C_d$, $\mathbf{v} \cdot \mathbf{w} = 0$). C_d is called the *dual code* of C . Therefore, a parity-check matrix for a linear code C is a generator matrix for its dual code C_d .

If the generator matrix of an (n, k) linear code is in the systematic form of (3.4), the parity-check matrix may take the following form:

$$\mathbf{H} = [\mathbf{I}_{n-k} \mathbf{P}^T] = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & p_{00} & p_{10} & \cdots & p_{k-1,0} \\ 0 & 1 & 0 & \cdots & 0 & p_{01} & p_{11} & \cdots & p_{k-1,1} \\ 0 & 0 & 1 & \cdots & 0 & p_{02} & p_{12} & \cdots & p_{k-1,2} \\ \vdots & & & & & & & & \\ 0 & 0 & 0 & \cdots & 1 & p_{0,n-k-1} & p_{1,n-k-1} & \cdots & p_{k-1,n-k-1} \end{bmatrix}, \quad (3.7)$$

where \mathbf{P}^T is the transpose of the matrix \mathbf{P} . Let \mathbf{h}_j be the j th row of \mathbf{H} . We can readily check that the inner product of the i th row of \mathbf{G} given by (3.4) and the j th row of \mathbf{H} given by (3.7) is

$$\mathbf{g}_i \cdot \mathbf{h}_j = p_{ij} + p_{ij} = 0$$

for $0 \leq i < k$ and $0 \leq j < n - k$. This implies that $\mathbf{G} \cdot \mathbf{H}^T = \mathbf{0}$. Also, the $n - k$ rows of \mathbf{H} are linearly independent. Therefore, the \mathbf{H} matrix of (3.7) is a parity-check matrix of the (n, k) linear code generated by the matrix \mathbf{G} of (3.4).

The parity-check equations given by (3.6b) also can be obtained from the parity-check matrix \mathbf{H} of (3.7). Let $\mathbf{u} = (u_0, u_1, \dots, u_{k-1})$ be the message to be encoded. In systematic form the corresponding codeword will be

$$\mathbf{v} = (v_0, v_1, \dots, v_{n-k-1}, u_0, u_1, \dots, u_{k-1}).$$

Using the fact that $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$, we obtain

$$v_j + u_0 p_{0j} + u_1 p_{1j} + \cdots + u_{k-1} p_{k-1,j} = 0 \quad (3.8)$$

for $0 \leq j < n - k$. Rearranging the equations of (3.8), we obtain the same parity-check equations of (3.6b). Therefore, an (n, k) linear code is completely specified by its parity-check matrix.

EXAMPLE 3.3

Consider the generator matrix of the $(7, 4)$ linear code given in Example 3.1. The corresponding parity-check matrix is

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

At this point we summarize the foregoing results: For any (n, k) linear block code C there exists a $k \times n$ matrix \mathbf{G} whose row space gives C . Furthermore, there exists an $(n - k) \times n$ matrix \mathbf{H} such that an n -tuple \mathbf{v} is a codeword in C if and only if $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$. If \mathbf{G} is of the form given by (3.4), then \mathbf{H} may take the form given by (3.7), and vice versa.

Based on the equations of (3.6a) and (3.6b), the encoding circuit for an (n, k) linear systematic code can easily be implemented. The encoding circuit is shown in Figure 3.2, where $\rightarrow \square \rightarrow$ denotes a shift-register stage (e.g., a flip-flop), \oplus denotes a modulo-2 adder, and $\xrightarrow{p_{ij}}$ denotes a connection if $p_{ij} = 1$, and no connection if $p_{ij} = 0$. The encoding operation is very simple. The message $\mathbf{u} = (u_0, u_1, \dots, u_{k-1})$ to be encoded is shifted into the message register and simultaneously into the channel. As soon as the entire message has entered the message register the $n - k$ parity-check digits are formed at the outputs of the $n - k$ modulo-2 adders. These parity-check digits are then serialized and shifted into the channel. We see that the complexity of the encoding circuit is linearly proportional to the block length of the code. The encoding circuit for the $(7, 4)$ code given in Table 3.1 is shown in Figure 3.3, where the connection is based on the parity-check equations given in Example 3.2.

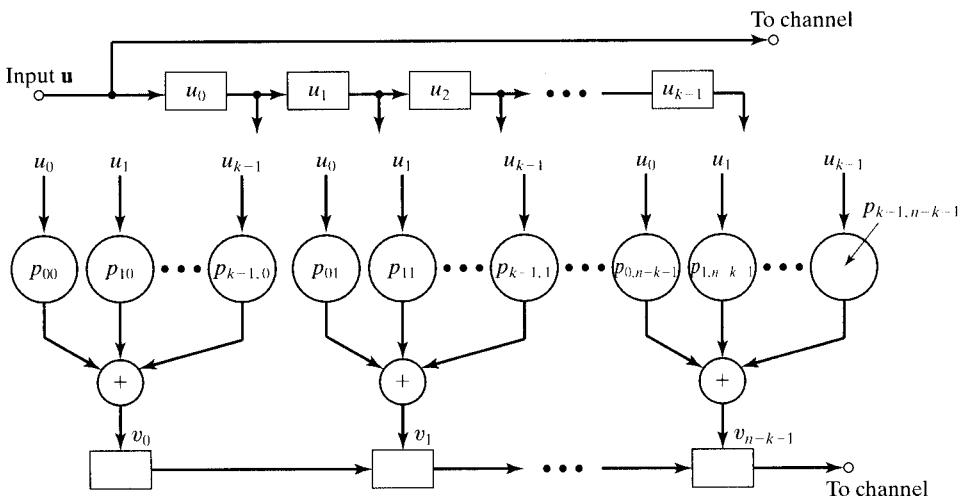


FIGURE 3.2: Encoding circuit for a linear systematic (n, k) code.

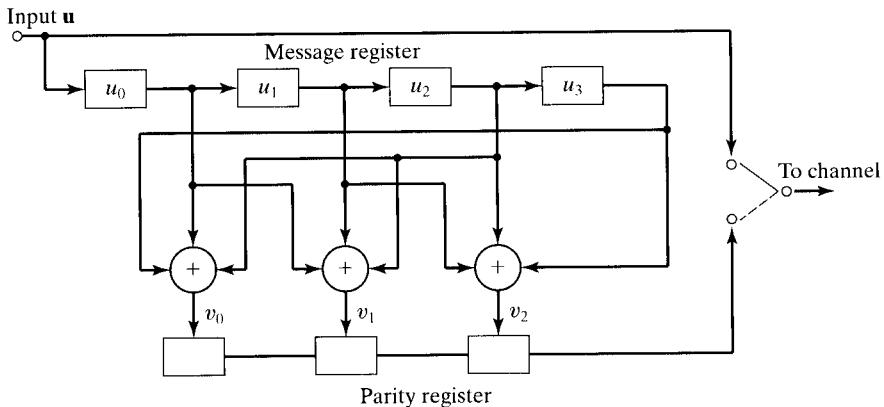


FIGURE 3.3: Encoding circuit for the $(7, 4)$ systematic code given in Table 3.1.

3.2 SYNDROME AND ERROR DETECTION

Consider an (n, k) linear code with generator matrix \mathbf{G} and parity-check matrix \mathbf{H} . Let $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ be a codeword that was transmitted over a noisy channel. Let $\mathbf{r} = (r_0, r_1, \dots, r_{n-1})$ be the received vector at the output of the channel. Because of the channel noise, \mathbf{r} may be different from \mathbf{v} . The vector sum

$$\begin{aligned}\mathbf{e} &= \mathbf{r} + \mathbf{v} \\ &= (e_0, e_1, \dots, e_{n-1})\end{aligned}\tag{3.9}$$

is an n -tuple, where $e_i = 1$ for $r_i \neq v_i$, and $e_i = 0$ for $r_i = v_i$. This n -tuple is called the *error vector* (or *error pattern*), which simply displays the positions where the received vector \mathbf{r} differ from the transmitted codeword \mathbf{v} . The 1's in \mathbf{e} are the *transmission errors* caused by the channel noise. It follows from (3.9) that the received vector \mathbf{r} is the vector sum of the transmitted codeword and the error vector; that is,

$$\mathbf{r} = \mathbf{v} + \mathbf{e}.$$

Of course, the receiver does not know either \mathbf{v} or \mathbf{e} . On receiving \mathbf{r} , the decoder must first determine whether \mathbf{r} contains transmission errors. If the presence of errors is detected, the decoder will either take actions to locate the errors and correct them (FEC) or request a retransmission of \mathbf{v} (ARQ).

When \mathbf{r} is received, the decoder computes the following $(n - k)$ -tuple:

$$\begin{aligned}\mathbf{s} &= \mathbf{r} \cdot \mathbf{H}^T \\ &= (s_0, s_1, \dots, s_{n-k-1}).\end{aligned}\tag{3.10}$$

which is called the *syndrome* of \mathbf{r} . Then, $\mathbf{s} = \mathbf{0}$ if and only if \mathbf{r} is a codeword, and $\mathbf{s} \neq \mathbf{0}$ if and only if \mathbf{r} is not a codeword. Therefore, when $\mathbf{s} \neq \mathbf{0}$, we know that \mathbf{r} is not a codeword and the presence of errors has been detected. When $\mathbf{s} = \mathbf{0}$, \mathbf{r} is a codeword, and the receiver accepts \mathbf{r} as the transmitted codeword. It is possible that the errors in certain error vectors are not detectable (i.e., \mathbf{r} contains errors but $\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T = \mathbf{0}$).

This happens when the error pattern \mathbf{e} is identical to a nonzero codeword. In this event, \mathbf{r} is the sum of two codewords, which is a codeword, and consequently $\mathbf{r} \cdot \mathbf{H}^T = \mathbf{0}$. Error patterns of this kind are called *undetectable* error patterns. Because there are $2^k - 1$ nonzero codewords, there are $2^k - 1$ undetectable error patterns. When an undetectable error pattern occurs, the decoder makes a *decoding error*. In a later section of this chapter we derive the probability of an undetected error for a BSC and show that this error probability can be made very small.

Based on (3.7) and (3.10), the syndrome digits are as follows:

$$\begin{aligned}s_0 &= r_0 + r_{n-k} p_{00} + r_{n-k+1} p_{10} + \cdots + r_{n-1} p_{k-1,0} \\s_1 &= r_1 + r_{n-k} p_{01} + r_{n-k+1} p_{11} + \cdots + r_{n-1} p_{k-1,1} \\&\vdots \\s_{n-k-1} &= r_{n-k-1} + r_{n-k} p_{0,n-k-1} + r_{n-k+1} p_{1,n-k-1} + \cdots + r_{n-1} p_{k-1,n-k-1}.\end{aligned}\tag{3.11}$$

If we examine the preceding equations carefully, we find that the syndrome \mathbf{s} is simply the vector sum of the received parity digits ($r_0, r_1, \dots, r_{n-k-1}$) and the parity-check digits recomputed from the received information digits ($r_{n-k}, r_{n-k+1}, \dots, r_{n-1}$). Therefore, the syndrome can be formed by a circuit similar to the encoding circuit. A general syndrome circuit is shown in Figure 3.4.

EXAMPLE 3.4

Consider the (7, 4) linear code whose parity-check matrix is given in Example 3.3. Let $\mathbf{r} = (r_0, r_1, r_2, r_3, r_4, r_5, r_6)$ be the received vector. Then, the syndrome is given by

$$\mathbf{s} = (s_0, s_1, s_2)$$

$$= (r_0, r_1, r_2, r_3, r_4, r_5, r_6) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}.$$

The syndrome digits are

$$s_0 = r_0 + r_3 + r_5 + r_6$$

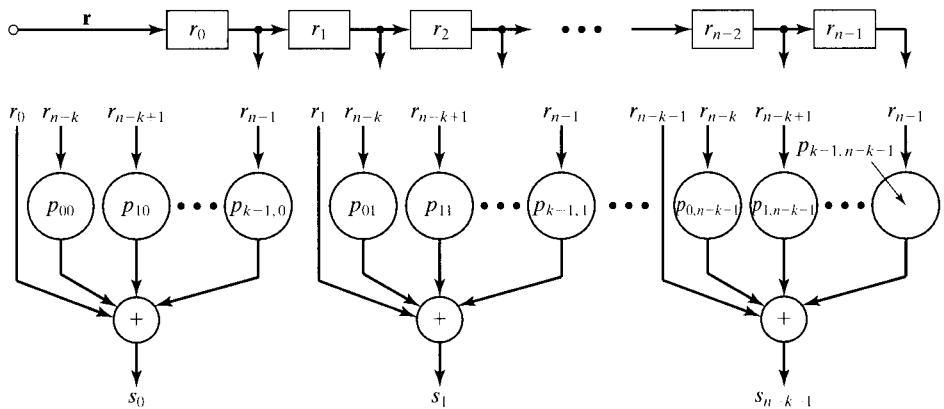
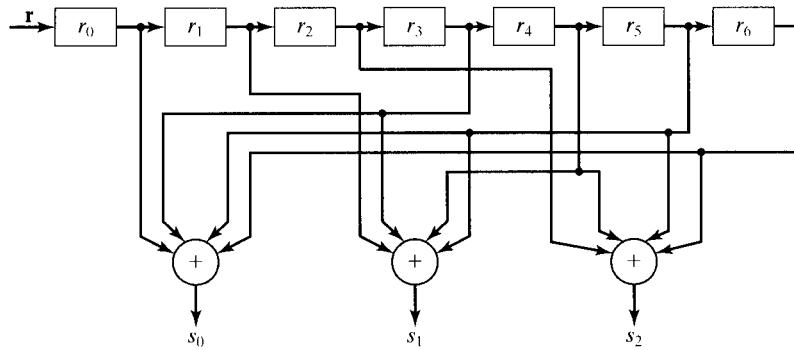
$$s_1 = r_1 + r_3 + r_4 + r_5$$

$$s_2 = r_2 + r_4 + r_5 + r_6.$$

The syndrome circuit for this code is shown in Figure 3.5.

The syndrome \mathbf{s} computed from the received vector \mathbf{r} depends only on the error pattern \mathbf{e} and not on the transmitted codeword \mathbf{v} . Because \mathbf{r} is the vector sum of \mathbf{v} and \mathbf{e} , it follows from (3.10) that

$$\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T = (\mathbf{v} + \mathbf{e}) \mathbf{H}^T = \mathbf{v} \cdot \mathbf{H}^T + \mathbf{e} \cdot \mathbf{H}^T;$$

FIGURE 3.4: Syndrome circuit for a linear systematic (n, k) code.FIGURE 3.5: Syndrome circuit for the $(7, 4)$ code given in Table 3.1.

however, $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$. Consequently, we obtain the following relation between the syndrome and the error pattern:

$$\mathbf{s} = \mathbf{e} \cdot \mathbf{H}^T. \quad (3.12)$$

If the parity-check matrix \mathbf{H} is expressed in the systematic form as given by (3.7), multiplying out $\mathbf{e} \cdot \mathbf{H}^T$ yields the following linear relationship between the syndrome digits and the error digits:

$$\begin{aligned} s_0 &= e_0 + e_{n-k} p_{00} + e_{n-k+1} p_{10} + \cdots + e_{n-1} p_{k-1,0} \\ s_1 &= e_1 + e_{n-k} p_{01} + e_{n-k+1} p_{11} + \cdots + e_{n-1} p_{k-1,1} \\ &\vdots \\ s_{n-k-1} &= e_{n-k-1} + e_{n-k} p_{0,n-k-1} + e_{n-k+1} p_{1,n-k-1} + \cdots + e_{n-1} p_{k-1,n-k-1}. \end{aligned} \quad (3.13)$$

The syndrome digits are simply linear combinations of the error digits. Clearly, they provide information about the error digits and therefore can be used for error correction.

At this point, one would think that any error correction scheme would be a method of solving the $n - k$ linear equations of (3.13) for the error digits. Once the error pattern \mathbf{e} was found, the vector $\mathbf{r} + \mathbf{e}$ would be taken as the actual transmitted codeword. Unfortunately, determining the true error vector \mathbf{e} is not a simple matter. This is because the $n - k$ linear equations of (3.13) do not have a unique solution but have 2^k solutions (this will be proved in Theorem 3.6). In other words, there are 2^k error patterns that result in the same syndrome, and the true error pattern \mathbf{e} is just one of them. Therefore, the decoder has to determine the true error vector from a set of 2^k candidates. To minimize the probability of a decoding error, the most *probable* error pattern that satisfies the equations of (3.13) is chosen as the true error vector. If the channel is a BSC, the most probable error pattern is the one that has the smallest number of nonzero digits.

The notion of using syndrome for error correction may be clarified by an example.

EXAMPLE 3.5

Again, we consider the $(7, 4)$ code whose parity-check matrix is given in Example 3.3. Let $\mathbf{v} = (1001011)$ be the transmitted codeword and $\mathbf{r} = (1001001)$ be the received vector. On receiving \mathbf{r} , the receiver computes the syndrome:

$$\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T = (1\ 1\ 1).$$

Next, the receiver attempts to determine the true error vector $\mathbf{e} = (e_0, e_1, e_2, e_3, e_4, e_5, e_6)$, which yields the given syndrome. It follows from (3.12) or (3.13) that the error digits are related to the syndrome digits by the following linear equations:

$$1 = e_0 + e_3 + e_5 + e_6$$

$$1 = e_1 + e_3 + e_4 + e_5$$

$$1 = e_2 + e_4 + e_5 + e_6.$$

There are $2^4 = 16$ error patterns that satisfy the preceding equations, namely,

$$\begin{array}{ll} (0000010), & (1010011), \\ (1101010), & (0111011), \\ (0110110), & (1100111), \\ (1011110), & (0001111), \\ (1110000), & (0100001), \\ (0011000), & (1001001), \\ (1000100), & (0010101), \\ (0101100), & (1111101). \end{array}$$

The error vector $\mathbf{e} = (0000010)$ has the smallest number of nonzero components. If the channel is a BSC, $\mathbf{e} = (0000010)$ is the most probable error vector that satisfies the preceding equations. Taking $\mathbf{e} = (0000010)$ as the true error vector, the

receiver decodes the received vector $\mathbf{r} = (1001001)$ into the following codeword:

$$\begin{aligned}\mathbf{v}^* &= \mathbf{r} + \mathbf{e} \\ &= (1001001) + (0000010) \\ &= (1001011).\end{aligned}$$

We see that \mathbf{v}^* is the actual transmitted codeword. Hence, the receiver has performed a correct decoding. Later we show that the $(7, 4)$ linear code considered in this example is capable of correcting any single error over a span of seven digits; that is, if a codeword is transmitted and if only one digit is changed by the channel noise, the receiver will be able to determine the true error vector and to perform a correct decoding.

More discussion on error correction based on syndrome is given in Section 3.5. Various methods of determining the true error pattern from the $n-k$ linear equations of (3.13) are presented in later chapters.

3.3 THE MINIMUM DISTANCE OF A BLOCK CODE

In this section we introduce an important parameter of a block code called the *minimum distance*. This parameter determines the random-error-detecting and random-error-correcting capabilities of a code. Let $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ be a binary n -tuple. The *Hamming weight* (or simply *weight*) of \mathbf{v} , denoted by $w(\mathbf{v})$, is defined as the number of nonzero components of \mathbf{v} . For example, the Hamming weight of $\mathbf{v} = (1001011)$ is 4. Let \mathbf{v} and \mathbf{w} be two n -tuples. The *Hamming distance* (or simply *distance*) between \mathbf{v} and \mathbf{w} , denoted $d(\mathbf{v}, \mathbf{w})$, is defined as the number of places where they differ. For example, the Hamming distance between $\mathbf{v} = (1001011)$ and $\mathbf{w} = (0100011)$ is 3; they differ in the zeroth, first, and third places. The Hamming distance is a metric function that satisfies the *triangle inequality*. Let \mathbf{v} , \mathbf{w} , and \mathbf{x} be three n -tuples. Then,

$$d(\mathbf{v}, \mathbf{w}) + d(\mathbf{w}, \mathbf{x}) \geq d(\mathbf{v}, \mathbf{x}). \quad (3.14)$$

(The proof of this inequality is left as a problem.) It follows from the definition of Hamming distance and the definition of modulo-2 addition that the Hamming distance between two n -tuples \mathbf{v} and \mathbf{w} is equal to the Hamming weight of the sum of \mathbf{v} and \mathbf{w} ; that is,

$$d(\mathbf{v}, \mathbf{w}) = w(\mathbf{v} + \mathbf{w}). \quad (3.15)$$

For example, the Hamming distance between $\mathbf{v} = (1001011)$ and $\mathbf{w} = (1110010)$ is 4, and the weight of $\mathbf{v} + \mathbf{w} = (0111001)$ is also 4.

Given a block code C , one can compute the Hamming distance between any two distinct codewords. The *minimum distance* of C , denoted by d_{\min} , is defined as

$$d_{\min} \stackrel{\Delta}{=} \min\{d(\mathbf{v}, \mathbf{w}) : \mathbf{v}, \mathbf{w} \in C, \mathbf{v} \neq \mathbf{w}\}. \quad (3.16)$$

If C is a linear block code, the sum of two codewords is also a codeword. It follows from (3.15) that the Hamming distance between two codewords in C is equal to the

Hamming weight of a third codeword in C . Then, it follows from (3.16) that

$$\begin{aligned} d_{\min} &= \min\{w(\mathbf{v} + \mathbf{w}) : \mathbf{v}, \mathbf{w} \in C, \mathbf{v} \neq \mathbf{w}\} \\ &= \min\{w(\mathbf{x}) : \mathbf{x} \in C, \mathbf{x} \neq \mathbf{0}\} \\ &\stackrel{\Delta}{=} w_{\min}. \end{aligned} \quad (3.17)$$

The parameter $w_{\min} \stackrel{\Delta}{=} \{w(\mathbf{x}) : \mathbf{x} \in C, \mathbf{x} \neq \mathbf{0}\}$ is called the *minimum weight* of the linear code C . Summarizing the preceding result, we have the following theorem.

THEOREM 3.1 The minimum distance of a linear block code is equal to the minimum weight of its nonzero codewords and vice versa.

Therefore, for a linear block code, determining the minimum distance of the code is equivalent to determining its minimum weight. The $(7, 4)$ code given in Table 3.1 has minimum weight 3; thus, its minimum distance is 3. Next, we prove a number of theorems that relate the weight structure of a linear block code to its parity-check matrix.

THEOREM 3.2 Let C be an (n, k) linear code with parity-check matrix \mathbf{H} . For each codeword of Hamming weight l , there exist l columns of \mathbf{H} such that the vector sum of these l columns is equal to the zero vector. Conversely, if there exist l columns of \mathbf{H} whose vector sum is the zero vector, there exists a codeword of Hamming weight l in C .

Proof. We express the parity-check matrix in the following form:

$$\mathbf{H} = [\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_{n-1}],$$

where \mathbf{h}_i represents the i th column of \mathbf{H} . Let $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ be a codeword of weight l . Then, \mathbf{v} has l nonzero components. Let $v_{i_1}, v_{i_2}, \dots, v_{i_l}$ be the l nonzero components of \mathbf{v} , where $0 \leq i_1 < i_2 < \dots < i_l \leq n - 1$. Then, $v_{i_1} = v_{i_2} = \dots = v_{i_l} = 1$. Because \mathbf{v} is a codeword, we must have

$$\begin{aligned} \mathbf{0} &= \mathbf{v} \cdot \mathbf{H}^T \\ &= v_0 \mathbf{h}_0 + v_1 \mathbf{h}_1 + \dots + v_{n-1} \mathbf{h}_{n-1} \\ &= v_{i_1} \mathbf{h}_{i_1} + v_{i_2} \mathbf{h}_{i_2} + \dots + v_{i_l} \mathbf{h}_{i_l} \\ &= \mathbf{h}_{i_1} + \mathbf{h}_{i_2} + \dots + \mathbf{h}_{i_l}. \end{aligned}$$

This proves the first part of the theorem.

Now, suppose that $\mathbf{h}_{i_1}, \mathbf{h}_{i_2}, \dots, \mathbf{h}_{i_l}$ are l columns of \mathbf{H} such that

$$\mathbf{h}_{i_1} + \mathbf{h}_{i_2} + \dots + \mathbf{h}_{i_l} = \mathbf{0}. \quad (3.18)$$

We form a binary n -tuple $\mathbf{x} = (x_0, x_1, \dots, x_{n-1})$ whose nonzero components are $x_{i_1}, x_{i_2}, \dots, x_{i_l}$. The Hamming weight of \mathbf{x} is l . Consider the product

$$\begin{aligned} \mathbf{x} \cdot \mathbf{H}^T &= x_0 \mathbf{h}_0 + x_1 \mathbf{h}_1 + \dots + x_{n-1} \mathbf{h}_{n-1} \\ &= x_{i_1} \mathbf{h}_{i_1} + x_{i_2} \mathbf{h}_{i_2} + \dots + x_{i_l} \mathbf{h}_{i_l} \\ &= \mathbf{h}_{i_1} + \mathbf{h}_{i_2} + \dots + \mathbf{h}_{i_l}. \end{aligned}$$

It follows from (3.18) that $\mathbf{x} \cdot \mathbf{H}^T = \mathbf{0}$. Thus, \mathbf{x} is a codeword of weight l in C . This proves the second part of the theorem. **Q.E.D.**

The following two corollaries follow from Theorem 3.2.

COROLLARY 3.2.1 Let C be a linear block code with parity-check matrix \mathbf{H} . If no $d - 1$ or fewer columns of \mathbf{H} add to $\mathbf{0}$, the code has minimum weight at least d .

COROLLARY 3.2.2 Let C be a linear code with parity-check matrix \mathbf{H} . The minimum weight (or the minimum distance) of C is equal to the smallest number of columns of \mathbf{H} that sum to $\mathbf{0}$.

Consider the $(7, 4)$ linear code given in Table 3.1. The parity-check matrix of this code is

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

We see that all columns of \mathbf{H} are nonzero and that no two of them are alike. Therefore, no two or fewer columns sum to $\mathbf{0}$. Hence, the minimum weight of this code is at least 3; however, the zeroth, second, and sixth columns sum to $\mathbf{0}$. Thus, the minimum weight of the code is 3. From Table 3.1 we see that the minimum weight of the code is indeed 3. It follows from Theorem 3.1 that the minimum distance is 3.

Corollaries 3.2.1 and 3.2.2 are generally used to determine the minimum distance or to establish a lower bound on the minimum distance of a linear block code.

3.4 ERROR-DETECTING AND ERROR-CORRECTING CAPABILITIES OF A BLOCK CODE

When a codeword \mathbf{v} is transmitted over a noisy channel, an error pattern of l errors will result in a received vector \mathbf{r} that differs from the transmitted codeword \mathbf{v} in l places [i.e., $d(\mathbf{v}, \mathbf{r}) = l$]. If the minimum distance of a block code C is d_{min} , any two distinct codewords of C differ in at least d_{min} places. For this code C , no error pattern of $d_{min} - 1$ or fewer errors can change one codeword into another. Therefore, any error pattern of $d_{min} - 1$ or fewer errors will result in a received vector \mathbf{r} that is not a codeword in C . When the receiver detects that the received vector is not a codeword of C , we say that errors are detected. Hence, a block code with minimum distance d_{min} is capable of detecting all the error patterns of $d_{min} - 1$ or fewer errors. However, it cannot detect all the error patterns of d_{min} errors because there exists at least one pair of codewords that differ in d_{min} places, and there is an error pattern of d_{min} errors that will carry one into the other. The same argument applies to error patterns of more than d_{min} errors. For this reason we say that the random-error-detecting capability of a block code with minimum distance d_{min} is $d_{min} - 1$.

Even though a block code with minimum distance d_{min} guarantees detection of all the error patterns of $d_{min} - 1$ or fewer errors, it is also capable of detecting a large fraction of error patterns with d_{min} or more errors. In fact, an (n, k) linear code is capable of detecting $2^n - 2^k$ error patterns of length n . This can be shown as follows. Among the $2^n - 1$ possible nonzero error patterns, there are $2^k - 1$ error

patterns that are identical to the $2^k - 1$ nonzero codewords. If any of these $2^k - 1$ error patterns occurs, it alters the transmitted codeword \mathbf{v} into another codeword \mathbf{w} . Thus, \mathbf{w} is received, and its syndrome is zero. In this case, the decoder accepts \mathbf{w} as the transmitted codeword and thus performs an incorrect decoding. Therefore, there are $2^k - 1$ *undetectable* error patterns. If an error pattern is not identical to a nonzero codeword, the received vector \mathbf{r} will not be a codeword and the syndrome will not be zero. In this case, an error will be detected. There are exactly $2^n - 2^k$ error patterns that are not identical to the codewords of an (n, k) linear code. These $2^n - 2^k$ error patterns are *detectable* error patterns. For large n , $2^k - 1$ is, in general, much smaller than 2^n . Therefore, only a small fraction of error patterns pass through the decoder without being detected.

Let C be an (n, k) linear code. Let A_i be the number of codewords of weight i in C . The numbers A_0, A_1, \dots, A_n are called the *weight distribution* of C . If C is used only for error detection on a BSC, the probability that the decoder will fail to detect the presence of errors can be computed from the weight distribution of C . Let $P_u(E)$ denote the probability of an undetected error. Because an undetected error occurs only when the error pattern is identical to a nonzero codeword of C ,

$$P_u(E) = \sum_{i=1}^n A_i p^i (1-p)^{n-i}, \quad (3.19)$$

where p is the transition probability of the BSC. If the minimum distance of C is d_{min} , then A_1 to $A_{d_{min}-1}$ are zero.

Consider the $(7, 4)$ code given in Table 3.1. The weight distribution of this code is $A_0 = 1, A_1 = A_2 = 0, A_3 = 7, A_4 = 7, A_5 = A_6 = 0$, and $A_7 = 1$. The probability of an undetected error is

$$P_u(E) = 7p^3(1-p)^4 + 7p^4(1-p)^3 + p^7.$$

If $p = 10^{-2}$, this probability is approximately 7×10^{-6} . In other words, if 1 million codewords are transmitted over a BSC with $p = 10^{-2}$, on average seven erroneous codewords pass through the decoder without being detected.

If a block code C with minimum distance d_{min} is used for random-error correction, one would like to know how many errors the code is able to correct. The minimum distance d_{min} is either odd or even. Let t be a positive integer such that

$$2t + 1 \leq d_{min} \leq 2t + 2. \quad (3.20)$$

Next, we show that the code C is capable of correcting all the error patterns of t or fewer errors. Let \mathbf{v} and \mathbf{r} be the transmitted codeword and the received vector, respectively. Let \mathbf{w} be any other codeword in C . The Hamming distances among \mathbf{v} , \mathbf{r} , and \mathbf{w} satisfy the triangle inequality:

$$d(\mathbf{v}, \mathbf{r}) + d(\mathbf{w}, \mathbf{r}) \geq d(\mathbf{v}, \mathbf{w}). \quad (3.21)$$

Suppose that an error pattern of t' errors occurs during the transmission of \mathbf{v} . Then, the received vector \mathbf{r} differs from \mathbf{v} in t' places, and therefore $d(\mathbf{v}, \mathbf{r}) = t'$. Because \mathbf{v} and \mathbf{w} are codewords in C , we have

$$d(\mathbf{v}, \mathbf{w}) \geq d_{min} \geq 2t + 1. \quad (3.22)$$

Combining (3.21) and (3.22) and using the fact that $d(\mathbf{v}, \mathbf{r}) = t'$, we obtain the following inequality:

$$d(\mathbf{w}, \mathbf{r}) \geq 2t + 1 - t'.$$

If $t' \leq t$,

$$d(\mathbf{w}, \mathbf{r}) > t.$$

The preceding inequality says that if an error pattern of t or fewer errors occurs, the received vector \mathbf{r} is closer (in Hamming distance) to the transmitted codeword \mathbf{v} than to any other codeword \mathbf{w} in C . For a BSC, this means that the conditional probability $P(\mathbf{r}|\mathbf{v})$ is greater than the conditional probability $P(\mathbf{r}|\mathbf{w})$ for $\mathbf{w} \neq \mathbf{v}$. Based on the maximum likelihood decoding scheme, \mathbf{r} is decoded into \mathbf{v} , which is the actual transmitted codeword. The result is a correct decoding, and thus errors are corrected.

In contrast, the code is not capable of correcting all the error patterns of l errors with $l > t$, for there is at least one case in which an error pattern of l errors results in a received vector that is closer to an incorrect codeword than to the transmitted codeword. To show this, let \mathbf{v} and \mathbf{w} be two codewords in C such that

$$d(\mathbf{v}, \mathbf{w}) = d_{\min}.$$

Let \mathbf{e}_1 and \mathbf{e}_2 be two error patterns that satisfy the following conditions:

- i. $\mathbf{e}_1 + \mathbf{e}_2 = \mathbf{v} + \mathbf{w}$.
- ii. \mathbf{e}_1 and \mathbf{e}_2 do not have nonzero components in common places.

Obviously, we have

$$w(\mathbf{e}_1) + w(\mathbf{e}_2) = w(\mathbf{v} + \mathbf{w}) = d(\mathbf{v}, \mathbf{w}) = d_{\min}. \quad (3.23)$$

Now, suppose that \mathbf{v} is transmitted and is corrupted by the error pattern \mathbf{e}_1 . Then, the received vector is

$$\mathbf{r} = \mathbf{v} + \mathbf{e}_1.$$

The Hamming distance between \mathbf{v} and \mathbf{r} is

$$d(\mathbf{v}, \mathbf{r}) = w(\mathbf{v} + \mathbf{r}) = w(\mathbf{e}_1). \quad (3.24)$$

The Hamming distance between \mathbf{w} and \mathbf{r} is

$$d(\mathbf{w}, \mathbf{r}) = w(\mathbf{w} + \mathbf{r}) = w(\mathbf{w} + \mathbf{v} + \mathbf{e}_1) = w(\mathbf{e}_2). \quad (3.25)$$

Now, suppose that the error pattern \mathbf{e}_1 contains more than t errors (i.e., $w(\mathbf{e}_1) > t$). Because $2t + 1 \leq d_{\min} \leq 2t + 2$, it follows from (3.23) that

$$w(\mathbf{e}_2) \leq t + 1.$$

Combining (3.24) and (3.25) and using the fact that $w(\mathbf{e}_1) > t$ and $w(\mathbf{e}_2) \leq t + 1$, we obtain the following inequality:

$$d(\mathbf{v}, \mathbf{r}) \geq d(\mathbf{w}, \mathbf{r}).$$

This inequality says that there exists an error pattern of l ($l > t$) errors that results in a received vector that is closer to an incorrect codeword than to the transmitted codeword. Based on the maximum likelihood decoding scheme, an incorrect decoding would be performed.

In summary, a block code with minimum distance d_{\min} guarantees correction of all the error patterns of $t = \lfloor (d_{\min} - 1)/2 \rfloor$ or fewer errors, where $\lfloor (d_{\min} - 1)/2 \rfloor$ denotes the largest integer no greater than $(d_{\min} - 1)/2$. The parameter $t = \lfloor (d_{\min} - 1)/2 \rfloor$ is called the *random-error-correcting capability* of the code. The code is referred to as a t -error-correcting code. The (7, 4) code given in Table 3.1 has minimum distance 3 and thus $t = 1$. The code is capable of correcting any error pattern of single error over a block of seven digits.

A block code with random-error-correcting capability t is usually capable of correcting many error patterns of $t + 1$ or more errors. A t -error-correcting (n, k) linear code is capable of correcting a total of 2^{n-k} error patterns, including those with t or fewer errors (this will be seen in the next section). If a t -error-correcting block code is used strictly for error correction on a BSC with transition probability p , the probability that the decoder commits an erroneous decoding is upper bounded by

$$P(E) \leq \sum_{i=t+1}^n \binom{n}{i} p^i (1-p)^{n-i}. \quad (3.26)$$

In practice, a code is often used for correcting λ or fewer errors and simultaneously detecting l ($l > \lambda$) or fewer errors. That is, when λ or fewer errors occur, the code is capable of correcting them; when more than λ but fewer than $l + 1$ errors occur, the code is capable of detecting their presence without making a decoding error. For this purpose, the minimum distance d_{\min} of the code is at least $\lambda + l + 1$ (left as a problem). Thus, a block code with $d_{\min} = 10$ is capable of correcting three or fewer errors and simultaneously detecting six or fewer errors.

So far, we have considered only the case in which the receiver makes a hard-decision for each received symbol; however, a receiver may be designed to declare a symbol erased when it is received ambiguously (or unreliably). In this case, the received sequence consists of zeros, ones, or erasures. A code can be used to correct combinations of errors and erasures. A code with minimum distance d_{\min} is capable of correcting any pattern of v errors and e erasures provided the following condition

$$d_{\min} \geq 2v + e + 1$$

is satisfied. To see this, delete from all the codewords the e components where the receiver has declared erasures. This deletion results in a shortened code of length $n - e$. The minimum distance of this shortened code is at least $d_{\min} - e \geq 2v + 1$. Hence, v errors can be corrected in the unerased positions. As a result, the shortened codeword with e components erased can be recovered. Finally, because $d_{\min} \geq e + 1$, there is one and only one codeword in the original code that agrees with the unerased components. Consequently, the entire codeword can be recovered. This correction of combinations of errors and erasures is often used in practice.

From the preceding discussion we see that the random-error-detecting and random-error-correcting capabilities of a block code are determined by the code's minimum distance. Clearly, for a given n and k , one would like to construct a block

code with as large a minimum distance as possible, in addition to the implementation considerations.

Often an (n, k) linear block code with minimum distance d_{min} is denoted by (n, k, d_{min}) . The code given by Table 3.1 is a $(7, 4, 3)$ code.

3.5 STANDARD ARRAY AND SYNDROME DECODING

In this section we present a scheme for decoding linear block codes. Let C be an (n, k) linear code. Let $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_{2^k}$ be the codewords of C . No matter which codeword is transmitted over a noisy channel, the received vector \mathbf{r} may be any of the 2^n n -tuples over $GF(2)$. Any decoding scheme used at the receiver is a rule to partition the 2^n possible received vectors into 2^k disjoint subsets D_1, D_2, \dots, D_{2^k} such that the codeword \mathbf{v}_i is contained in the subset D_i for $1 \leq i \leq 2^k$. Thus, each subset D_i is one-to-one correspondence to a codeword \mathbf{v}_i . If the received vector \mathbf{r} is found in the subset D_i , \mathbf{r} is decoded into \mathbf{v}_i . Decoding is correct if and only if the received vector \mathbf{r} is in the subset D_i that corresponds to the codeword transmitted.

A method to partition the 2^n possible received vectors into 2^k disjoint subsets such that each subset contains one and only one codeword is described here. The partition is based on the linear structure of the code. First, we place the 2^k codewords of C in a row with the all-zero codeword $\mathbf{v}_1 = (0, 0, \dots, 0)$ as the first (leftmost) element. From the remaining $2^n - 2^k$ n -tuples, we choose an n -tuple \mathbf{e}_2 and place it under the zero vector \mathbf{v}_1 . Now, we form a second row by adding \mathbf{e}_2 to each codeword \mathbf{v}_i in the first row and placing the sum $\mathbf{e}_2 + \mathbf{v}_i$ under \mathbf{v}_i . Having completed the second row, we choose an unused n -tuple \mathbf{e}_3 from the remaining n -tuples and place it under \mathbf{v}_1 . Then, we form a third row by adding \mathbf{e}_3 to each codeword \mathbf{v}_i in the first row and placing $\mathbf{e}_3 + \mathbf{v}_i$ under \mathbf{v}_i . We continue this process until we have used all the n -tuples. Then, we have an array of rows and columns, as shown in Figure 3.6. This array is called a *standard array* of the given linear code C .

It follows from the construction rule of a standard array that the sum of any two vectors in the same row is a codeword in C . Next, we prove some important properties of a standard array.

THEOREM 3.3 No two n -tuples in the same row of a standard array are identical. Every n -tuple appears in one and only one row.

Proof. The first part of the theorem follows from the fact that all the codewords of C are distinct. Suppose that two n -tuples in the l th row are identical, say

$$\begin{array}{ccccccc}
 \mathbf{v}_1 & = 0 & \mathbf{v}_2 & \cdots & \mathbf{v}_i & \cdots & \mathbf{v}_{2^k} \\
 \mathbf{e}_2 & & \mathbf{e}_2 + \mathbf{v}_2 & \cdots & \mathbf{e}_2 + \mathbf{v}_i & \cdots & \mathbf{e}_2 + \mathbf{v}_{2^k} \\
 \mathbf{e}_3 & & \mathbf{e}_3 + \mathbf{v}_2 & \cdots & \mathbf{e}_3 + \mathbf{v}_i & \cdots & \mathbf{e}_3 + \mathbf{v}_{2^k} \\
 \vdots & & & & & & \vdots \\
 \mathbf{e}_l & & \mathbf{e}_l + \mathbf{v}_2 & \cdots & \mathbf{e}_l + \mathbf{v}_i & \cdots & \mathbf{e}_l + \mathbf{v}_{2^k} \\
 \vdots & & & & & & \vdots \\
 \mathbf{e}_{2^{n-k}} & & \mathbf{e}_{2^{n-k}} + \mathbf{v}_2 & \cdots & \mathbf{e}_{2^{n-k}} + \mathbf{v}_i & \cdots & \mathbf{e}_{2^{n-k}} + \mathbf{v}_{2^k}
 \end{array}$$

FIGURE 3.6: Standard array for an (n, k) linear code.

$\mathbf{e}_l + \mathbf{v}_i = \mathbf{e}_l + \mathbf{v}_j$ with $i \neq j$. This means that $\mathbf{v}_i = \mathbf{v}_j$, which is impossible. Therefore, no two n -tuples in the same row are identical.

It follows from the construction rule of the standard array that every n -tuple appears at least once. Now, suppose that an n -tuple appears in both the l th row and the m th row with $l < m$. Then this n -tuple must be equal to $\mathbf{e}_l + \mathbf{v}_i$ for some i and equal to $\mathbf{e}_m + \mathbf{v}_j$ for some j . As a result, $\mathbf{e}_l + \mathbf{v}_i = \mathbf{e}_m + \mathbf{v}_j$. From this equality we obtain $\mathbf{e}_m = \mathbf{e}_l + (\mathbf{v}_i - \mathbf{v}_j)$. Because \mathbf{v}_i and \mathbf{v}_j are codewords in C , $\mathbf{v}_i - \mathbf{v}_j$ is also a codeword in C , say \mathbf{v}_s . Then $\mathbf{e}_m = \mathbf{e}_l + \mathbf{v}_s$. This implies that the n -tuple \mathbf{e}_m is in the l th row of the array, which contradicts the construction rule of the array that \mathbf{e}_m , the first element of the m th row, should be unused in any previous row. Therefore, no n -tuple can appear in more than one row of the array. This concludes the proof of the second part of the theorem. **Q.E.D.**

From Theorem 3.3 we see that there are $2^n/2^k = 2^{n-k}$ disjoint rows in the standard array, and that each row consists of 2^k distinct elements. The 2^{n-k} rows are called the *cosets* of the code C , and the first n -tuple \mathbf{e}_j of each coset is called a *coset leader* (or coset representative). The coset concept for a subgroup was presented in Section 2.1. Any element in a coset can be used as its coset leader. This does not change the elements of the coset; it simply permutes them.

EXAMPLE 3.6

Consider the $(6, 3)$ linear code generated by the following matrix:

$$\mathbf{G} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The standard array of this code is shown in Figure 3.7.

A standard array of an (n, k) linear code C consists of 2^k disjoint columns. Each column consists of 2^{n-k} n -tuples, with the topmost one as a codeword in C . Let D_j denote the j th column of the standard array. Then,

$$D_j = \{\mathbf{v}_j, \mathbf{e}_2 + \mathbf{v}_j, \mathbf{e}_3 + \mathbf{v}_j, \dots, \mathbf{e}_{2^{n-k}} + \mathbf{v}_j\}, \quad (3.27)$$

Coset leader	011100	101010	110001	110110	101101	011011	000111
000000	111100	001010	010001	010110	001101	111011	100111
100000	001100	111010	100001	100110	111101	001011	010111
010000	010100	100010	111001	111110	100101	010011	001111
001000	011000	101110	110101	110010	101001	011111	000011
000100	011110	101000	110011	110100	101111	011001	000101
000010	011101	101011	110000	110111	101100	011010	000110
000001	011011	101110	010101	010010	001001	111111	100011
100100	111000	001110	010101	010010	001001		

FIGURE 3.7: Standard array for a $(6, 3)$ code.

where \mathbf{v}_j is a codeword of C , and $\mathbf{e}_2, \mathbf{e}_3, \dots, \mathbf{e}_{2^{n-k}}$ are the coset leaders. The 2^k disjoint columns D_1, D_2, \dots, D_{2^k} can be used for decoding the code C as described earlier in this section. Suppose that the codeword \mathbf{v}_j is transmitted over a noisy channel. From (3.27) we see that the received vector \mathbf{r} is in D_j if the error pattern caused by the channel is a coset leader. In this event, the received vector \mathbf{r} will be decoded correctly into the transmitted codeword \mathbf{v}_j ; however, if the error pattern caused by the channel is not a coset leader, an erroneous decoding will result. This can be seen as follows. The error pattern \mathbf{x} caused by the channel must be in some coset and under some nonzero codeword, say in the l th coset and under the codeword $\mathbf{v}_i \neq \mathbf{0}$. Then, $\mathbf{x} = \mathbf{e}_l + \mathbf{v}_i$, and the received vector is

$$\mathbf{r} = \mathbf{v}_j + \mathbf{x} = \mathbf{e}_l + (\mathbf{v}_i + \mathbf{v}_j) = \mathbf{e}_l + \mathbf{v}_s.$$

The received vector \mathbf{r} is thus in D_s and is decoded into \mathbf{v}_s , which is not the transmitted codeword. This results in an erroneous decoding. Therefore, the decoding is correct if and only if the error pattern caused by the channel is a coset leader. For this reason, the 2^{n-k} coset leaders (including the zero vector $\mathbf{0}$) are called the *correctable error patterns*. Summarizing the preceding results, we have the following theorem:

THEOREM 3.4 Every (n, k) linear block code is capable of correcting 2^{n-k} error patterns.

To minimize the probability of a decoding error, the error patterns that are most likely to occur for a given channel should be chosen as the coset leaders. For a BSC, an error pattern of smaller weight is more probable than an error pattern of larger weight. Therefore, when a standard array is formed, each coset leader should be chosen to be a vector of *least weight* from the remaining available vectors. If coset leaders are chosen in this manner, each coset leader has minimum weight in its coset. As a result, the decoding based on the standard array is the minimum distance decoding (i.e., the maximum likelihood decoding). To see this, let \mathbf{r} be the received vector. Suppose that \mathbf{r} is found in the i th column D_i and l th coset of the standard array. Then, \mathbf{r} is decoded into the codeword \mathbf{v}_i . Because $\mathbf{r} = \mathbf{e}_l + \mathbf{v}_i$, the distance between \mathbf{r} and \mathbf{v}_i is

$$d(\mathbf{r}, \mathbf{v}_i) = w(\mathbf{r} + \mathbf{v}_i) = w(\mathbf{e}_l + \mathbf{v}_i + \mathbf{v}_i) = w(\mathbf{e}_l). \quad (3.28)$$

Now, consider the distance between \mathbf{r} and any other codeword, say \mathbf{v}_j ,

$$d(\mathbf{r}, \mathbf{v}_j) = w(\mathbf{r} + \mathbf{v}_j) = w(\mathbf{e}_l + \mathbf{v}_i + \mathbf{v}_j).$$

Because \mathbf{v}_i and \mathbf{v}_j are two different codewords, their vector sum, $\mathbf{v}_i + \mathbf{v}_j$, is a nonzero codeword, say \mathbf{v}_s . Thus,

$$d(\mathbf{r}, \mathbf{v}_j) = w(\mathbf{e}_l + \mathbf{v}_s). \quad (3.29)$$

Because \mathbf{e}_l and $\mathbf{e}_l + \mathbf{v}_s$ are in the same coset and since $w(\mathbf{e}_l) \leq w(\mathbf{e}_l + \mathbf{v}_s)$, it follows from (3.28) and (3.29) that

$$d(\mathbf{r}, \mathbf{v}_i) \leq d(\mathbf{r}, \mathbf{v}_j).$$

This says that the received vector is decoded into a closest codeword. Hence, if each coset leader is chosen to have minimum weight in its coset, the decoding based on the standard array is the minimum distance decoding, or MLD.

Let α_i denote the number of coset leaders of weight i . The numbers $\alpha_0, \alpha_1, \dots, \alpha_n$ are called the *weight distribution* of the coset leaders. Knowing these numbers, we can compute the probability of a decoding error. Because a decoding error occurs if and only if the error pattern is not a coset leader, the error probability for a BSC with transition probability p is

$$P(E) = 1 - \sum_{i=0}^n \alpha_i p^i (1-p)^{n-i}. \quad (3.30)$$

EXAMPLE 3.7

Consider the $(6, 3)$ code given in Example 3.6. The standard array for this code is shown in Figure 3.7. The weight distribution of the coset leaders is $\alpha_0 = 1, \alpha_1 = 6, \alpha_2 = 1$, and $\alpha_3 = \alpha_4 = \alpha_5 = \alpha_6 = 0$. Thus,

$$P(E) = 1 - (1-p)^6 - 6p(1-p)^5 - p^2(1-p)^4.$$

For $p = 10^{-2}$, we have $P(E) \approx 1.37 \times 10^{-3}$.

An (n, k) linear code is capable of detecting $2^n - 2^k$ error patterns; however, it is capable of correcting only 2^{n-k} error patterns. For large n , 2^{n-k} is a small fraction of $2^n - 2^k$. Therefore, the probability of a decoding error is much higher than the probability of an undetected error.

THEOREM 3.5 For an (n, k) linear code C with minimum distance d_{\min} , all the n -tuples of weight $t = \lfloor (d_{\min} - 1)/2 \rfloor$ or less can be used as coset leaders of a standard array of C . If all the n -tuples of weight t or less are used as coset leaders, there is at least one n -tuple of weight $t + 1$ that cannot be used as a coset leader.

Proof. Because the minimum distance of C is d_{\min} , the minimum weight of C is also d_{\min} . Let \mathbf{x} and \mathbf{y} be two n -tuples of weight t or less. Clearly, the weight of $\mathbf{x} + \mathbf{y}$ is

$$w(\mathbf{x} + \mathbf{y}) \leq w(\mathbf{x}) + w(\mathbf{y}) \leq 2t < d_{\min}.$$

Suppose that \mathbf{x} and \mathbf{y} are in the same coset; then, $\mathbf{x} + \mathbf{y}$ must be a nonzero codeword in C . This is impossible, because the weight of $\mathbf{x} + \mathbf{y}$ is less than the minimum weight of C . Therefore, no two n -tuples of weight t or less can be in the same coset of C , and all the n -tuples of weight t or less can be used as coset leaders.

Let \mathbf{v} be a minimum weight codeword of C [i.e., $w(\mathbf{v}) = d_{\min}$]. Let \mathbf{x} and \mathbf{y} be two n -tuples that satisfy the following two conditions:

- i. $\mathbf{x} + \mathbf{y} = \mathbf{v}$.
- ii. \mathbf{x} and \mathbf{y} do not have nonzero components in common places.

It follows from the definition that \mathbf{x} and \mathbf{y} must be in the same coset and

$$w(\mathbf{x}) + w(\mathbf{y}) = w(\mathbf{v}) = d_{\min}.$$

Suppose we choose \mathbf{y} such that $w(\mathbf{y}) = t + 1$. Because $2t + 1 \leq d_{\min} \leq 2t + 2$, we have $w(\mathbf{x}) = t$ or $t + 1$. If \mathbf{x} is used as a coset leader, then \mathbf{y} cannot be a coset leader. **Q.E.D.**

Theorem 3.5 reconfirms the fact that an (n, k) linear code with minimum distance d_{\min} is capable of correcting all the error patterns of $\lfloor(d_{\min} - 1)/2\rfloor$ or fewer errors, but it is not capable of correcting all the error patterns of weight $t + 1$.

A standard array has an important property that can be used to simplify the decoding process. Let \mathbf{H} be the parity-check matrix of the given (n, k) linear code C .

THEOREM 3.6 All the 2^k n -tuples of a coset have the same syndrome. The syndromes for different cosets are different.

Proof. Consider the coset whose coset leader is \mathbf{e}_l . A vector in this coset is the sum of \mathbf{e}_l and some codeword \mathbf{v}_i in C . The syndrome of this vector is

$$(\mathbf{e}_l + \mathbf{v}_i)\mathbf{H}^T = \mathbf{e}_l\mathbf{H}^T + \mathbf{v}_i\mathbf{H}^T = \mathbf{e}_l\mathbf{H}^T$$

(since $\mathbf{v}_i\mathbf{H}^T = \mathbf{0}$). The preceding equality says that the syndrome of any vector in a coset is equal to the syndrome of the coset leader. Therefore, all the vectors of a coset have the same syndrome.

Let \mathbf{e}_j and \mathbf{e}_l be the coset leaders of the j th and l th cosets, respectively, where $j < l$. Suppose that the syndromes of these two cosets are equal. Then,

$$\begin{aligned}\mathbf{e}_j\mathbf{H}^T &= \mathbf{e}_l\mathbf{H}^T, \\ (\mathbf{e}_j + \mathbf{e}_l)\mathbf{H}^T &= \mathbf{0}.\end{aligned}$$

This implies that $\mathbf{e}_j + \mathbf{e}_l$ is a codeword in C , say \mathbf{v}_i . Thus, $\mathbf{e}_j + \mathbf{e}_l = \mathbf{v}_i$, and $\mathbf{e}_l = \mathbf{e}_j + \mathbf{v}_i$. This implies that \mathbf{e}_l is in the j th coset, which contradicts the construction rule of a standard array that a coset leader should be previously unused. Therefore, no two cosets have the same syndrome. **Q.E.D.**

We recall that the syndrome of an n -tuple is an $(n - k)$ -tuple, and there are 2^{n-k} distinct $(n - k)$ -tuples. It follows from Theorem 3.6 that there is a one-to-one correspondence between a coset and an $(n - k)$ -tuple syndrome; or there is a one-to-one correspondence between a coset leader (a correctable error pattern) and a syndrome. Using this one-to-one correspondence relationship, we can form a decoding table, which is much simpler to use than a standard array. The table consists of 2^{n-k} coset leaders (the correctable error patterns) and their corresponding syndromes. This table is either stored or wired in the receiver. The decoding of a received vector consists of three steps:

1. Compute the syndrome of \mathbf{r} , $\mathbf{r} \cdot \mathbf{H}^T$.
2. Locate the coset leader \mathbf{e}_l whose syndrome is equal to $\mathbf{r} \cdot \mathbf{H}^T$. Then \mathbf{e}_l is assumed to be the error pattern caused by the channel.
3. Decode the received vector \mathbf{r} into the codeword $\mathbf{v}^* = \mathbf{r} + \mathbf{e}_l$.

The described decoding scheme is called the *syndrome decoding* or *table-lookup decoding*. In principle, table-lookup decoding can be applied to any (n, k) linear code. It results in minimum decoding delay and minimum error probability;

however, for large $n - k$, the implementation of this decoding scheme becomes impractical, and either a large storage or a complicated logic circuitry is needed. Several practical decoding schemes that are variations of table-lookup decoding are discussed in subsequent chapters. Each of these decoding schemes requires additional properties of a code other than the linear structure.

EXAMPLE 3.8

Consider the $(7, 4)$ linear code given in Table 3.1. The parity-check matrix, as given in Example 3.3, is

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

The code has $2^3 = 8$ cosets, and therefore there are eight correctable error patterns (including the all-zero vector). Because the minimum distance of the code is 3, it is capable of correcting all the error patterns of weight 1 or 0. Hence, all the 7-tuples of weight 1 or 0 can be used as coset leaders. There are $\binom{7}{0} + \binom{7}{1} = 8$ such vectors. We see that for the $(7, 4)$ linear code considered in this example, the number of correctable error patterns guaranteed by the minimum distance is equal to the total number of correctable error patterns. The correctable error patterns and their corresponding syndromes are given in Table 3.2.

Suppose that the codeword $\mathbf{v} = (1001011)$ is transmitted, and $\mathbf{r} = (1001111)$ is received. For decoding \mathbf{r} , we compute the syndrome of \mathbf{r} :

$$\mathbf{s} = (1001111) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = (011).$$

TABLE 3.2: Decoding table for the $(7, 4)$ linear code given in Table 3.1.

Syndrome	Coset leaders
(1 0 0)	(1 0 0 0 0 0 0)
(0 1 0)	(0 1 0 0 0 0 0)
(0 0 1)	(0 0 1 0 0 0 0)
(1 1 0)	(0 0 0 1 0 0 0)
(0 1 1)	(0 0 0 0 1 0 0)
(1 1 1)	(0 0 0 0 0 1 0)
(1 0 1)	(0 0 0 0 0 0 1)

From Table 3.2 we find that $(0 \ 1 \ 1)$ is the syndrome of the coset leader $\mathbf{e} = (0000100)$. Thus, (0000100) is assumed to be the error pattern caused by the channel, and \mathbf{r} is decoded into

$$\begin{aligned}\mathbf{v}^* &= \mathbf{r} + \mathbf{e} \\ &= (1\ 0\ 0\ 1\ 1\ 1\ 1) + (0\ 0\ 0\ 0\ 1\ 0\ 0) \\ &= (1\ 0\ 0\ 1\ 0\ 1\ 1),\end{aligned}$$

which is the codeword transmitted. The decoding is correct, since the error pattern caused by the channel is a coset leader.

Now, suppose that $\mathbf{v} = (0000000)$ is transmitted, and $\mathbf{r} = (1000100)$ is received. We see that two errors have occurred during the transmission of \mathbf{v} . The error pattern is not correctable and will cause a decoding error. When \mathbf{r} is received, the receiver computes the syndrome:

$$\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T = (1\ 1\ 1).$$

From the decoding table we find that the coset leader $\mathbf{e} = (0000010)$ corresponds to the syndrome $\mathbf{s} = (1\ 1\ 1)$. As a result, \mathbf{r} is decoded into the codeword

$$\begin{aligned}\mathbf{v}^* &= \mathbf{r} + \mathbf{e} \\ &= (1\ 0\ 0\ 0\ 1\ 0\ 0) + (0\ 0\ 0\ 0\ 0\ 1\ 0) \\ &= (1\ 0\ 0\ 0\ 1\ 1\ 0).\end{aligned}$$

Because \mathbf{v}^* is not the codeword transmitted, a decoding error is committed.

Using Table 3.2, we see that the code is capable of correcting any single error over a block of seven digits. When two or more errors occur, a decoding error will be committed.

The table-lookup decoding of an (n, k) linear code may be implemented as follows. The decoding table is regarded as the truth table of n switching functions:

$$\begin{aligned}e_0 &= f_0(s_0, s_1, \dots, s_{n-k-1}), \\ e_1 &= f_1(s_0, s_1, \dots, s_{n-k-1}), \\ &\vdots \\ e_{n-1} &= f_{n-1}(s_0, s_1, \dots, s_{n-k-1}),\end{aligned}$$

where $s_0, s_1, \dots, s_{n-k-1}$ are the syndrome digits, which are regarded as switching variables, and e_0, e_1, \dots, e_{n-1} are the estimated error digits. When these n switching functions are derived and simplified, a combinational logic circuit with the $n - k$ syndrome digits as inputs and the estimated error digits as outputs can be realized. The implementation of the syndrome circuit was discussed in Section 3.2. The general decoder for an (n, k) linear code based on the table-lookup scheme is shown

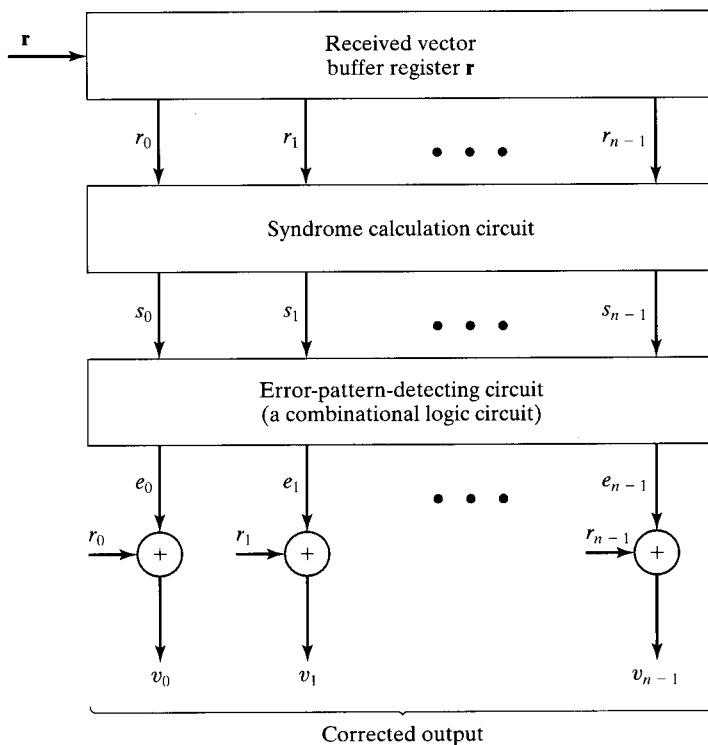


FIGURE 3.8: General decoder for a linear block code.

in Figure 3.8. The cost of this decoder depends primarily on the complexity of the combinational logic circuit.

EXAMPLE 3.9

Again, we consider the (7, 4) code given in Table 3.1. The syndrome circuit for this code is shown in Figure 3.5. The decoding table is given by Table 3.2. From this table we form the truth table (Table 3.3). The switching expressions for the seven error digits are

$$\begin{aligned}
 e_0 &= s_0 \Lambda s'_1 \Lambda s'_2, & e_1 &= s'_0 \Lambda s_1 \Lambda s'_2, \\
 e_2 &= s'_0 \Lambda s'_1 \Lambda s_2, & e_3 &= s_0 \Lambda s_1 \Lambda s'_2, \\
 e_4 &= s'_0 \Lambda s_1 \Lambda s_2, & e_5 &= s_0 \Lambda s_1 \Lambda s_2, \\
 e_6 &= s_0 \Lambda s'_1 \Lambda s_2,
 \end{aligned}$$

where Λ denotes the logic-AND operation and s' denotes the logic-COMPLEMENT of s . These seven switching expressions can be realized by seven 3-input AND gates. The complete circuit of the decoder is shown in Figure 3.9.

TABLE 3.3: Truth table for the error digits of the correctable error patterns of the $(7, 4)$ linear code given in Table 3.1.

Syndromes			Correctable error patterns (coset leaders)						
s_0	s_1	s_2	e_0	e_1	e_2	e_3	e_4	e_5	e_6
0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0
0	0	1	0	0	1	0	0	0	0
1	1	0	0	0	0	1	0	0	0
0	1	1	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1

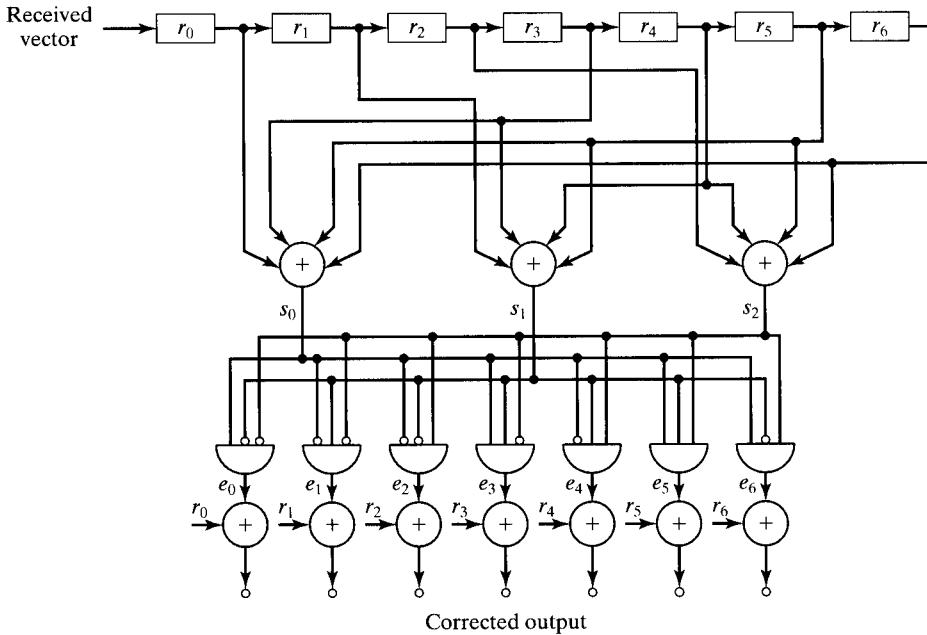


FIGURE 3.9: Decoding circuit for the $(7, 4)$ code given in Table 3.1.

3.6 PROBABILITY OF AN UNDETECTED ERROR FOR LINEAR CODES OVER A BSC

If an (n, k) linear code is used only for error detection over a BSC, the probability of an undetected error $P_u(E)$ can be computed from (3.19) if the weight distribution of the code is known. There exists an interesting relationship between the weight distribution of a linear code and the weight distribution of its dual code. This relationship often makes the computation of $P_u(E)$ much easier. Let $\{A_0, A_1, \dots, A_n\}$

be the weight distribution of an (n, k) linear code C , and let $\{B_0, B_1, \dots, B_n\}$ be the weight distribution of its dual code C_d . Now, we represent these two weight distributions in polynomial form as follows:

$$\begin{aligned} A(z) &= A_0 + A_1 z + \cdots + A_n z^n, \\ B(z) &= B_0 + B_1 z + \cdots + B_n z^n. \end{aligned} \quad (3.31)$$

Then, $A(z)$ and $B(z)$ are related by the following identity:

$$A(z) = 2^{-(n-k)}(1+z)^n B\left(\frac{1-z}{1+z}\right). \quad (3.32)$$

This identity is known as the *MacWilliams identity* [13]. The polynomials $A(z)$ and $B(z)$ are called the *weight enumerators* for the (n, k) linear code C and its dual C_d . From the MacWilliams identity, we see that if the weight distribution of the dual of a linear code is known, the weight distribution of the code itself can be determined. As a result, this gives us more flexibility in computing the weight distribution of a linear code.

Using the MacWilliams identity, we can compute the probability of an undetected error for an (n, k) linear code from the weight distribution of its dual. First, we put the expression of (3.19) into the following form:

$$\begin{aligned} P_u(E) &= \sum_{i=1}^n A_i p^i (1-p)^{n-i} \\ &= (1-p)^n \sum_{i=1}^n A_i \left(\frac{p}{1-p}\right)^i. \end{aligned} \quad (3.33)$$

Substituting $z = p/(1-p)$ in $A(z)$ of (3.31) and using the fact that $A_0 = 1$, we obtain the following identity:

$$A\left(\frac{p}{1-p}\right) - 1 = \sum_{i=1}^n A_i \left(\frac{p}{1-p}\right)^i. \quad (3.34)$$

Combining (3.33) and (3.34), we have the following expression for the probability of an undetected error:

$$P_u(E) = (1-p)^n \left[A\left(\frac{p}{1-p}\right) - 1 \right]. \quad (3.35)$$

From (3.35) and the MacWilliams identity of (3.32), we finally obtain the following expression for $P_u(E)$:

$$P_u(E) = 2^{-(n-k)} B(1-2p) - (1-p)^n, \quad (3.36)$$

where

$$B(1-2p) = \sum_{i=0}^n B_i (1-2p)^i.$$

Hence, there are two ways for computing the probability of an undetected error for a linear code; often, one is easier than the other. If $n - k$ is smaller than k , it is much easier to compute $P_u(E)$ from (3.36); otherwise, it is easier to use (3.35).

EXAMPLE 3.10

Consider the $(7, 4)$ linear code given in Table 3.1. The dual of this code is generated by its parity-check matrix,

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

(see Example 3.3). Taking the linear combinations of the rows of \mathbf{H} , we obtain the following eight vectors in the dual code:

$$(0\ 0\ 0\ 0\ 0\ 0\ 0), \quad (1\ 1\ 0\ 0\ 1\ 0\ 1),$$

$$(1\ 0\ 0\ 1\ 0\ 1\ 1), \quad (1\ 0\ 1\ 1\ 1\ 0\ 0),$$

$$(0\ 1\ 0\ 1\ 1\ 1\ 0), \quad (0\ 1\ 1\ 1\ 0\ 0\ 1),$$

$$(0\ 0\ 1\ 0\ 1\ 1\ 1), \quad (1\ 1\ 1\ 0\ 0\ 1\ 0).$$

Thus, the weight enumerator for the dual code is $B(z) = 1 + 7z^4$. Using (3.36), we obtain the probability of an undetected error for the $(7, 4)$ linear code given in Table 3.1:

$$P_u(E) = 2^{-3}[1 + 7(1 - 2p)^4] - (1 - p)^7.$$

This probability was also computed in Section 3.4 using the weight distribution of the code itself.

Theoretically, we can compute the weight distribution of an (n, k) linear code by examining its 2^k codewords or by examining the 2^{n-k} codewords of its dual and then applying the MacWilliams identity; however, for large n , k , and $n - k$, the computation becomes practically impossible. Except for some short linear codes and a few small classes of linear codes, the weight distributions for many known linear codes are still unknown. Consequently, it is very difficult, if not impossible, to compute their probability of an undetected error.

Although it is difficult to compute the probability of an undetected error for a specific (n, k) linear code for large n and k , it is quite easy to derive an upper bound on the average probability of an undetected error for the ensemble of all (n, k) linear systematic codes. As we showed earlier, an (n, k) linear systematic code is completely specified by a matrix \mathbf{G} of the form given by (3.4). The submatrix \mathbf{P} consists of $k(n - k)$ entries. Because each entry p_{ij} can be either a 0 or a 1, there are $2^{k(n-k)}$ distinct matrices \mathbf{G} 's of the form given by (3.4). Let Γ denote the ensemble of codes generated by these $2^{k(n-k)}$ matrices. Suppose that we choose a code randomly from Γ and use it for error detection. Let C_j be the chosen code. Then, the probability that C_j will be chosen is

$$P(C_j) = 2^{-k(n-k)}. \quad (3.37)$$

Let A_{ji} denote the number of codewords in C_j with weight i . It follows from (3.19) that the probability of an undetected error for C_j is given by

$$P_u(E|C_j) = \sum_{i=1}^n A_{ji} p^i (1-p)^{n-i}. \quad (3.38)$$

The average probability of an undetected error for a linear code in Γ is defined as

$$\mathbf{P}_u(\mathbf{E}) = \sum_{j=1}^{|\Gamma|} P(C_j) P_u(E|C_j), \quad (3.39)$$

where $|\Gamma|$ denotes the number of codes in Γ . Substituting (3.37) and (3.38) into (3.39), we obtain

$$\mathbf{P}_u(\mathbf{E}) = 2^{-k(n-k)} \sum_{i=1}^n p^i (1-p)^{n-i} \sum_{j=1}^{|\Gamma|} A_{ji}. \quad (3.40)$$

A nonzero n -tuple is contained in either exactly $2^{(k-1)(n-k)}$ codes in Γ or in none of the codes (left as a problem). Because there are $\binom{n}{i}$ n -tuples of weight i , we have

$$\sum_{j=1}^{|\Gamma|} A_{ji} \leq \binom{n}{i} 2^{(k-1)(n-k)}. \quad (3.41)$$

Substituting (3.41) into (3.40), we obtain the following upper bound on the average probability of an undetected error for an (n, k) linear systematic code:

$$\begin{aligned} \mathbf{P}_u(\mathbf{E}) &\leq 2^{-(n-k)} \sum_{i=1}^n \binom{n}{i} p^i (1-p)^{n-i} \\ &= 2^{-(n-k)} [1 - (1-p)^n]. \end{aligned} \quad (3.42)$$

Because $[1 - (1-p)^n] \leq 1$, it is clear that $\mathbf{P}_u(\mathbf{E}) \leq 2^{-(n-k)}$.

The preceding result says that there exist (n, k) linear codes with the probability of an undetected error, $P_u(E)$, upper bounded by $2^{-(n-k)}$. In other words, there exist (n, k) linear codes with $P_u(E)$ decreasing exponentially with the number of parity-check digits, $n - k$. Even for moderate $n - k$, these codes have a very small probability of an undetected error. For example, let $n - k = 30$. There exist (n, k) linear codes for which $P_u(E)$ is upper bounded by $2^{-30} \approx 10^{-9}$. Many classes of linear codes have been constructed for the past five decades; however, only a few small classes of linear codes have been proved to have a $P_u(E)$ that satisfies the upper bound $2^{-(n-k)}$. It is still not known whether the other known linear codes satisfy this upper bound.

Good codes for error detection and their applications in error control will be presented in later chapters. An excellent treatment of error-detecting codes can be found in [12].

3.7 SINGLE-PARITY-CHECK CODES, REPETITION CODES, AND SELF-DUAL CODES

A single-parity-check (SPC) code is a linear block code with a single parity-check digit. Let $\mathbf{u} = (u_0, u_1, \dots, u_{k-1})$ be the message to be encoded. The single parity-check digit is given by

$$p = u_0 + u_1 + \dots + u_{k-1} \quad (3.43)$$

which is simply the modulo-2 sum of all the message digits. Adding this parity-check digit to each k -digit message results in a $(k+1, k)$ linear block code. Each codeword is of the form

$$\mathbf{v} = (p, u_0, u_1, \dots, u_{k-1}).$$

From (3.43), we readily see that $p = 1$ if the weight of message \mathbf{u} is odd, and $p = 0$ if the weight of message \mathbf{u} is even. Therefore, all the codewords of a SPC code have even weights, and the minimum weight (or minimum distance) of the code is 2. The generator of the code in systematic form is given by

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 0 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 0 & 0 & 1 & 0 & \cdots & 0 \\ \vdots & & \vdots & & & & \\ 1 & 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \mathbf{I}_k \quad (3.44)$$

From (3.44) we find that the parity-check matrix of the code is

$$\mathbf{H} = [1 \ 1 \ \cdots \ 1]. \quad (3.45)$$

Because all the codewords have even weights, a SPC code is also called an even-parity-check code. SPC codes are often used for simple error detection. Any error pattern with an odd number of errors will change a codeword into a received vector of odd weight that is not a codeword. Hence, the syndrome of the received vector is not equal to zero. Consequently, all the error patterns of odd weight are detectable.

A repetition code of length n is an $(n, 1)$ linear block code that consists of only two codewords, the all-zero codeword $(0 \ 0 \ \cdots \ 0)$ and the all-one codeword $(1 \ 1 \ \cdots \ 1)$. This code is obtained by simply repeating a single message bit n times. The generator matrix of the code is

$$\mathbf{G} = [1 \ 1 \ \cdots \ 1]. \quad (3.46)$$

From (3.44) through (3.46), we readily see that the $(n, 1)$ repetition code and the $(n, n-1)$ SPC code are dual codes to each other.

SPC and repetition codes are often used as component codes for constructing long, powerful codes as will be seen in later chapters.

A linear block code C that is equal to its dual code C_d is called a *self-dual code*. For a self-dual code, the code length n must be even, and the dimension k of the code must be equal to $n/2$. Therefore, its rate R is equal to $\frac{1}{2}$. Let \mathbf{G} be a generator matrix of a self-dual code C . Then, \mathbf{G} is also a generator matrix of its dual code C_d and hence is a parity-check matrix of C . Consequently,

$$\mathbf{G} \cdot \mathbf{G}^T = \mathbf{0} \quad (3.47)$$

Suppose \mathbf{G} is in systematic form, $\mathbf{G} = [\mathbf{P} \ \mathbf{I}_{n/2}]$. From (3.47), we can easily see that

$$\mathbf{P} \cdot \mathbf{P}^T = \mathbf{I}_{n/2}. \quad (3.48)$$

Conversely, if a rate- $\frac{1}{2}$ ($n, n/2$) linear block code C satisfies the condition of (3.47) [or (3.48)], then it is a self-dual code (the proof is left as a problem).

EXAMPLE 3.11

Consider the (8, 4) linear block code generated by the matrix

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

The code has a rate $R = \frac{1}{2}$. It is easy to check that $\mathbf{G} \cdot \mathbf{G}^T = \mathbf{0}$. Therefore, it is a self-dual code.

There are many good self-dual codes but the most well known self-dual code is the (24, 12) Golay code, which will be discussed in Chapter 4.

PROBLEMS

3.1 Consider a systematic (8, 4) code whose parity-check equations are

$$v_0 = u_1 + u_2 + u_3,$$

$$v_1 = u_0 + u_1 + u_2,$$

$$v_2 = u_0 + u_1 + u_3,$$

$$v_3 = u_0 + u_2 + u_3.$$

where u_0, u_1, u_2 , and u_3 , are message digits, and v_0, v_1, v_2 , and v_3 are parity-check digits. Find the generator and parity-check matrices for this code. Show analytically that the minimum distance of this code is 4.

3.2 Construct an encoder for the code given in Problem 3.1.

3.3 Construct a syndrome circuit for the code given in Problem 3.1.

3.4 Let \mathbf{H} be the parity-check matrix of an (n, k) linear code C that has both odd- and even-weight codewords. Construct a new linear code C_1 with the following parity-check matrix:

$$\mathbf{H}_1 = \left[\begin{array}{c|ccccc} 0 & & & & & \\ 0 & & & & & \\ \vdots & & \mathbf{H} & & & \\ 0 & & & & & \\ \hline 1 & 1 & 1 & \cdots & 1 \end{array} \right].$$

(Note that the last row of \mathbf{H}_1 consists of all 1's.)

- a. Show that C_1 is an $(n+1, k)$ linear code. C_1 is called an *extension* of C .
- b. Show that every codeword of C_1 has even weight.

- c. Show that C_1 can be obtained from C by adding an extra parity-check digit, denoted by v_∞ , to the left of each codeword \mathbf{v} as follows: (1) if \mathbf{v} has odd weight, then $v_\infty = 1$, and (2) if \mathbf{v} has even weight, then $v_\infty = 0$. The parity-check digit v_∞ is called an *overall parity-check* digit.
- 3.5** Let C be a linear code with both even- and odd-weight codewords. Show that the number of even-weight codewords is equal to the number of odd-weight codewords.
- 3.6** Consider an (n, k) linear code C whose generator matrix \mathbf{G} contains no zero column. Arrange all the codewords of C as rows of a 2^k -by- n array.
- Show that no column of the array contains only zeros.
 - Show that each column of the array consists of 2^{k-1} zeros and 2^{k-1} ones.
 - Show that the set of all codewords with zeros in a particular component position forms a subspace of C . What is the dimension of this subspace?
- 3.7** Prove that the Hamming distance satisfies the triangle inequality; that is, let \mathbf{x}, \mathbf{y} , and \mathbf{z} be three n -tuples over $GF(2)$, and show that
- $$d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) \geq d(\mathbf{x}, \mathbf{z}).$$
- 3.8** Prove that a linear code is capable of correcting λ or fewer errors and simultaneously detecting l ($l > \lambda$) or fewer errors if its minimum distance $d_{\min} \geq \lambda + l + 1$.
- 3.9** Determine the weight distribution of the $(8, 4)$ linear code given in Problem 3.1. Let the transition probability of a BSC be $p = 10^{-2}$. Compute the probability of an undetected error of this code.
- 3.10** Because the $(8, 4)$ linear code given in Problem 3.1 has minimum distance 4, it is capable of correcting all the single-error patterns and simultaneously detecting any combination of double errors. Construct a decoder for this code. The decoder must be capable of correcting any single error and detecting any double errors.
- 3.11** Let Γ be the ensemble of all the binary systematic (n, k) linear codes. Prove that a nonzero binary n -tuple \mathbf{v} is contained in either exactly $2^{(k-1)(n-k)}$ codes in Γ or in none of the codes in Γ .
- 3.12** The $(8, 4)$ linear code given in Problem 3.1 is capable of correcting 16 error patterns (the coset leaders of a standard array). Suppose that this code is used for a BSC. Devise a decoder for this code based on the table-lookup decoding scheme. The decoder is designed to correct the 16 most probable error patterns.
- 3.13** Let C_1 be an (n_1, k) linear systematic code with minimum distance d_1 and generator matrix $\mathbf{G}_1 = [\mathbf{P}_1 \ \mathbf{I}_k]$. Let C_2 be an (n_2, k) linear systematic code with minimum distance d_2 and generator matrix $\mathbf{G}_2 = [\mathbf{P}_2 \ \mathbf{I}_k]$. Consider an $(n_1 + n_2, k)$ linear code with the following parity-check matrix:

$$\mathbf{H} = \left[\begin{array}{c|c} & \mathbf{P}_1^T \\ \mathbf{I}_{n_1+n_2-k} & \mathbf{I}_k \\ & \mathbf{P}_2^T \end{array} \right].$$

Show that this code has a minimum distance of at least $d_1 + d_2$.

- 3.14** Show that the $(8, 4)$ linear code C given in Problem 3.1 is self-dual.
- 3.15** For any binary (n, k) linear code with minimum distance (or minimum weight) $2t + 1$ or greater, show that the number of parity-check digits satisfies the following inequality:

$$n - k \geq \log_2 \left[1 + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{t} \right].$$

The preceding inequality gives an upper bound on the random-error-correcting capability t of an (n, k) linear code. This bound is known as the *Hamming*

bound [14]. (*Hint:* For an (n, k) linear code with minimum distance $2t + 1$ or greater, all the n -tuples of weight t or less can be used as coset leaders in a standard array.)

- 3.16** Show that the minimum distance d_{\min} of an (n, k) linear code satisfies the following inequality:

$$d_{\min} \leq \frac{n \cdot 2^{k-1}}{2^k - 1}.$$

(*Hint:* Use the result of Problem 3.6(b). This bound is known as the *Plotkin bound* [1-3].)

- 3.17** Show that there exists an (n, k) linear code with a minimum distance of at least d if

$$\sum_{i=1}^{d-1} \binom{n}{i} < 2^{n-k}.$$

(*Hint:* Use the result of Problem 3.11 and the fact that the nonzero n -tuples of weight $d - 1$ or less can be at most in

$$\left\{ \sum_{i=1}^{d-1} \binom{n}{i} \right\} \cdot 2^{(k-1)(n-k)}$$

(n, k) systematic linear codes.)

- 3.18** Show that there exists an (n, k) linear code with a minimum distance of at least d_{\min} that satisfies the following inequality:

$$\sum_{i=1}^{d_{\min}-1} \binom{n}{i} < 2^{n-k} \leq \sum_{i=1}^{d_{\min}} \binom{n}{i}.$$

(*Hint:* See Problem 3.17. The second inequality provides a lower bound on the minimum distance attainable with an (n, k) linear code. This bound is known as the *Varshamov–Gilbert bound* [1-3].)

- 3.19** Consider a rate- $\frac{1}{2}$ $(n, n/2)$ linear block code C with a generator matrix \mathbf{G} . Prove that C is self-dual if $\mathbf{G} \cdot \mathbf{G}^T = \mathbf{0}$.
- 3.20** Devise an encoder for the $(n, n - 1)$ SPC code with only one memory element (or flip-flop) and one X-OR gate (or modulo-2 adder).

BIBLIOGRAPHY

1. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968; Rev. ed., Aegean Park Press, Laguna Hills, N.Y., 1984.
2. W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2d ed., MIT Press, Cambridge, 1972.
3. F. J. MacWilliams and J. J. A. Sloane, *The Theory of Error-Correcting Codes*, North Holland, Amsterdam, 1977.
4. R. J. McEliece, *The Theory of Information and Coding*, Addison-Wesley, Reading, Mass., 1977.

5. G. Clark and J. Cain, *Error-Correcting Codes for Digital Communications*, Plenum Press, New York, 1981.
6. R. E. Blahut, *Algebraic Codes for Data Transmission*, Cambridge University Press, Cambridge, UK, 2003.
7. A. M. Michelson and A. H. Levesque, *Error Control Techniques for Digital Communication*, John Wiley & Sons, New York, 1985.
8. T. R. N. Rao and E. Fujiwara, *Error-Correcting Codes for Computer Systems*, Prentice Hall, Englewood Cliffs, N.J., 1989.
9. S. A. Vanstone and P. C. van Oorschot, *An Introduction to Error Correcting Codes with Applications*, Kluwer Academic, Boston, Mass., 1989.
10. A. Poli and L. Huguet, *Error Correcting Codes, Theory and Applications*, Prentice Hall, Hemel Hempstead, UK, 1992.
11. S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, Englewood Cliffs, N.J., 1995.
12. T. Klove and V. I. Korzhik, *Error Detecting Codes*, Kluwer Academic, Boston, Mass., 1995.
13. F. J. MacWilliams, "A Theorem on the Distribution of Weights in a Systematic Codes," *Bell Syst. Tech. J.* 42: 79–94, 1963.
14. R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell Syst. Tech. J.* 29: 147–160, April 1950.

Important Linear Block Codes

This chapter presents several classes of important linear block codes that were discovered in the early days of error-correcting codes. The first class of linear block codes for error correction was discovered by Richard W. Hamming in 1950 [1], two years after Shannon published his landmark paper, which asserted that by proper encoding of information, errors induced by a noisy channel or storage medium can be reduced to any desired level without sacrificing the rate of information transmission or storage. Hamming codes have a minimum distance of 3 and therefore are capable of correcting any single error over the span of the code block length. The weight enumerator of Hamming codes is known. Hamming codes are *perfect codes* and can be decoded easily using a table-lookup scheme. Good codes with a minimum distance of 4 for single-error correction and double-error detection can be obtained by properly shortening the Hamming codes. Hamming codes and their shortened versions have been widely used for error control in digital communication and data storage systems over the years owing to their high rates and decoding simplicity.

The second class of linear block codes constructed in the early days for error correction and detection was the class of Reed–Muller codes. Reed–Muller codes were first discovered by David E. Muller in 1954 [9] for switching-circuit design and error detection. It was Irwin S. Reed, also in 1954 [10], who reformulated these codes for error correction and detection in communication and data storage systems and devised the first decoding algorithm for these codes. Reed–Muller codes form a large class of codes for multiple random error correction. These codes are simple in construction and rich in structural properties. They can be decoded in many ways using either hard-decision or soft-decision decoding algorithms. The Reed decoding algorithm for Reed–Muller codes is a majority-logic decoding algorithm that can easily be implemented. Several soft-decision decoding algorithms for Reed–Muller codes have been devised that achieve very good error performance with low decoding complexity.

Also presented in this chapter is the (24, 12) Golay code with minimum distance 8. This code has many interesting structural properties and has been extensively studied by many coding theorists and mathematicians. It has been used for error control in many communication systems, especially by U.S. space communication programs.

Several additional code construction methods are also presented in this chapter. These construction methods allow us to construct long, powerful codes from short component codes.

4.1 HAMMING CODES

For any positive integer $m \geq 3$, there exists a Hamming code with the following parameters:

$$\begin{array}{ll} \text{Code length:} & n = 2^m - 1, \\ \text{Number of information symbols:} & k = 2^m - m - 1, \\ \text{Number of parity-check symbols:} & n - k = m, \\ \text{Error-correcting capability:} & t = 1 (d_{\min} = 3). \end{array}$$

The parity-check matrix \mathbf{H} of this code consists of all the nonzero m -tuples as its columns. In systematic form, the columns of \mathbf{H} are arranged in the following form:

$$\mathbf{H} = [\mathbf{I}_m \quad \mathbf{Q}],$$

where \mathbf{I}_m is an $m \times m$ identity matrix, and the submatrix \mathbf{Q} consists of $2^m - m - 1$ columns that are the m -tuples of weight 2 or more. For example, let $m = 3$. The parity-check matrix of a Hamming code of length 7 can be put in the form

$$\mathbf{H} = \left[\begin{array}{ccccccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array} \right],$$

which is the parity-check matrix of the $(7, 4)$ linear code given in Table 3.1 (see Example 3.3). Hence, the code given in Table 3.1 is a Hamming code. The columns of \mathbf{Q} may be arranged in any order without affecting the distance property and weight distribution of the code. In systematic form, the generator matrix of the code is

$$\mathbf{G} = [\mathbf{Q}^T \quad \mathbf{I}_{2^m - m - 1}],$$

where \mathbf{Q}^T is the transpose of \mathbf{Q} , and $\mathbf{I}_{2^m - m - 1}$ is a $(2^m - m - 1) \times (2^m - m - 1)$ identity matrix.

Because the columns of \mathbf{H} are nonzero and distinct, no two columns add to zero. It follows from Corollary 3.2.1 that the minimum distance of a Hamming code is at least 3. Since \mathbf{H} consists of all the nonzero m -tuples as its columns, the vector sum of any two columns, say \mathbf{h}_i and \mathbf{h}_j , must also be a column in \mathbf{H} , say \mathbf{h}_l . Thus,

$$\mathbf{h}_i + \mathbf{h}_j + \mathbf{h}_l = 0.$$

It follows from Corollary 3.2.2 that the minimum distance of a Hamming code is exactly 3. Hence, the code is capable of correcting all the error patterns with a single error or detecting all the error patterns of two or fewer errors.

If we form the standard array for the Hamming code of length $2^m - 1$, all the $(2^m - 1)$ -tuples of weight 1 can be used as coset leaders (Theorem 3.5). The number of $(2^m - 1)$ -tuples of weight 1 is $2^m - 1$. Because $n - k = m$, the code has 2^m cosets. Thus, the zero vector $\mathbf{0}$ and the $(2^m - 1)$ -tuples of weight 1 form all the coset leaders of the standard array. Thus, a Hamming code corrects only error patterns of single error and no others. This is a very interesting structure. A t -error-correcting code is called a *perfect code* if its standard array has all the

error patterns of t or fewer errors and no others as coset leaders. Thus, Hamming codes form a class of single-error-correcting perfect codes [2]. Perfect codes are rare. Besides the Hamming codes, the only other nontrivial binary perfect code is the (23, 12) Golay code (see Section 5.9) [3–5].

Hamming codes can easily be decoded with the table-lookup scheme described in Section 3.5. The decoder for a Hamming code of length $2^m - 1$ can be implemented in the same manner as that for the (7, 4) Hamming code given in Example 3.9.

We may delete any l columns from the parity-check matrix \mathbf{H} of a Hamming code. This deletion results in an $m \times (2^m - l - 1)$ matrix \mathbf{H}' . Using \mathbf{H}' as a parity-check matrix, we obtain a shortened Hamming code with the following parameters:

$$\begin{aligned} \text{Code length:} & n = 2^m - l - 1 \\ \text{Number of information symbols:} & k = 2^m - m - l - 1 \\ \text{Number of parity-check symbols:} & n - k = m \\ \text{Error-correcting capability:} & d_{\min} \geq 3. \end{aligned}$$

If we delete columns from \mathbf{H} properly, we may obtain a shortened Hamming code with a minimum distance of 4. For example, if we delete from the submatrix \mathbf{Q} all the columns of even weight, we obtain an $m \times 2^{m-1}$ matrix,

$$\mathbf{H}' = [\mathbf{I}_m \quad \mathbf{Q}'],$$

where \mathbf{Q}' consists of $2^{m-1} - m$ columns of odd weight. Because all the columns of \mathbf{H}' have odd weight, no three columns add to zero; however, for a column \mathbf{h}_i of weight 3 in \mathbf{Q}' , there exist three columns \mathbf{h}_j , \mathbf{h}_l , and \mathbf{h}_s in \mathbf{I}_m such that $\mathbf{h}_i + \mathbf{h}_j + \mathbf{h}_l + \mathbf{h}_s = \mathbf{0}$. Thus, the shortened Hamming code with \mathbf{H}' as a parity-check matrix has a minimum distance of exactly 4.

The distance-4 shortened Hamming code can be used for correcting all error patterns of single error and simultaneously detecting all error patterns of double errors. When a single error occurs during the transmission of a code vector, the resultant syndrome is nonzero, and it contains an odd number of 1's; however, when double errors occur, the syndrome is also nonzero, but it contains an even number of 1's. Based on these facts, decoding can be accomplished as follows:

1. If the syndrome \mathbf{s} is zero, we assume that no error occurred.
2. If \mathbf{s} is nonzero and it contains an odd number of 1's, we assume that a single error occurred. The error pattern of a single error that corresponds to \mathbf{s} is added to the received vector for error correction.
3. If \mathbf{s} is nonzero and it contains an even number of 1's, an uncorrectable error pattern has been detected.

The weight distribution of a Hamming code of length $n = 2^m - 1$ is known [2]. The number of code vectors of weight i , A_i , is simply the coefficient of z^i in the expansion of the following polynomial:

$$A(z) = \frac{1}{n+1} \{(1+z)^n + n(1-z)(1-z^2)^{(n-1)/2}\}. \quad (4.1)$$

This polynomial is the weight enumerator for the Hamming codes.

EXAMPLE 4.1

Let $m = 3$. Then, $n = 2^3 - 1 = 7$, and the weight enumerator for the $(7, 4)$ Hamming code is

$$A(z) = \frac{1}{8} \{(1+z)^7 + 7(1-z)(1-z^2)^3\} = 1 + 7z^3 + 7z^4 + z^7.$$

Hence, the weight distribution for the $(7, 4)$ Hamming code is $A_0 = 1$, $A_3 = A_4 = 7$, and $A_7 = 1$.

The dual code of a $(2^m - 1, 2^m - m - 1)$ Hamming code is a $(2^m - 1, m)$ linear code. This code has a very simple weight distribution; it consists of the all-zero codeword and $2^m - 1$ codewords of weight 2^{m-1} . Thus, its weight enumerator is

$$B(z) = 1 + (2^m - 1)z^{2^{m-1}}. \quad (4.2)$$

If a Hamming code is used for error detection over a BSC, its probability of an undetected error, $P_u(E)$, can be computed either from (3.35) and (4.1) or from (3.36) and (4.2). Computing $P_u(E)$ from (3.36) and (4.2) is easier. Combining (3.36) and (4.2), we obtain

$$P_u(E) = 2^{-m} \{1 + (2^m - 1)(1 - 2p)^{2^{m-1}}\} - (1-p)^{2^m - 1}. \quad (4.3)$$

The probability $P_u(E)$ for Hamming codes does satisfy the upper bound $2^{-(n-k)} = 2^{-m}$ for $p \leq \frac{1}{2}$ (i.e., $P_u(E) \leq 2^{-m}$) [6, 7], as can be shown by using the expression of (4.3) (see Problem 4.3). Therefore, Hamming codes are good error-detection codes.

4.2 A CLASS OF SINGLE-ERROR-CORRECTING AND DOUBLE-ERROR-DETECTING CODES

Single-error-correcting and double-error-detecting (SEC-DED) codes have been widely used for error control in communication and computer memory systems. In this section we present a class of SEC-DED codes that are suitable and commonly used for improving computer memory reliability. This class of codes was constructed by Hsiao [8]. The most important feature of this class of codes is their fast encoding and error detection in the decoding process, which are the most critical on-line processes in memory operations.

SEC-DED codes that have the features described can be constructed by shortening Hamming codes presented in the previous section. Construction begins with a Hamming code of length $2^m - 1$ and minimum distance 3. Let H be its parity-check matrix. Delete columns from H such that the new parity-check matrix \mathbf{H}_0 satisfies the following requirements:

1. Every column should have an odd number of 1's.
2. The total number of 1's in the \mathbf{H}_0 matrix should be a minimum.
3. The number of 1's in each row of \mathbf{H}_0 should be made equal, or as close as possible, to the average number (i.e., the total number of 1's in \mathbf{H}_0 divided by the number of rows).

TABLE 4.1: Parameters of a list of Hsiao's codes.

n	k	$n - k$	Total number of 1's in \mathbf{H}	Average number of 1's per row
12	8	4	16	4
14	9	5	32	6.4
15	10	5	35	7
16	11	5	40	8
22	16	6	54	9
26	20	6	66	11
30	24	6	86	14.3
39	32	7	103	14.7
43	36	7	117	16.7
47	40	7	157	22.4
55	48	7	177	25.3
72	64	8	216	27
80	72	8	256	32
88	80	8	296	37
96	88	8	336	42
104	96	8	376	47
112	104	8	416	52
120	112	8	456	57
128	120	8	512	64
130	121	9	446	49.6
137	128	9	481	53.5

The first requirement guarantees the code generated by \mathbf{H}_0 has a minimum distance of at least 4. Therefore, it can be used for single-error correction and double-error detection. The second and third requirements yields minimum logic levels in forming parity or syndrome bits, and less hardware in implementation of the code. Hsiao [8] provided an algorithm to construct \mathbf{H}_0 and found some optimal SEC-DED codes. Several parity-check matrices in systematic form for message (or data) lengths of 16, 32, and 64 are given in Figure 4.1. The parameters of a list of Hsiao's codes are given in Table 4.1.

In computer applications, these codes are encoded and decoded in parallel manner. In encoding, the message bits enter the encoding circuit in parallel, and the parity-check bits are formed simultaneously. In decoding, the received bits enter the decoding circuit in parallel, the syndrome bits are formed simultaneously, and the received bits are corrected in parallel. Single-error correction is accomplished by the table-lookup decoding described in Example 3.9. Double-error detection is accomplished by examining the number of 1's in the syndrome vector \mathbf{s} . If the syndrome \mathbf{s} contains an even number of 1's, then either a double-error pattern or a multiple-even-error pattern has occurred.

Consider the parity-check matrix of the $(72, 64)$ SEC-DED code given in Figure 4.1(c). Each row contains 27 ones. If three-input X-OR gates are used to form syndrome bits, each syndrome bit is formed by a three-level X-OR tree with

$$\mathbf{H}_0 = \begin{bmatrix} 1000001001100100111100 \\ 010000001111010001010 \\ 001000111011100110000 \\ 000100111000011101000 \\ 0000100001001111000111 \\ 000001010001000011111 \end{bmatrix}$$

(a)

$$\mathbf{H}_0 = \begin{bmatrix} 1000000100010101000010000011100011011 \\ 0100000000100000011110111000101100001 \\ 001000000010110111100001001001010100110 \\ 00010001111111000000011010010001000100 \\ 00001000110110011111110000100000001000 \\ 000001000100010010010011111110010000 \\ 00000011100000101001000010000001111111 \end{bmatrix}$$

(b)

$$\mathbf{H}_0 = \begin{bmatrix} 01000000 & 01100100 & 11111111 & 00000111 & 00111000 & 11001000 & 00001000 & 00001001 & 10010010 \\ 00100000 & 10010010 & 01100100 & 11111111 & 00000111 & 00111000 & 11001000 & 00001000 & 00001001 \\ 00010000 & 00001001 & 10010010 & 01100100 & 11111111 & 00000111 & 00111000 & 11001000 & 00001000 \\ 00001000 & 00001000 & 00001001 & 10010010 & 01100100 & 11111111 & 00000111 & 00111000 & 00001000 \\ 00000100 & 11001000 & 00001000 & 00001001 & 10010010 & 01100100 & 11111111 & 00000111 & 11001000 \\ 00000010 & 00111000 & 11001000 & 00001000 & 00001001 & 10010010 & 01100100 & 11111111 & 00000111 \\ 00000001 & 00000111 & 00111000 & 11001000 & 00001000 & 00001001 & 10010010 & 01100100 & 11111111 \end{bmatrix}$$

(c)

$$\mathbf{H}_0 = \begin{bmatrix} 10000000 & 11111111 & 00001111 & 00001111 & 00001100 & 01101000 & 10001000 & 10001000 & 10000000 \\ 01000000 & 11110000 & 11111111 & 00000000 & 11110011 & 01100100 & 01000100 & 01000100 & 01000000 \\ 00100000 & 00110000 & 11110000 & 11111111 & 00001111 & 00000010 & 00100010 & 00100010 & 00100110 \\ 00010000 & 11001111 & 00000000 & 11110000 & 11111111 & 00000001 & 00010001 & 00010001 & 00010110 \\ 00001000 & 01101000 & 10001000 & 10001000 & 10000000 & 11111111 & 00001111 & 00000000 & 11110011 \\ 00000100 & 01100100 & 01000100 & 01000100 & 01000000 & 11110000 & 11111111 & 00001111 & 00001100 \\ 00000010 & 00000010 & 00100010 & 00100010 & 00100110 & 11001111 & 00000000 & 11111111 & 00001111 \\ 00000001 & 00000001 & 00010001 & 00010001 & 00010110 & 00110000 & 11110000 & 11110000 & 11111111 \end{bmatrix}$$

(d)

FIGURE 4.1: (a) Parity-check matrix of an optimal (22, 16) SEC-DED code; (b) parity-check matrix of an optimal (39, 32) SEC-DED code; (c) parity-check matrix of an optimal (72, 64) SEC-DED code; (d) parity-check matrix of another optimal (72, 64) SEC-DED code.

13 gates. The eight X-OR trees for generating the eight syndrome bits are identical. These provide uniform and minimum delay in the error-correction process.

4.3 REED–MULLER CODES

Reed–Muller (RM) codes form a class of multiple-error-correction codes. These codes were discovered by Muller in 1954 [9], but the first decoding algorithm for these codes was devised by Reed, also in 1954 [10]. They are simple in construction and rich in structural properties. They can be decoded easily in several ways using either hard- or soft-decision algorithms.

For any integers m and r with $0 \leq r \leq m$, there exists a binary r th-order RM code, denoted by $\text{RM}(r, m)$, with the following parameters:

Code length: $n = 2^m$,

Dimension: $k(r, m) = 1 + \binom{m}{1} + \binom{m}{2} + \cdots + \binom{m}{r}$,

Minimum distance: $d_{\min} = 2^{m-r}$,

where $\binom{m}{i} = \frac{m!}{(m-i)i!}$ is the binomial coefficient. For example, let $m = 5$ and $r = 2$. Then, $n = 32$, $k(2, 5) = 16$, and $d_{\min} = 8$. There exists a $(32, 16)$ RM code with a minimum distance of 8.

For $1 \leq i \leq m$, let \mathbf{v}_i be a 2^m -tuple over $GF(2)$ of the following form:

$$\mathbf{v}_i = (\underbrace{0 \cdots 0}_{2^{i-1}}, \underbrace{1 \cdots 1}_{2^{i-1}}, \underbrace{0 \cdots 0}_{2^{i-1}}, \dots, \underbrace{1 \cdots 1}_{2^{i-1}}) \quad (4.4)$$

which consists of 2^{m-i+1} alternating all-zero and all-one 2^{i-1} -tuples. For $m = 4$, we have the following four 16-tuples:

$$\mathbf{v}_4 = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1),$$

$$\mathbf{v}_3 = (0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1),$$

$$\mathbf{v}_2 = (0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1),$$

$$\mathbf{v}_1 = (0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1).$$

Let $\mathbf{a} = (a_0, a_1, \dots, a_{n-1})$ and $\mathbf{b} = (b_0, b_1, \dots, b_{n-1})$ be two binary n -tuples. We define the following logic (Boolean) product of \mathbf{a} and \mathbf{b} :

$$\mathbf{a} \cdot \mathbf{b} \stackrel{\Delta}{=} (a_0 \cdot b_0, a_1 \cdot b_1, \dots, a_{n-1} \cdot b_{n-1}),$$

where “ \cdot ” denotes the logic product (or AND operation), i.e., $a_i \cdot b_i = 1$ if and only if $a_i = b_i = 1$. For $m = 4$,

$$\mathbf{v}_3 \cdot \mathbf{v}_2 = (0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1).$$

For simplicity, we use \mathbf{ab} for $\mathbf{a} \cdot \mathbf{b}$.

Let \mathbf{v}_0 denote the all-one 2^m -tuple, $\mathbf{v}_0 = (1, 1, \dots, 1)$. For $1 \leq i_1 < i_2 < \dots < i_l \leq m$, the product vector

$$\mathbf{v}_{i_1} \mathbf{v}_{i_2} \cdots \mathbf{v}_{i_l}$$

is said to have degree l . Because the weights of $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m$ are even and powers of 2, the weight of the product $\mathbf{v}_{i_1} \mathbf{v}_{i_2} \cdots \mathbf{v}_{i_l}$ is also even and a power of 2, in fact, 2^{m-l} .

The r th-order RM code, $\text{RM}(r, m)$, of length 2^m is generated (or spanned) by the following set of independent vectors:

$$\begin{aligned} G_{\text{RM}}(r, m) = \{ & \mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_m, \mathbf{v}_1 \mathbf{v}_2, \mathbf{v}_1 \mathbf{v}_3, \dots, \mathbf{v}_{m-1} \mathbf{v}_m, \\ & \dots, \text{up to products of degree } r \}. \end{aligned} \quad (4.5)$$

There are

$$k(r, m) = 1 + \binom{m}{1} + \binom{m}{2} + \dots + \binom{m}{r}$$

vectors in $G_{\text{RM}}(r, m)$. Therefore, the dimension of the code is $k(r, m)$.

If the vectors in $G_{\text{RM}}(r, m)$ are arranged as rows of a matrix, then the matrix is a generator matrix of the $\text{RM}(r, m)$ code. Hereafter, we use $G_{\text{RM}}(r, m)$ as the generator matrix. For $0 \leq l \leq r$, there are exactly $\binom{m}{l}$ rows in $G_{\text{RM}}(r, m)$ of weight 2^{m-l} . Because all the vectors in $G_{\text{RM}}(r, m)$ are of even weight, all the codewords in the $\text{RM}(r, m)$ code have even weight. From the code construction we readily see that the $\text{RM}(r-1, m)$ code is a proper subcode of the $\text{RM}(r, m)$ code. Hence, we have the following inclusion chain:

$$\text{RM}(0, m) \subset \text{RM}(1, m) \subset \dots \subset \text{RM}(r, m). \quad (4.6)$$

Furthermore, RM codes have the following structural property: the $(m-r-1)$ th-order RM code, $\text{RM}(m-r-1, m)$, is the dual code of the r th-order RM code, $\text{RM}(r, m)$ (see Problem 4.9). The zeroth-order RM code is a repetition code and the $(m-1)$ th-order RM code is a single-parity-check code.

EXAMPLE 4.2

Let $m = 4$ and $r = 2$. The second-order RM code of length $n = 16$ is generated by the following 11 vectors:

\mathbf{v}_0	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
\mathbf{v}_4	0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
\mathbf{v}_3	0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1
\mathbf{v}_2	0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1
\mathbf{v}_1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
$\mathbf{v}_3 \mathbf{v}_4$	0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1
$\mathbf{v}_2 \mathbf{v}_4$	0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1
$\mathbf{v}_1 \mathbf{v}_4$	0 0 0 0 0 0 0 0 1 0 1 0 1 0 1
$\mathbf{v}_2 \mathbf{v}_3$	0 0 0 0 0 1 1 0 0 0 0 0 1 1
$\mathbf{v}_1 \mathbf{v}_3$	0 0 0 0 1 0 1 0 0 0 0 0 1 0 1
$\mathbf{v}_1 \mathbf{v}_2$	0 0 1 0 0 0 1 0 0 0 1 0 0 0 1

This is a $(16, 11)$ code with a minimum distance of 4.

With the preceding construction, the generator matrix $G_{\text{RM}}(r, m)$ of the $\text{RM}(r, m)$ code is not in systematic form. It can be put in systematic form with

elementary row and column operations; however, RM codes in nonsystematic form have many interesting and useful structures that reduce decoding complexity. This topic will be discussed in a later chapter.

The Reed decoding algorithm for RM codes is best explained by an example. Consider the second-order RM code RM(2, 4) of length 16 given in Example 4.2. Suppose

$$(a_0, a_4, a_3, a_2, a_1, a_{34}, a_{24}, a_{14}, a_{23}, a_{13}, a_{12})$$

is the message to be encoded. Then, the corresponding codeword is

$$\begin{aligned} (b_0, b_1, b_2, \dots, b_{15}) = & a_0 \mathbf{v}_0 + a_4 \mathbf{v}_4 + a_3 \mathbf{v}_3 + a_2 \mathbf{v}_2 + a_1 \mathbf{v}_1 \\ & + a_{34} \mathbf{v}_3 \mathbf{v}_4 + a_{24} \mathbf{v}_2 \mathbf{v}_4 + a_{14} \mathbf{v}_1 \mathbf{v}_4 \\ & + a_{23} \mathbf{v}_2 \mathbf{v}_3 + a_{13} \mathbf{v}_1 \mathbf{v}_3 + a_{12} \mathbf{v}_1 \mathbf{v}_2. \end{aligned}$$

Note that the sum of the first four components of each generator vector is zero except the vector $\mathbf{v}_1 \mathbf{v}_2$. The same is true for the other three groups of four consecutive components. As a result, we have the following four sums that relate the information bit a_{12} to the code bits:

$$\begin{aligned} a_{12} &= b_0 + b_1 + b_2 + b_3, \\ a_{12} &= b_4 + b_5 + b_6 + b_7, \\ a_{12} &= b_8 + b_9 + b_{10} + b_{11}, \\ a_{12} &= b_{12} + b_{13} + b_{14} + b_{15}. \end{aligned}$$

These four sums give four independent determinations (or reconstructions) of the information bit a_{12} from the code bits. If the codeword $(b_0, b_1, \dots, b_{15})$ is transmitted and there is a single transmission error in the received vector, the error can affect only one determination of a_{12} . As a result, the other three (*majority*) determinations of a_{12} will give the correct value of a_{12} . This is the basis for decoding RM codes.

Let $\mathbf{r} = (r_0, r_1, \dots, r_{15})$ be the received vector. In decoding a_{12} , we form the following sums:

$$\begin{aligned} A_1 &= r_0 + r_1 + r_2 + r_3, \\ A_2 &= r_4 + r_5 + r_6 + r_7, \\ A_3 &= r_8 + r_9 + r_{10} + r_{11}, \\ A_4 &= r_{12} + r_{13} + r_{14} + r_{15}. \end{aligned}$$

which are obtained by replacing the code bits with the corresponding received bits in the four independent determinations of a_{12} . These sums are called *check-sums*, which are simply the estimates of a_{12} . Then, a_{12} is decoded based on the following *majority-logic decision rule*: a_{12} is taken to be equal to the value assumed by the majority in $\{A_1, A_2, A_3, A_4\}$. If there is a tie, a random choice of the value of a_{12} is made. It is clear that if there is only one error in the received vector, a_{12} is always decoded correctly.

Similar independent determinations of information bits $a_{13}, a_{23}, a_{14}, a_{24}$, and a_{34} can be made from the code bits. For example, the four independent determinations of a_{13} are:

$$\begin{aligned}a_{13} &= b_0 + b_1 + b_4 + b_5, \\a_{13} &= b_2 + b_3 + b_6 + b_7, \\a_{13} &= b_8 + b_9 + b_{12} + b_{13}, \\a_{13} &= b_{10} + b_{11} + b_{14} + b_{15}.\end{aligned}$$

At the decoder, we decode a_{13} by forming the following four check-sums from the received bits using the preceding four independent determinations of a_{13} :

$$\begin{aligned}A_1 &= r_0 + r_1 + r_4 + r_5, \\A_2 &= r_2 + r_3 + r_6 + r_7, \\A_3 &= r_8 + r_9 + r_{12} + r_{13}, \\A_4 &= r_{10} + r_{11} + r_{14} + r_{15}.\end{aligned}$$

From these four check-sums we use the majority-logic decision rule to decode a_{13} . If there is a single transmission error in the received sequence, the information bits $a_{12}, a_{13}, a_{23}, a_{14}, a_{24}$, and a_{34} will be decoded correctly.

After the decoding of $a_{12}, a_{13}, a_{23}, a_{14}, a_{24}$, and a_{34} , the vector

$$a_{34}\mathbf{v}_3\mathbf{v}_4 + a_{24}\mathbf{v}_2\mathbf{v}_4 + a_{14}\mathbf{v}_1\mathbf{v}_4 + a_{23}\mathbf{v}_2\mathbf{v}_3 + a_{13}\mathbf{v}_1\mathbf{v}_3 + a_{12}\mathbf{v}_1\mathbf{v}_2$$

is subtracted from \mathbf{r} . The result is a modified received vector:

$$\begin{aligned}\mathbf{r}^{(1)} &= \left(r_0^{(1)}, r_1^{(1)}, \dots, r_{15}^{(1)}\right) \\&= \mathbf{r} - a_{34}\mathbf{v}_3\mathbf{v}_4 - a_{24}\mathbf{v}_2\mathbf{v}_4 - a_{14}\mathbf{v}_1\mathbf{v}_4 - a_{23}\mathbf{v}_2\mathbf{v}_3 - a_{13}\mathbf{v}_1\mathbf{v}_3 - a_{12}\mathbf{v}_1\mathbf{v}_2.\end{aligned}$$

In the absence of errors, $\mathbf{r}^{(1)}$ is simply the following codeword:

$$a_0\mathbf{v}_0 + a_4\mathbf{v}_4 + a_3\mathbf{v}_3 + a_2\mathbf{v}_2 + a_1\mathbf{v}_1 = (b_0^{(1)}, b_1^{(1)}, \dots, b_{15}^{(1)}).$$

We note that starting from the first component, the sums of every two consecutive components in $\mathbf{v}_0, \mathbf{v}_4, \mathbf{v}_3$, and \mathbf{v}_2 are zero; however, the sum of every two consecutive components of \mathbf{v}_1 is equal to 1. As a consequence, we can form the following eight independent determinations of the information bit a_1 from the code bits $b_0^{(1)}$ through $b_{15}^{(1)}$:

$$\begin{aligned}a_1 &= b_0^{(1)} + b_1^{(1)}, \quad a_1 = b_8^{(1)} + b_9^{(1)}, \\a_1 &= b_2^{(1)} + b_3^{(1)}, \quad a_1 = b_{10}^{(1)} + b_{11}^{(1)}, \\a_1 &= b_4^{(1)} + b_5^{(1)}, \quad a_1 = b_{12}^{(1)} + b_{13}^{(1)}, \\a_1 &= b_6^{(1)} + b_7^{(1)}, \quad a_1 = b_{14}^{(1)} + b_{15}^{(1)}.\end{aligned}$$

Similar independent determinations of a_2, a_3 , and a_4 can be formed. In decoding a_1 , we form the following check-sums from the bits of the modified received vector

$\mathbf{r}^{(1)}$ and the preceding eight independent determinations of a_1 :

$$\begin{aligned} A_1^{(1)} &= r_0^{(1)} + r_1^{(1)}, & A_5^{(1)} &= r_8^{(1)} + r_9^{(1)}, \\ A_2^{(1)} &= r_2^{(1)} + r_3^{(1)}, & A_6^{(1)} &= r_{10}^{(1)} + r_{11}^{(1)}, \\ A_3^{(1)} &= r_4^{(1)} + r_5^{(1)}, & A_7^{(1)} &= r_{12}^{(1)} + r_{13}^{(1)}, \\ A_4^{(1)} &= r_6^{(1)} + r_7^{(1)}, & A_8^{(1)} &= r_{14}^{(1)} + r_{15}^{(1)}. \end{aligned}$$

From these check-sums we decode a_1 by using the majority-logic decision rule. Similarly, we can decode the information bits a_2 , a_3 , and a_4 .

After the decoding of a_1 , a_2 , a_3 , and a_4 , we remove the effect of a_1 , a_2 , a_3 , and a_4 from $\mathbf{r}^{(1)}$ and form the following modified received vector:

$$\begin{aligned} \mathbf{r}^{(2)} &= (r_0^{(2)}, r_1^{(2)}, \dots, r_{15}^{(2)}) \\ &= \mathbf{r}^{(1)} - a_4\mathbf{v}_4 - a_3\mathbf{v}_3 - a_2\mathbf{v}_2 - a_1\mathbf{v}_1. \end{aligned}$$

In the absence of errors, $\mathbf{r}^{(2)}$ is the following codeword:

$$a_0\mathbf{v}_0 = (a_0, a_0, \dots, a_0).$$

This result gives 16 independent determinations of a_0 . In decoding a_0 , we simply set a_0 to the value taken by the majority of the bits in $\mathbf{r}^{(2)}$. This step completes the entire decoding.

The demonstrated decoding is referred to as *majority-logic decoding*. Because it consists of three steps (or levels) of decoding, it is called three-step majority-logic decoding. It can easily be implemented using majority-logic elements.

If there is only one error in the received vector \mathbf{r} , the information bits a_{12} , a_{13} , a_{14} , a_{23} , a_{24} , and a_{34} will be correctly decoded. Then, the modified received vector $\mathbf{r}^{(1)}$ will still contain a single error at the same location. This single error affects only one of the eight check-sums for a_i , with $1 \leq i \leq 4$. The other seven check-sums give the correct value of a_i . Therefore, the information bits a_1 , a_2 , a_3 , and a_4 will be correctly decoded. As a result, the next modified received vector $\mathbf{r}^{(2)}$ will contain only one error (still at the same location), and the information bit a_0 will be correctly decoded.

If there are two transmission errors in \mathbf{r} , these two errors may affect two check-sums for information bit a_{ij} . In this case, there is no majority in the four check-sums; two check-sums take 0, and two other check-sums take 1. A choice between these two values as the decoded value of a_{ij} may result in an incorrect decoding of a_{ij} . This incorrect decoding affects the subsequent levels of decoding and results in *error propagation*. Consequently, the decodings of a_1 , a_2 , a_3 , a_4 , and a_0 are very likely to be incorrect. Because the code guarantees correcting any single error but not two errors, its minimum distance is at least 3 but less than 5. Since all the codewords have even weight, the minimum distance of the code must be 4.

We have used an example to introduce the concepts of majority-logic decoding and the multiple-step decoding process of the Reed algorithm. Now we are ready to present the general algorithm for decoding RM codes. The major part of the decoding is to form check-sums at each decoding step.

Consider the r th-order RM code, $\text{RM}(r, m)$. Let

$$\mathbf{a} = (a_0, a_1, \dots, a_m, a_{1,2}, \dots, a_{m-1,m}, \dots, a_{1,2,\dots,r}, \dots, a_{m-r+1,m-r+2,\dots,m})$$

be the message to be encoded. The corresponding codeword is

$$\begin{aligned} \mathbf{b} &= (b_0, b_1, \dots, b_{n-1}) \\ &= a_0 \mathbf{v}_0 + \sum_{1 \leq i_1 \leq m} a_{i_1} \mathbf{v}_{i_1} + \sum_{1 \leq i_1 < i_2 \leq m} a_{i_1} a_{i_2} \mathbf{v}_{i_1} \mathbf{v}_{i_2} \\ &\quad + \dots + \sum_{1 \leq i_1 < i_2 < \dots < i_r \leq m} a_{i_1} a_{i_2} \dots a_{i_r} \mathbf{v}_{i_1} \mathbf{v}_{i_2} \dots \mathbf{v}_{i_r}. \end{aligned} \quad (4.7)$$

Let $\mathbf{r} = (r_0, r_1, \dots, r_{n-1})$ be the received vector. Decoding of $\text{RM}(r, m)$ code consists of $r + 1$ steps. At the first step of decoding, the information bits $a_{i_1} a_{i_2} \dots a_{i_r}$ corresponding to the product vectors $\mathbf{v}_{i_1} \mathbf{v}_{i_2} \dots \mathbf{v}_{i_r}$ of degree r in (4.7) are decoded based on their check-sums formed from the received bits in \mathbf{r} . Based on these decoded information bits, the received vector $\mathbf{r} = (r_0, r_1, \dots, r_{n-1})$ is modified. Let $\mathbf{r}^{(1)} = (r_0^{(1)}, r_1^{(1)}, \dots, r_{n-1}^{(1)})$ denote the modified received vector. At the second step of decoding, the bits in the modified received vector $\mathbf{r}^{(1)}$ are used to form the check-sums for decoding the information bits $a_{i_1} a_{i_2} \dots a_{i_{r-1}}$ that correspond to the product vectors $\mathbf{v}_{i_1} \mathbf{v}_{i_2} \dots \mathbf{v}_{i_{r-1}}$ of degree $r - 1$ in (4.7). Then, the decoded information bits at the second step of decoding are used to modify $\mathbf{r}^{(1)}$. The modification results in the next modified received vector $\mathbf{r}^{(2)} = (r_0^{(2)}, r_1^{(2)}, \dots, r_{n-1}^{(2)})$ for the third step of decoding. This step-by-step decoding process continues until the last information bit a_0 that corresponds to the all-one vector \mathbf{v}_0 in (4.7) is decoded. This decoding process is called $(r + 1)$ -step majority-logic decoding [2, 11].

Now, we need to know how to form check-sums for decoding at each step. For $1 \leq i_1 < i_2 < \dots < i_{r-l} \leq m$ with $0 \leq l \leq r$, we form the following index set:

$$S \stackrel{\Delta}{=} \{c_{i_1-1} 2^{i_1-1} + c_{i_2-1} 2^{i_2-1} + \dots + c_{i_{r-l}-1} 2^{i_{r-l}-1} : c_{i_j-1} \in \{0, 1\} \text{ for } 1 \leq j \leq r-l\} \quad (4.8)$$

which is a set of 2^{r-l} nonnegative integers less than 2^m in binary form. The exponent set $\{i_1 - 1, i_2 - 1, \dots, i_{r-l} - 1\}$ is a subset of $\{0, 1, \dots, m - 1\}$. Let E be the set of integers in $\{0, 1, \dots, m - 1\}$ but not in $\{i_1 - 1, i_2 - 1, \dots, i_{r-l} - 1\}$; that is,

$$\begin{aligned} E &\stackrel{\Delta}{=} \{0, 1, \dots, m - 1\} \setminus \{i_1 - 1, i_2 - 1, \dots, i_{r-l} - 1\} \\ &= \{j_1, j_2, \dots, j_{m-r+l}\}, \end{aligned} \quad (4.9)$$

where $0 \leq j_1 < j_2 < \dots < j_{m-r+l} \leq m - 1$. We form the following set of integers:

$$S^c \stackrel{\Delta}{=} \{d_{j_1} 2^{j_1} + d_{j_2} 2^{j_2} + \dots + d_{j_{m-r+l}} 2^{j_{m-r+l}} : d_{j_t} \in \{0, 1\} \text{ for } 1 \leq t \leq m - r + l\}. \quad (4.10)$$

Note that there are 2^{m-r+l} nonnegative integers in S^c , and

$$S \cap S^c = \{0\}.$$

For $l = r$, $S = \{0\}$, and $S^c = \{0, 1, \dots, 2^m - 1\}$.

For $0 \leq l \leq r$, suppose we have just completed the l th step of decoding. The decoded information bits are $a_{i_1 i_2 \dots i_{r-l+1}}^*$. We form the following modified received vector:

$$\mathbf{r}^{(l)} \triangleq \mathbf{r}^{(l-1)} - \sum_{1 \leq i_1 < \dots < i_{r-l+1} \leq m} a_{i_1 i_2 \dots i_{r-l+1}}^* \mathbf{v}_{i_1} \mathbf{v}_{i_2} \dots \mathbf{v}_{i_{r-l+1}} \quad (4.11)$$

where $\mathbf{r}^{(l-1)}$ is the modified received vector for the l th step of decoding, and $\mathbf{r}^{(0)} = \mathbf{r}$. For each integer $q \in S^c$, we form the following set of integers (called *indices*):

$$\begin{aligned} B &\stackrel{\Delta}{=} q + S \\ &= \{q + s : s \in S\}. \end{aligned} \quad (4.12)$$

Then, the check-sums for decoding the information bits $a_{i_1 i_2 \dots i_{r-l}}$ are

$$A^{(l)} = \sum_{t \in B} r_t^{(l)} \quad (4.13)$$

for $q \in S^c$. Because there are 2^{m-r+l} integers q in S^c , we can form 2^{m-r+l} check-sums for decoding each information bit $a_{i_1 i_2 \dots i_{r-l}}$.

At the first step of decoding, $l = 0$ and 2^{m-r} check-sums can be formed for decoding each information bit $a_{i_1 i_2 \dots i_r}$. If there are $2^{m-r-1} - 1$ or fewer errors in the received vector \mathbf{r} , then more than half (majority) of the check-sums assume the value of $a_{i_1 i_2 \dots i_r}$ and hence the decoding of $a_{i_1 i_2 \dots i_r}$ is correct, using the majority-logic decision rule; however, if \mathbf{r} contains 2^{m-r-1} or more errors, there is no guarantee that a majority of the check-sums will assume the value of $a_{i_1 i_2 \dots i_r}$. In this case, majority-logic decision may result in an incorrect decoding. For example, consider an error pattern with 2^{m-r-1} errors such that each error appears in a different check-sum. In this case, half of the 2^{m-r} check-sums assume the value 0, and the other half assume the value 1. There is no clear majority. A random choice of the two values may result in an incorrect decoding of $a_{i_1 i_2 \dots i_r}$. Now, consider another error pattern of $2^{m-r-1} + 1$ errors such that each error appears in a different check-sum. In this case, $2^{m-r-1} + 1$ (majority) of the check-sums assume the opposite value of $a_{i_1 i_2 \dots i_r}$ and majority-logic decision based on the check-sums results in incorrect decoding of $a_{i_1 i_2 \dots i_r}$. Note that the number of check-sums is doubled at each subsequent decoding step. If there are $2^{m-r-1} - 1$ or fewer errors in the received vector \mathbf{r} , then majority-logic decision based on check-sums results in correct decoding at each step. This implies that the minimum distance of the $\text{RM}(r, m)$ code is at least $2 \cdot (2^{m-r-1} - 1) + 1 = 2^{m-r} - 1$. Because the codewords in $\text{RM}(r, m)$ have even weights, the minimum distance is at least 2^{m-r} ; however, each product vector of degree r in the generator matrix $G_{\text{RM}}(r, m)$ has weight 2^{m-r} and is a codeword. Therefore, the minimum distance of the code is exactly 2^{m-r} .

The Reed decoding algorithm for RM codes is simply a multistage decoding algorithm in which the decoded information at each stage of decoding is passed down for the next stage of decoding.

EXAMPLE 4.3

Consider the second-order RM code of length 16 with $m = 4$ given in Example 4.2. Suppose we want to construct the check-sums for the information bit a_{12} . Because

$i_1 = 1$ and $i_2 = 2$, we obtain the following sets:

$$\begin{aligned} S &= \{c_0 + c_1 2^1 : c_0, c_1 \in \{0, 1\}\} \\ &= \{0, 1, 2, 3\}, \\ E &= \{0, 1, 2, 3\} \setminus \{0, 1\} \\ &= \{2, 3\}, \\ S^c &= \{d_2 2^2 + d_3 2^3 : d_2, d_3 \in \{0, 1\}\} \\ &= \{0, 4, 8, 12\}. \end{aligned}$$

Then, the index sets for forming the check-sums for a_{12} are

$$\begin{aligned} B_1 &= 0 + S = \{0, 1, 2, 3\}, \\ B_2 &= 4 + S = \{4, 5, 6, 7\}, \\ B_3 &= 8 + S = \{8, 9, 10, 11\}, \\ B_4 &= 12 + S = \{12, 13, 14, 15\}. \end{aligned}$$

It follows from (4.13) with $l = 0$ that the four check-sums for a_{12} are

$$\begin{aligned} A_1^{(0)} &= r_0 + r_1 + r_2 + r_3, \\ A_2^{(0)} &= r_4 + r_5 + r_6 + r_7, \\ A_3^{(0)} &= r_8 + r_9 + r_{10} + r_{11}, \\ A_4^{(0)} &= r_{12} + r_{13} + r_{14} + r_{15}. \end{aligned}$$

Now, consider the check-sums for a_{13} . Because $i_1 = 1$ and $i_2 = 3$, we obtain the following sets:

$$\begin{aligned} S &= \{c_0 + c_2 2^2 : c_0, c_2 \in \{0, 1\}\} \\ &= \{0, 1, 4, 5\}, \\ E &= \{0, 1, 2, 3\} \setminus \{0, 2\} \\ &= \{1, 3\}, \\ S^c &= \{d_1 2 + d_3 2^3 : d_1, d_3 \in \{0, 1\}\} \\ &= \{0, 2, 8, 10\}. \end{aligned}$$

The index sets for constructing the check-sums for a_{13} are

$$\begin{aligned} B_1 &= 0 + S = \{0, 1, 4, 5\}, \\ B_2 &= 2 + S = \{2, 3, 6, 7\}, \end{aligned}$$

$$B_3 = 8 + S = \{8, 9, 12, 13\},$$

$$B_4 = 10 + S = \{10, 11, 14, 15\}.$$

From these index sets and (4.13), we obtain the following check-sums for a_{13} :

$$A_1^{(0)} = r_0 + r_1 + r_4 + r_5,$$

$$A_2^{(0)} = r_2 + r_3 + r_6 + r_7,$$

$$A_3^{(0)} = r_8 + r_9 + r_{12} + r_{13},$$

$$A_4^{(0)} = r_{10} + r_{11} + r_{14} + r_{15}.$$

Using the same procedure, we can form the check-sums for information bits a_{14} , a_{23} , a_{24} , and a_{34} .

To find the check-sums for a_1 , a_2 , a_3 , and a_4 , we first form the modified received vector $\mathbf{r}^{(1)}$ based on (4.11):

$$\mathbf{r}^{(1)} = \mathbf{r} - \sum_{1 \leq i_1 < i_2 \leq 4} a_{i_1 i_2}^* \mathbf{v}_{i_1} \mathbf{v}_{i_2}.$$

Suppose we want to form the check-sums for a_3 . Because $i_1 = 3$, we obtain the following sets:

$$S = \{c_2 2^2 : c_2 \in \{0, 1\}\} = \{0, 4\},$$

$$E = \{0, 1, 2, 3\} \setminus \{2\} = \{0, 1, 3\},$$

$$\begin{aligned} S^c &= \{d_0 + d_1 2 + d_3 2^3 : d_0, d_1, d_3 \in \{0, 1\}\} \\ &= \{0, 1, 2, 3, 8, 9, 10, 11\}. \end{aligned}$$

Then, the index sets for forming the check-sums of a_3 are

$$B_1 = \{0, 4\}, \quad B_5 = \{8, 12\},$$

$$B_2 = \{1, 5\}, \quad B_6 = \{9, 13\},$$

$$B_3 = \{2, 6\}, \quad B_7 = \{10, 14\},$$

$$B_4 = \{3, 7\}, \quad B_8 = \{11, 15\}.$$

It follows from (4.13) with $l = 1$ that we obtain the following eight check-sums:

$$\begin{aligned} A_1^{(1)} &= r_0^{(1)} + r_4^{(1)}, & A_5^{(1)} &= r_8^{(1)} + r_{12}^{(1)}, \\ A_2^{(1)} &= r_1^{(1)} + r_5^{(1)}, & A_6^{(1)} &= r_9^{(1)} + r_{13}^{(1)}, \\ A_3^{(1)} &= r_2^{(1)} + r_6^{(1)}, & A_7^{(1)} &= r_{10}^{(1)} + r_{14}^{(1)}, \\ A_4^{(1)} &= r_3^{(1)} + r_7^{(1)}, & A_8^{(1)} &= r_{11}^{(1)} + r_{15}^{(1)}. \end{aligned}$$

Similarly, we can form the check-sums for a_1 , a_2 , and a_4 .

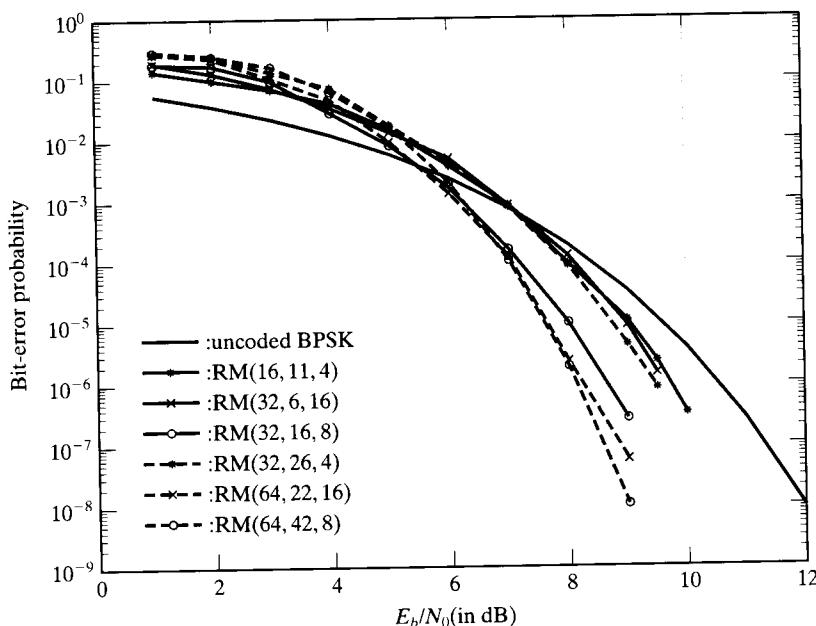


FIGURE 4.2: Bit-error performances of some RM codes with majority-logic decoding.

Error performances of some RM codes of lengths up to 64 using majority-logic decoding are shown in Figure 4.2.

4.4 OTHER CONSTRUCTIONS FOR REED–MULLER CODES

Besides the construction method presented in Section 4.3, there are other methods for constructing RM codes. We present three such methods in this section. These methods reveal more structures of RM code that are useful in constructing trellis diagrams and soft-decision decodings.

Let $A = [a_{ij}]$ be an $m \times m$ matrix and $B = [b_{ij}]$ be an $n \times n$ matrix over $GF(2)$. The *Kronecker product* of A and B , denoted by $A \otimes B$, is the $mn \times mn$ matrix obtained from A by replacing every entry a_{ij} with the matrix $a_{ij}B$. Note that for $a_{ij} = 1$, $a_{ij}B = B$, and for $a_{ij} = 0$, $a_{ij}B$ is an $n \times n$ zero matrix. Let

$$G_{(2,2)} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad (4.14)$$

be a 2×2 matrix over $GF(2)$. The *twofold Kronecker product* of $G_{(2,2)}$ is defined as

$$G_{(2^2, 2^2)} \triangleq \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (4.15)$$

The three-fold Kronecker product of $G_{(2,2)}$ is defined as

$$\begin{aligned}
G_{(2^3, 2^3)} &\triangleq \left[\begin{array}{cc} 1 & 1 \\ 0 & 1 \end{array} \right] \otimes \left[\begin{array}{cc} 1 & 1 \\ 0 & 1 \end{array} \right] \otimes \left[\begin{array}{cc} 1 & 1 \\ 0 & 1 \end{array} \right] \\
&= \left[\begin{array}{cc} 1 & 1 \\ 0 & 1 \end{array} \right] \otimes \left[\begin{array}{cccc} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{array} \right] \\
&= \left[\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array} \right]. \tag{4.16}
\end{aligned}$$

Similarly, we can define the m -fold Kronecker product of $G_{(2,2)}$. Let $n = 2^m$. We use $G_{(n,n)}$ to denote the m -fold Kronecker product of $G_{(2,2)}$. $G_{(n,n)}$ is a $2^m \times 2^m$ matrix over $GF(2)$. The rows of $G_{(n,n)}$ have weights $2^0, 2^1, 2^2, \dots, 2^m$, and the number of rows with weight 2^{m-l} is $\binom{m}{l}$ for $0 \leq l \leq m$.

The generator matrix $G_{\text{RM}}(r, m)$ of the r th-order RM code $\text{RM}(r, m)$ of length $n = 2^m$ consists of those rows of $G_{(n,n)}$ with weights equal to or greater than 2^{m-r} . These rows are the same vectors given by (4.5), except they are a different permutation.

EXAMPLE 4.4

Let $m = 4$. The fourfold Kronecker product of $G_{(2,2)}$ is

The generator matrix $G_{\text{RM}}(2, 4)$ of the second-order RM code, $\text{RM}(2, 4)$, of length 16 consists of the rows in $G_{(16, 16)}$ with weights 2^2 , 2^3 , and 2^4 . Thus, we obtain

$$G_{\text{RM}}(2, 4) = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix},$$

which is exactly the same matrix given in Example 4.2, except for the ordering of the rows.

Let $\mathbf{u} = (u_0, u_1, \dots, u_{n-1})$ and $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ be two n -tuples over $GF(2)$. From \mathbf{u} and \mathbf{v} we form the following $2n$ -tuple:

$$|\mathbf{u}|\mathbf{u} + \mathbf{v} \stackrel{\Delta}{=} (u_0, u_1, \dots, u_{n-1}, u_0 + v_0, u_1 + v_1, \dots, u_{n-1} + v_{n-1}). \quad (4.17)$$

For $i = 1, 2$, let C_i be a binary (n, k_i) linear code with generator matrix G_i and minimum distance d_i , respectively. Assume that $d_2 > d_1$. We form the following linear code of length $2n$:

$$\begin{aligned} C &= |C_1|C_1 + C_2| \\ &= \{|\mathbf{u}|\mathbf{u} + \mathbf{v}| : \mathbf{u} \in C_1 \text{ and } \mathbf{v} \in C_2\}. \end{aligned} \quad (4.18)$$

C is a binary $(2n, k_1 + k_2)$ linear code with generator matrix

$$G = \begin{bmatrix} G_1 & G_1 \\ \mathbf{0} & G_2 \end{bmatrix} \quad (4.19)$$

where $\mathbf{0}$ is a $k_2 \times n$ zero matrix. The minimum distance $d_{\min}(C)$ of C is

$$d_{\min}(C) = \min\{2d_1, d_2\}. \quad (4.20)$$

To prove this, let $\mathbf{x} = |\mathbf{u}|\mathbf{u} + \mathbf{v}$ and $\mathbf{y} = |\mathbf{u}'|\mathbf{u}' + \mathbf{v}'$ be two distinct codewords in C . The Hamming distance between \mathbf{x} and \mathbf{y} can be expressed in terms of Hamming weights as follows:

$$d(\mathbf{x}, \mathbf{y}) = w(\mathbf{u} + \mathbf{u}') + w(\mathbf{u} + \mathbf{u}' + \mathbf{v} + \mathbf{v}'), \quad (4.21)$$

where $w(\mathbf{z})$ denotes the Hamming weight of \mathbf{z} . There are two cases to be considered, $\mathbf{v} = \mathbf{v}'$ and $\mathbf{v} \neq \mathbf{v}'$. If $\mathbf{v} = \mathbf{v}'$, since $\mathbf{x} \neq \mathbf{y}$, we must have $\mathbf{u} \neq \mathbf{u}'$. In this case,

$$d(\mathbf{x}, \mathbf{y}) = w(\mathbf{u} + \mathbf{u}') + w(\mathbf{u} + \mathbf{u}'). \quad (4.22)$$

Because $\mathbf{u} + \mathbf{u}'$ is a nonzero codeword in C_1 , $w(\mathbf{u} + \mathbf{u}') \geq d_1$. Then, it follows from (4.22) that

$$d(\mathbf{x}, \mathbf{y}) \geq 2d_1. \quad (4.23)$$

If $\mathbf{v} \neq \mathbf{v}'$, we have

$$\begin{aligned} d(\mathbf{x}, \mathbf{y}) &\geq w(\mathbf{u} + \mathbf{u}') + w(\mathbf{v} + \mathbf{v}') - w(\mathbf{u} + \mathbf{u}') \\ &= w(\mathbf{v} + \mathbf{v}'). \end{aligned} \quad (4.24)$$

Since $\mathbf{v} + \mathbf{v}'$ is a nonzero codeword in C_2 , $w(\mathbf{v} + \mathbf{v}') \geq d_2$. From (4.24) we have

$$d(\mathbf{x}, \mathbf{y}) \geq d_2. \quad (4.25)$$

Inequalities (4.23) and (4.25) imply that

$$d(\mathbf{x}, \mathbf{y}) \geq \min\{2d_1, d_2\}. \quad (4.26)$$

Because \mathbf{x} and \mathbf{y} are two arbitrary different codewords in C , the minimum distance $d_{\min}(C)$ must be lower bounded as follows:

$$d_{\min}(C) \geq \min\{2d_1, d_2\}. \quad (4.27)$$

Let \mathbf{u}_0 and \mathbf{v}_0 be two minimum-weight codewords in C_1 and C_2 , respectively. Then, $w(\mathbf{u}_0) = d_1$ and $w(\mathbf{v}_0) = d_2$. The vector $|\mathbf{u}_0|\mathbf{u}_0$ is a codeword in C with weight $w(|\mathbf{u}_0|\mathbf{u}_0|) = 2d_1$. The vector $|\mathbf{0}|\mathbf{v}_0$ is also a codeword in C with weight $w(|\mathbf{0}|\mathbf{v}_0|) = d_2$. From (4.27) we see that $d_{\min}(C)$ must be either $2d_1$ or d_2 . Therefore, we conclude that

$$d_{\min}(C) = \min\{2d_1, d_2\}. \quad (4.28)$$

The preceding construction of a code from two component codes is called the $|\mathbf{u}|\mathbf{u} + |\mathbf{v}|\mathbf{v}$ -construction [12, 13] which is a powerful technique for constructing powerful long codes from short codes.

EXAMPLE 4.5

Let C_1 be the binary $(8, 4)$ linear code of minimum distance 4 generated by

$$G_1 = \left[\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array} \right].$$

Let C_2 be the $(8, 1)$ repetition code of minimum distance 8 generated by

$$G_2 = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1].$$

Using $|\mathbf{u}|\mathbf{u} + \mathbf{v}$ -construction, we obtain a $(16, 5)$ binary linear code of minimum distance 8 with the following generator matrix,

$$G = \begin{bmatrix} G_1 & G_1 \\ 0 & G_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

RM codes of length 2^m can be constructed from RM codes of length 2^{m-1} using the $|\mathbf{u}|\mathbf{u} + \mathbf{v}$ -construction [12]. For $m \geq 2$, the r th-order RM code in $|\mathbf{u}|\mathbf{u} + \mathbf{v}$ -construction is given as follows:

$$\text{RM}(r, m) = \{|\mathbf{u}|\mathbf{u} + \mathbf{v} : \mathbf{u} \in \text{RM}(r, m-1) \text{ and } \mathbf{v} \in \text{RM}(r-1, m-1)\} \quad (4.29)$$

with generator matrix

$$G_{\text{RM}}(r, m) = \begin{bmatrix} G_{\text{RM}}(r, m-1) & G_{\text{RM}}(r, m-1) \\ 0 & G_{\text{RM}}(r-1, m-1) \end{bmatrix}. \quad (4.30)$$

The matrix of (4.30) shows that a RM code can be constructed recursively from short RM codes by a sequence of $|\mathbf{u}|\mathbf{u} + \mathbf{v}$ -constructions. For example, the r th-order RM code $\text{RM}(r, m)$ of length 2^m can be constructed from RM codes $\text{RM}(r, m-2)$, $\text{RM}(r-1, m-2)$, and $\text{RM}(r-2, m-2)$ of length 2^{m-2} . The generator matrix in terms of component codes is given as follows:

$$G = \begin{bmatrix} G_{\text{RM}}(r, m-2) & G_{\text{RM}}(r, m-2) & G_{\text{RM}}(r, m-2) & G_{\text{RM}}(r, m-2) \\ 0 & G_{\text{RM}}(r-1, m-2) & 0 & G_{\text{RM}}(r-1, m-2) \\ 0 & 0 & G_{\text{RM}}(r-1, m-2) & G_{\text{RM}}(r-1, m-2) \\ 0 & 0 & 0 & G_{\text{RM}}(r-2, m-2) \end{bmatrix}. \quad (4.31)$$

The recursive structure of RM codes is very useful in analyzing and constructing their trellises [15, 16]. This structure also allows us to devise multistage soft-decision decoding schemes for RM codes that achieve good error performance with reduced decoding complexity. This topic will also be discussed in a later chapter.

Consider a Boolean function $f(X_1, X_2, \dots, X_m)$ of m variables, X_1, X_2, \dots, X_m , that take values of 0 or 1 [12, 14]. For each combination of values of X_1, X_2, \dots , and X_m , the function f takes a truth value of either 0 or 1. For the 2^m combinations of values of X_1, X_2, \dots, X_m , the truth values of f form a 2^m -tuple over $GF(2)$.

For a nonnegative integer l less than 2^m , let $(b_{l1}, b_{l2}, \dots, b_{lm})$ be the standard binary representation of l , such that $l = b_{l1} + b_{l2}2 + b_{l3}2^2 + \dots + b_{lm}2^{m-1}$. For a

given Boolean function $f(X_1, X_2, \dots, X_m)$, we form the following 2^m -tuple (truth vector):

$$\mathbf{v}(f) = (v_0, v_1, \dots, v_l, \dots, v_{2^m-1}) \quad (4.32)$$

where

$$v_l \stackrel{\Delta}{=} f(b_{l1}, b_{l2}, \dots, b_{lm}) \quad (4.33)$$

and $(b_{l1}, b_{l2}, \dots, b_{lm})$ is the standard binary representation of the index integer l . We say that the Boolean function $f(X_1, X_2, \dots, X_m)$ represents the vector \mathbf{v} . We use the notation $\mathbf{v}(f)$ for the vector represented by $f(X_1, X_2, \dots, X_m)$. For $1 \leq i \leq m$, consider the Boolean function

$$f(X_1, X_2, \dots, X_m) = X_i. \quad (4.34)$$

It is easy to see that this Boolean function represents the vector \mathbf{v}_i defined by (4.4). For $1 \leq i, j \leq m$, the function

$$f(X_1, X_2, \dots, X_m) = X_i X_j \quad (4.35)$$

represents the logic product of \mathbf{v}_i and \mathbf{v}_j , represented by $g(X_1, X_2, \dots, X_m) = X_i$ and $h(X_1, X_2, \dots, X_m) = X_j$, respectively. For $1 \leq i_1 < i_2 < \dots < i_r \leq m$, the Boolean function

$$f(X_1, X_2, \dots, X_m) = X_{i_1} X_{i_2} \cdots X_{i_r} \quad (4.36)$$

represents the logic product of $\mathbf{v}_{i_1}, \mathbf{v}_{i_2}, \dots$, and \mathbf{v}_{i_r} . Therefore, the generator vectors of the r th-order RM code of length $n = 2^m$ (the rows in $G_{\text{RM}}(r, m)$) are represented by the Boolean functions in the following set:

$$B(r, m) = \{1, X_1, X_2, \dots, X_m, X_1 X_2, X_1 X_3, \dots, X_{m-1} X_m, \dots \text{ up to all products of } r \text{ variables}\}. \quad (4.37)$$

Let $P(r, m)$ denote the set of all Boolean functions (or polynomials) of degree r or less with m variables. Then, $\text{RM}(r, m)$ is given by the following set of vectors [12]:

$$\text{RM}(r, m) = \{\mathbf{v}(f) : f \in P(r, m)\}. \quad (4.38)$$

The Boolean representation is very useful in studying the weight distribution of RM codes [18, 20].

4.5 THE SQUARING CONSTRUCTION OF CODES

Consider a binary (n, k) linear code C with generator matrix G . For $0 \leq k_1 \leq k$, let C_1 be an (n, k_1) linear subcode of C that is spanned by k_1 rows of G . Partition C into 2^{k-k_1} cosets of C_1 . This partition of C with respect to C_1 is denoted by C/C_1 . As shown in Section 3.5, each coset of C_1 is of the following form:

$$\mathbf{v}_l \oplus C_1 = \{\mathbf{v}_l + \mathbf{u} : \mathbf{u} \in C_1\} \quad (4.39)$$

with $1 \leq l \leq 2^{k-k_1}$, where for $\mathbf{v}_l \neq \mathbf{0}$, \mathbf{v}_l is in C but not in C_1 , and for $\mathbf{v}_l = \mathbf{0}$, the coset $\mathbf{0} \oplus C_1$ is just the subcode C_1 itself. The codeword \mathbf{v}_l is called the *leader* (or *representative*) of the coset $\mathbf{v}_l \oplus C_1$. We also showed in Section 3.5 that any codeword

in a coset can be used as the coset representative without changing the composition of the coset. The all-zero codeword $\mathbf{0}$ is always used as the representative for C_1 . The set of representatives for the cosets in the partition C/C_1 is denoted by $[C/C_1]$, which is called the *coset representative space* for the partition C/C_1 . Because all the cosets in C/C_1 are disjoint, C_1 and $[C/C_1]$ have only the all-zero codeword $\mathbf{0}$ in common; that is, $C_1 \cap [C/C_1] = \{\mathbf{0}\}$. Then, we can express C as the sum of the coset representatives in $[C/C_1]$ and the codewords in C_1 as follows:

$$C = [C/C_1] \oplus C_1 \stackrel{\Delta}{=} \{\mathbf{v} + \mathbf{u} : \mathbf{v} \in [C/C_1], \mathbf{u} \in C_1\}. \quad (4.40)$$

The preceding sum is called the *direct-sum* of $[C/C_1]$ and C_1 .

Let G_1 be the subset of k_1 rows of the generator matrix G that generates the linear subcode C_1 . Then, the 2^{k-k_1} codewords generated by the $k - k_1$ rows in the set $G \setminus G_1$ can be used as the representatives for the cosets in C/C_1 . These 2^{k-k_1} codewords form an $(n, k - k_1)$ linear subcode of C .

Let C_2 be an (n, k_2) linear subcode of C_1 with $0 \leq k_2 \leq k_1$. We can further partition each coset $\mathbf{v}_l \oplus C_1$ in the partition C/C_1 based on C_2 into $2^{k_1-k_2}$ cosets of C_2 ; each coset consists of the following codewords in C :

$$\mathbf{v}_l \oplus (\mathbf{w}_q \oplus C_2) \stackrel{\Delta}{=} \{\mathbf{v}_l + \mathbf{w}_q + \mathbf{u} : \mathbf{u} \in C_2\} \quad (4.41)$$

with $1 \leq l \leq 2^{k-k_1}$ and $1 \leq q \leq 2^{k_1-k_2}$, where for $\mathbf{w}_q \neq \mathbf{0}$, \mathbf{w}_q is a codeword in C_1 but not in C_2 . We denote this partition $C/C_1/C_2$. This partition consists of 2^{k-k_2} cosets of C_2 . Now, we can express C as the following direct-sum:

$$C = [C/C_1] \oplus [C_1/C_2] \oplus C_2. \quad (4.42)$$

Let C_1, C_2, \dots, C_m be a sequence of linear subcodes of C with dimensions k_1, k_2, \dots, k_m , respectively, such that

$$C \supseteq C_1 \supseteq C_2 \supseteq \cdots \supseteq C_m \quad (4.43)$$

and

$$k \geq k_1 \geq k_2 \geq \cdots \geq k_m \geq 0. \quad (4.44)$$

Then, we can form a chain of partitions,

$$C/C_1, C/C_1/C_2, \dots, C/C_1/C_2/\cdots/C_m, \quad (4.45)$$

and can express C as the following direct-sum:

$$C = [C/C_1] \oplus [C_1/C_2] \oplus \cdots \oplus [C_{m-1}/C_m] \oplus C_m. \quad (4.46)$$

We now present another method for constructing long codes from a sequence of subcodes of a given short code. This method is known as the *squaring construction* [15].

Let C_0 be a binary (n, k_0) linear block code with minimum Hamming distance d_0 . Let C_1, C_2, \dots, C_m be a sequence of subcodes of C_0 such that

$$C_0 \supset C_1 \supset C_2 \supset \cdots \supset C_m. \quad (4.47)$$

For $0 \leq i \leq m$, let G_i , k_i , and d_i be the generator matrix, the dimension, and the minimum distance of the subcode C_i , respectively. We form a chain of partitions as follows:

$$C_0/C_1, C_0/C_1/C_2, \dots, C_0/C_1/\cdots/C_m. \quad (4.48)$$

For $0 \leq i < m$, let $G_{i/i+1}$ denote the generator matrix for the coset representative space $[C_i/C_{i+1}]$. The rank of $G_{i/i+1}$ is

$$\text{Rank}(G_{i/i+1}) = \text{Rank}(G_i) - \text{Rank}(G_{i+1}). \quad (4.49)$$

Without loss of generality, we assume that $G_0 \supset G_1 \supset G_2 \supset \cdots \supset G_m$. Then, for $0 \leq i < m$,

$$G_{i/i+1} = G_i \setminus G_{i+1}. \quad (4.50)$$

One-level squaring construction is based on C_1 and the partition C_0/C_1 . Let $\mathbf{a} = (a_0, a_1, \dots, a_{n-1})$ and $\mathbf{b} = (b_0, b_1, \dots, b_{n-1})$ be two binary n -tuples, and let (\mathbf{a}, \mathbf{b}) denote the $2n$ -tuple $(a_0, a_1, \dots, a_{n-1}, b_0, b_1, \dots, b_{n-1})$. We form the following set of $2n$ -tuples:

$$|C_0/C_1|^2 \stackrel{\Delta}{=} \{(\mathbf{a} + \mathbf{x}, \mathbf{b} + \mathbf{x}) : \mathbf{a}, \mathbf{b} \in C_1 \text{ and } \mathbf{x} \in [C_0/C_1]\}. \quad (4.51)$$

Then, $|C_0/C_1|^2$ is a $(2n, k_0 + k_1)$ linear block code with minimum Hamming distance

$$D_1 \stackrel{\Delta}{=} \min\{2d_0, d_1\}. \quad (4.52)$$

The generator matrix for $|C_0/C_1|^2$ is given by

$$G = \begin{bmatrix} G_1 & 0 \\ 0 & G_1 \\ G_{0/1} & G_{0/1} \end{bmatrix}. \quad (4.53)$$

Let M_1 and M_2 be two matrices with the same number of columns. The matrix

$$M = \begin{bmatrix} M_1 \\ M_2 \end{bmatrix}$$

is called the direct-sum of M_1 and M_2 , denoted by $M = M_1 \oplus M_2$. Then, we can express the generator matrix for the one-level squaring construction code $|C_0/C_1|^2$ in the following form:

$$G = I_2 \otimes G_1 \oplus (1, 1) \otimes G_{0/1}, \quad (4.54)$$

where \otimes denotes the Kronecker product, \oplus the direct-sum, I_2 the identity matrix of dimension 2,

$$I_2 \otimes G_1 = \begin{bmatrix} G_1 & 0 \\ 0 & G_1 \end{bmatrix}, \quad (4.55)$$

and

$$(1, 1) \otimes G_{0/1} = [G_{0/1} \ G_{0/1}]. \quad (4.56)$$

Now, we extend the one-level squaring construction to a two-level squaring construction. First, we form two codes, $U \triangleq |C_0/C_1|^2$ and $V \triangleq |C_1/C_2|^2$, using one-level squaring construction. It is easy to see that V is a subcode of U . The two-level squaring construction based on the partitions C_0/C_1 , C_1/C_2 , and $C_0/C_1/C_2$ gives the following code:

$$\begin{aligned} |C_0/C_1/C_2|^4 &\triangleq \{(\mathbf{a} + \mathbf{x}, \mathbf{b} + \mathbf{x}) : \mathbf{a}, \mathbf{b} \in V \text{ and } \mathbf{x} \in [U/V]\} \\ &= \{(\mathbf{a} + \mathbf{x}, \mathbf{b} + \mathbf{x}) : \mathbf{a}, \mathbf{b} \in |C_1/C_2|^2\} \\ &\quad \text{and } \mathbf{x} \in [|C_0/C_1|^2 / |C_1/C_2|^2]\}, \end{aligned} \quad (4.57)$$

which is simply the code obtained by one-level squaring construction based on V and $[U/V]$. This code is a $(4n, k_0 + 2k_1 + k_2)$ linear block code with minimum Hamming distance

$$D_2 \triangleq \min\{4d_0, 2d_1, d_2\}. \quad (4.58)$$

Let G_U , G_V , and $G_{U/V}$ denote the generator matrices for U , V , and $[U/V]$, respectively. Then, the generator matrix for $|U/V|^2$ is

$$G = \begin{bmatrix} G_V & 0 \\ 0 & G_V \\ G_{U/V} & G_{U/V} \end{bmatrix}. \quad (4.59)$$

We put G_V and G_U in the form of (4.53) and note that $G_{U/V} = G_U \setminus G_V$. Then, we can put the generator matrix G of $|C_0/C_1/C_2|^4 = |U/V|^2$ given by (4.59) in the following form:

$$G = \begin{bmatrix} G_2 & 0 & 0 & 0 \\ 0 & G_2 & 0 & 0 \\ 0 & 0 & G_2 & 0 \\ 0 & 0 & 0 & G_2 \\ G_{0/1} & G_{0/1} & G_{0/1} & G_{0/1} \\ G_{1/2} & G_{1/2} & G_{1/2} & G_{1/2} \\ 0 & 0 & G_{1/2} & G_{1/2} \\ 0 & G_{1/2} & 0 & G_{1/2} \end{bmatrix}. \quad (4.60)$$

We can express this matrix in the following compact form:

$$G = I_4 \otimes G_2 \oplus (1111) \otimes G_{0/1} \oplus \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \otimes G_{1/2}. \quad (4.61)$$

Note that

$$(1111) \text{ and } \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

are the generator matrices of the zeroth- and first-order RM codes of length 4.

Higher-level squaring construction can be carried out recursively in a similar manner. For $m \geq 2$, let

$$U_m \triangleq |C_0/C_1/\cdots/C_{m-1}|^{2^{m-1}}$$

and

$$V_m \triangleq |C_1/C_2/\cdots/C_m|^{2^{m-1}}$$

denote the two codes obtained by $(m-1)$ -level squaring construction. The code obtained by m -level squaring construction is given by [15]:

$$|C_0/C_1/\cdots/C_m|^{2^m} \triangleq \{(\mathbf{a} + \mathbf{x}, \mathbf{b} + \mathbf{x}) : \mathbf{a}, \mathbf{b} \in V_m \text{ and } \mathbf{x} \in [U_m/V_m]\}. \quad (4.62)$$

The generator matrix is given by [15, 16]:

$$G = I_{2^m} \otimes G_m \oplus \sum_{0 \leq r < m} G_{\text{RM}}(r, m) \otimes G_{r/r+1}, \quad (4.63)$$

where I_{2^m} denotes the identity matrix of dimension 2^m , and $G_{\text{RM}}(r, m)$ is the generator matrix of the r th-order RM code, $\text{RM}(r, m)$, of length 2^m .

RM codes are good examples of the squaring construction. Long RM codes can be constructed from short RM codes iteratively using the squaring construction [15, 16]. From the construction of RM codes given in Section 4.2, we find that for $0 < i \leq r$,

$$\text{RM}(r, m) \supset \text{RM}(r-1, m) \supset \cdots \supset \text{RM}(r-i, m). \quad (4.64)$$

Consider the RM code $\text{RM}(r, m)$. As shown in Section 4.4, this code can be obtained from $\text{RM}(r, m-1)$ and $\text{RM}(r-1, m-1)$ codes using the $|\mathbf{u}|\mathbf{u} + \mathbf{v}|$ -construction. It follows from (4.30) that the generator matrix for the $\text{RM}(r, m)$ code can be expressed as follows:

$$G_{\text{RM}}(r, m) = \begin{bmatrix} G_{\text{RM}}(r, m-1) & G_{\text{RM}}(r, m-1) \\ 0 & G_{\text{RM}}(r-1, m-1) \end{bmatrix}. \quad (4.65)$$

We define

$$\Delta_{\text{RM}}(r/r-1, m-1) \triangleq G_{\text{RM}}(r, m-1) \setminus G_{\text{RM}}(r-1, m-1). \quad (4.66)$$

Note that $\Delta_{\text{RM}}(r/r-1, m-1)$ consists of those rows in $G_{\text{RM}}(r, m-1)$ but not in $G_{\text{RM}}(r-1, m-1)$ and it spans the coset representative space $[\text{RM}(r, m-1)/\text{RM}(r-1, m-1)]$. Now, we can put $G_{\text{RM}}(r, m-1)$ in the following form:

$$G_{\text{RM}}(r, m-1) = \begin{bmatrix} G_{\text{RM}}(r-1, m-1) \\ \Delta_{\text{RM}}(r/r-1, m-1) \end{bmatrix}. \quad (4.67)$$

Replacing $G_{\text{RM}}(r, m-1)$ in (4.65) with the expression of (4.67) and performing row operations, we put $G_{\text{RM}}(r, m)$ in the following form:

$$G_{\text{RM}}(r, m) = \begin{bmatrix} G_{\text{RM}}(r-1, m-1) & 0 \\ 0 & G_{\text{RM}}(r-1, m-1) \\ \Delta_{\text{RM}}(r/r-1, m-1) & \Delta_{\text{RM}}(r/r-1, m-1) \end{bmatrix}. \quad (4.68)$$

This is exactly the generator matrix form of one-level squaring construction. Therefore, the r th-order RM code of length 2^m can be constructed from the r th-order and $(r-1)$ th order RM codes of length 2^{m-1} ; that is,

$$\text{RM}(r, m) = |\text{RM}(r, m-1)/\text{RM}(r-1, m-1)|^2. \quad (4.69)$$

Because

$$\text{RM}(r, m - 1) = |\text{RM}(r, m - 2)/\text{RM}(r - 1, m - 2)|^2$$

and

$$\text{RM}(r - 1, m - 1) = |\text{RM}(r - 1, m - 2)/\text{RM}(r - 2, m - 2)|^2,$$

then we can construct $\text{RM}(r, m)$ from $\text{RM}(r, m - 2)$, $\text{RM}(r - 1, m - 2)$, and $\text{RM}(r - 2, m - 2)$ using two-level squaring construction; that is,

$$\text{RM}(r, m) = |\text{RM}(r, m - 2)/\text{RM}(r - 1, m - 2)/\text{RM}(r - 2, m - 2)|^{2^2}. \quad (4.70)$$

Repeating the preceding process, we find that for $1 \leq \mu \leq r$, we can express the $\text{RM}(r, m)$ code as a μ -level squaring construction code as follows:

$$\text{RM}(r, m) = |\text{RM}(r, m - \mu)/\text{RM}(r - 1, m - \mu)/\cdots/\text{RM}(r - \mu, m - \mu)|^{2^\mu}. \quad (4.71)$$

A problem related to the construction of codes from component codes is code *decomposition*. A code is said to be *decomposable* if it can be expressed in terms of component codes. A code is said to be μ -level decomposable if it can be expressed as a μ -level squaring construction code from a sequence of subcodes of a given code, as shown in (4.62). From (4.71) we see that a RM code is μ -level decomposable.

A μ -level decomposable code can be decoded in multiple stages: component codes are decoded sequentially one at a time, and decoded information is passed from one stage to the next stage. This multistage decoding provides a good trade-off between error performance and decoding complexity, especially for long codes.

RM codes also can be constructed from Euclidean geometry, which is discussed in Chapter 8. This construction reveals more algebraic and geometric structures of these codes, especially the structures and the number of minimum-weight codewords. There is a one-to-one correspondence between a minimum-weight codeword of the r th-order RM code, $\text{RM}(r, m)$, and an $(m - r)$ -dimensional flat in m -dimensional Euclidean geometry, $\text{EG}(m, 2)$, over $GF(2)$. This correspondence gives the number of minimum-weight codewords of the $\text{RM}(r, m)$ code [12, 17]

$$A_{2^{m-r}} = 2^r \prod_{i=0}^{m-r-1} \left(\frac{2^{m-i} - 1}{2^{m-r-i} - 1} \right). \quad (4.72)$$

In fact, these minimum-weight codewords span (or generate) the code; that is the linear combinations of these minimum-weight codewords produce all the codewords of the $\text{RM}(r, m)$ code.

The weight distribution of several subclasses of RM codes and all RM codes of lengths up to 512 have been enumerated [12, 18–21]. The first-order RM code, $\text{RM}(1, m)$, has only three weights, 1, 2^{m-1} , and 2^m . The number of codewords of these weights are

$$\begin{aligned} A_0 &= A_{2^m} = 1, \\ A_{2^{m-1}} &= 2^{m+1} - 2. \end{aligned} \quad (4.73)$$

The second-order RM code, $\text{RM}(2, m)$ has the following weight distribution:

$$\begin{aligned} A_0 &= A_{2^m} = 1, \\ A_{2^{m-1} \pm 2^{m-1-l}} &= 2^{l(l+1)} \frac{\prod_{i=m-2l+1}^m (2^i - 1)}{\prod_{i=1}^l (2^{2i} - 1)}, \quad \text{for } 1 \leq l \leq \lfloor \frac{m}{2} \rfloor \quad (4.74) \\ A_{2^{m-1}} &= 2^{(m^2+m+2)/2} - 2 - 2 \sum_{l=1}^{\lfloor \frac{m}{2} \rfloor} 2^{l(l+1)} \frac{\prod_{i=m-2l+1}^m (2^i - 1)}{\prod_{i=1}^l (2^{2i} - 1)}. \end{aligned}$$

Because the $(m - r - 1)$ th-order RM code, $\text{RM}(m - r - 1, m)$, is the dual code of the r th-order RM code, $\text{RM}(r, m)$, the weight distributions of the $\text{RM}(m - 2, m)$ and $\text{RM}(m - 3, m)$ codes can be derived from (4.73), (4.74), and the MacWilliams identity of (3.32).

EXAMPLE 4.6

The weight distribution of the $(32, 16)$ RM code, $\text{RM}(2, 5)$, is

i	0	8	12	16	20	24	32
A_i	1	620	13888	36518	13888	620	1

This code is a self-dual code.

RM codes form a remarkable class of linear block codes. Their rich structural properties make them very easy to decode by either hard- or soft-decision decoding. Various soft-decision decoding algorithms for these codes have been devised, and some will be discussed in later chapters. Other classes of codes are more powerful than RM codes—for the same minimum distance, these codes have higher rates; however, the low decoding complexity of RM codes makes them very attractive in practical applications. In fact, in terms of both error performance and decoding complexity, RM codes often outperform their corresponding more powerful codes.

The $(m - 2)$ th-order RM code of length 2^m is actually the distance-4 extended Hamming code obtained by adding an overall parity bit to the Hamming code of length $2^m - 1$.

4.6 THE (24, 12) GOLAY CODE

Besides the Hamming codes, the only other nontrivial binary perfect code is the $(23, 12)$ Golay code constructed by M. J. E. Golay in 1949 [3]. This code has a minimum distance of 7 and is capable of correcting any combination of three or fewer random errors in a block of 23 digits. The code has abundant and beautiful algebraic structure, and it has become a subject of study by many coding theorists and mathematicians; many research papers have been published on its structure and decoding. The Golay code is the most extensively studied single code. In addition to having beautiful structure, this code has been used in many real communication systems for error control. This code in its cyclic form will be studied in Chapter 5.

The $(23, 12)$ Golay code can be extended by adding an overall parity-check bit to each codeword. This extension results in a $(24, 12)$ code with a minimum distance

of 8. This code is capable of correcting all error patterns of three or fewer errors and detecting all error patterns of four errors. It is not a perfect code anymore; however, it has many interesting structural properties and has been widely used for error control in many communication systems, especially in the U.S. space program. It served as the primary *Voyager* error-control system, providing clear color pictures of Jupiter and Saturn between 1979 and 1981.

In this section we study the (24, 12) Golay code and its decoding. A generator matrix in systematic form for this code is as follows [12, 23, 24]:

$$\mathbf{G} = [\mathbf{P} \quad \mathbf{I}_{12}],$$

where \mathbf{I}_{12} is the identity matrix of dimension 12 and

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}. \quad (4.75)$$

The \mathbf{P} matrix has the following properties: (1) it is symmetrical with respect to its diagonal; (2) the i th column is the transpose of the i th row; (3) $\mathbf{P} \cdot \mathbf{P}^T = \mathbf{I}_{12}$, where \mathbf{P}^T is the transpose of \mathbf{P} ; and (4) the submatrix obtained by deleting the last row and last column is formed by cyclically shifting the first row to the left 11 times (or cyclically shifting the first column upward 11 times). It follows from the second property that

$$\mathbf{P}^T = \mathbf{P}.$$

Consequently, the parity-check matrix in systematic form for the (24, 12) extended Golay code is given by

$$\begin{aligned} \mathbf{H} &= [\mathbf{I}_{12} \quad \mathbf{P}^T] \\ &= [\mathbf{I}_{12} \quad \mathbf{P}]. \end{aligned} \quad (4.76)$$

It can be proved that the code is self-dual [see Problem 4.18].

A simple decoding algorithm for the (24, 12) Golay code can be devised using the properties of the \mathbf{P} matrix [23]. For $0 \leq i \leq 11$, let \mathbf{p}_i denote the i th row of \mathbf{P} and $\mathbf{u}^{(i)}$ the 12-tuple in which only the i th component is nonzero. For example, $\mathbf{u}^{(5)} = (0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0)$. We readily see that

$$\mathbf{p}_i = \mathbf{u}^{(i)} \cdot \mathbf{P}. \quad (4.77)$$

Let $\mathbf{e} = (\mathbf{x}, \mathbf{y})$ be an error vector, where \mathbf{x} and \mathbf{y} are binary 12-tuples. Suppose a codeword \mathbf{v} is transmitted, and a correctable error pattern $\mathbf{e} = (\mathbf{x}, \mathbf{y})$ occurs. Then,

the received vector is $\mathbf{r} = \mathbf{v} + \mathbf{e}$. The syndrome of \mathbf{r} is

$$\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T = (\mathbf{v} + \mathbf{e}) \cdot \mathbf{H}^T = \mathbf{e} \cdot \mathbf{H}^T.$$

It follows from (4.76) and $\mathbf{P}^T = \mathbf{P}$ that

$$\begin{aligned}\mathbf{s} &= (\mathbf{x}, \mathbf{y}) \cdot \begin{bmatrix} \mathbf{I}_{12} \\ \mathbf{P} \end{bmatrix} \\ &= \mathbf{x} \cdot \mathbf{I}_{12} + \mathbf{y} \cdot \mathbf{P} \\ &= \mathbf{x} + \mathbf{y} \cdot \mathbf{P}.\end{aligned}\tag{4.78}$$

Using the property $\mathbf{P} \cdot \mathbf{P}^T = \mathbf{I}_{12}$, we can express \mathbf{y} in terms of \mathbf{x} , \mathbf{s} , and \mathbf{P} as follows:

$$\mathbf{y} = (\mathbf{s} + \mathbf{x}) \cdot \mathbf{P}.\tag{4.79}$$

In the following, we first show that a correctable error pattern for the (24, 12) Golay code can be expressed in terms of \mathbf{P} , \mathbf{p}_i , $\mathbf{u}^{(i)}$, and \mathbf{s} . We then present a decoding algorithm for the code.

For any correctable error pattern with weight $w(\mathbf{e}) \leq 3$, we have the following four possibilities:

- (1) $w(\mathbf{x}) \leq 3$ and $w(\mathbf{y}) = 0$,
- (2) $w(\mathbf{x}) \leq 2$ and $w(\mathbf{y}) = 1$,
- (3) $w(\mathbf{x}) \leq 1$ and $w(\mathbf{y}) = 2$,
- (4) $w(\mathbf{x}) = 0$ and $w(\mathbf{y}) = 3$.

These four possibilities define four different types of correctable error patterns. For $0 \leq j \leq 3$, let $\mathbf{e}^{(j)} = (\mathbf{x}, \mathbf{y})$, for which $w(\mathbf{y}) = j$, and $w(\mathbf{x}) \leq 3 - j$. Suppose $\mathbf{e} = \mathbf{e}^{(0)}$. It follows from (4.78) that $\mathbf{s} = \mathbf{x}$ and $w(\mathbf{s}) = w(\mathbf{x}) \leq 3$. In this case,

$$\mathbf{e} = (\mathbf{s}, \mathbf{0}),$$

where $\mathbf{0}$ is the all-zero 12-tuple. Suppose $\mathbf{e} = \mathbf{e}^{(1)}$ and $\mathbf{y} = \mathbf{u}^{(i)}$. Then, it follows from (4.78) that

$$\mathbf{s} = \mathbf{x} + \mathbf{u}^{(i)} \cdot \mathbf{P} = \mathbf{x} + \mathbf{p}_i.$$

Hence, $\mathbf{x} = \mathbf{s} + \mathbf{p}_i$, and $w(\mathbf{s} + \mathbf{p}_i) = w(\mathbf{x}) \leq 2$. In this case,

$$\mathbf{e} = (\mathbf{s} + \mathbf{p}_i, \mathbf{u}^{(i)}).$$

Suppose $\mathbf{e} = \mathbf{e}^{(2)}$ or $\mathbf{e}^{(3)}$, and $w(\mathbf{x}) = 0$. It follows from (4.79) that

$$\mathbf{y} = \mathbf{s} \cdot \mathbf{P},$$

and $w(\mathbf{s} \cdot \mathbf{P}) = w(\mathbf{y}) = 2$ or 3. For this case, we can express \mathbf{e} as follows:

$$\mathbf{e} = (\mathbf{0}, \mathbf{s} \cdot \mathbf{P}).$$

Now, suppose $\mathbf{e} = \mathbf{e}^{(2)}$, and $w(\mathbf{x}) = 1$. If the nonzero component of \mathbf{x} is at the i th position, then $\mathbf{x} = \mathbf{u}^{(i)}$. It follows from (4.79) that

$$\begin{aligned}\mathbf{y} &= (\mathbf{s} + \mathbf{u}^{(i)}) \cdot \mathbf{P} \\ &= \mathbf{s} \cdot \mathbf{P} + \mathbf{u}^{(i)} \cdot \mathbf{P} \\ &= \mathbf{s} \cdot \mathbf{P} + \mathbf{p}_i\end{aligned}$$

and $w(\mathbf{s} \cdot \mathbf{P} + \mathbf{p}_i) = w(\mathbf{y}) = 2$. Consequently, we can express \mathbf{e} as follows:

$$\mathbf{e} = (\mathbf{u}^{(i)}, \mathbf{s} \cdot \mathbf{P} + \mathbf{p}_i).$$

A decoding algorithm can be devised for the (24, 12) Golay code based on the preceding analysis and expressions of correctable error patterns. The decoding consists of the following steps:

- Step 1.** Compute the syndrome \mathbf{s} of the received sequence \mathbf{r} .
- Step 2.** If $w(\mathbf{s}) \leq 3$, then set $\mathbf{e} = (\mathbf{s}, \mathbf{0})$ and go to step 8.
- Step 3.** If $w(\mathbf{s} + \mathbf{p}_i) \leq 2$ for some row \mathbf{p}_i in \mathbf{P} , then set $\mathbf{e} = (\mathbf{s} + \mathbf{p}_i, \mathbf{u}^{(i)})$ and go to step 8.
- Step 4.** Compute $\mathbf{s} \cdot \mathbf{P}$.
- Step 5.** If $w(\mathbf{s} \cdot \mathbf{P}) = 2$ or 3, then set $\mathbf{e} = (\mathbf{0}, \mathbf{s} \cdot \mathbf{P})$ and go to step 8.
- Step 6.** If $w(\mathbf{s} \cdot \mathbf{P} + \mathbf{p}_i) = 2$ for some row \mathbf{p}_i in \mathbf{P} , then set $\mathbf{e} = (\mathbf{u}^{(i)}, \mathbf{s} \cdot \mathbf{P} + \mathbf{p}_i)$ and go to step 8.
- Step 7.** If the syndrome does not correspond to a correctable error pattern, stop the decoding process, or request a retransmission. (This represents a decoding failure.)
- Step 8.** Set the decoded codeword $\mathbf{v}^* = \mathbf{r} + \mathbf{e}$ and stop.

EXAMPLE 4.7

Suppose the (24, 12) Golay code is used for error control. Let $\mathbf{r} = (1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1)$ be the received sequence. To decode \mathbf{r} , we first compute the syndrome \mathbf{s} of \mathbf{r} :

$$\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T = (1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0).$$

Because $w(\mathbf{s}) > 3$, we go to decoding step 3. We find that

$$\begin{aligned}\mathbf{s} + \mathbf{p}_{11} &= (1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0) + (1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0) \\ &= (0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0),\end{aligned}$$

and $w(\mathbf{s} + \mathbf{p}_{11}) = 2$. So we set

$$\begin{aligned}\mathbf{e} &= (\mathbf{s} + \mathbf{p}_{11}, \mathbf{u}^{(11)}) \\ &= (0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1)\end{aligned}$$

and decode \mathbf{r} into

$$\begin{aligned}\mathbf{v}^* &= \mathbf{r} + \mathbf{e} \\ &= (1\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0).\end{aligned}$$

4.7 PRODUCT CODES

Besides the $|\mathbf{u}| \mathbf{u} + \mathbf{v}|$ and squaring constructions of codes, another technique for constructing long, powerful codes from short component codes is the *product coding* technique.

Let C_1 be an (n_1, k_1) linear code, and let C_2 be an (n_2, k_2) linear code. Then, an $(n_1 n_2, k_1 k_2)$ linear code can be formed such that each codeword is a rectangular array of n_1 columns and n_2 rows in which every row is a codeword in C_1 , and every column is a codeword in C_2 , as shown in Figure 4.3. This two-dimensional code is called the *direct product* (or simply the *product*) of C_1 , and C_2 [25]. The $k_1 k_2$ digits in the upper right corner of the array are information symbols. The digits in the upper left corner of this array are computed from the parity-check rules for C_1 on rows, and the digits in the lower right corner are computed from the parity-check rules for C_2 on columns. Now, should we compute the check digits in the lower left corner by using the parity-check rules for C_2 on columns or the parity-check rules for C_1 on rows? It turns out that either way yields the same $(n_1 - k_1) \times (n_2 - k_2)$ check digits (see Problem 4.21), and it is possible to have all row codewords in C_1 and all column codewords in C_2 simultaneously.

The product code $C_1 \times C_2$ is encoded in two steps. A message of $k_1 k_2$ information symbols is first arranged as shown in the upper right corner of Figure 4.3. At the first step of encoding, each row of the information array is encoded into a codeword in C_1 . This row encoding results in an array of k_2 rows and n_1 columns, as shown in the upper part of Figure 4.3. At the second step of encoding each of the n_1 columns of the array formed at the first encoding step is encoded into a codeword in C_2 . This results in a code array of n_2 rows and n_1 columns, as shown in Figure 4.3. This code array also can be formed by first performing the column encoding and then the row encoding. Transmission can be carried out either column by column or row by row.

If code C_1 has minimum weight d_1 and code C_2 has minimum weight d_2 , the minimum weight of the product code is exactly $d_1 d_2$. A minimum-weight codeword in the product code is formed by (1) choosing a minimum-weight codeword in C_1 and a minimum-weight codeword in C_2 and (2) forming an array in which all columns corresponding to zeros in the codeword from C_1 are zeros, and all columns corresponding to ones in the codeword from C_1 are the minimum-weight codeword chosen from C_2 .

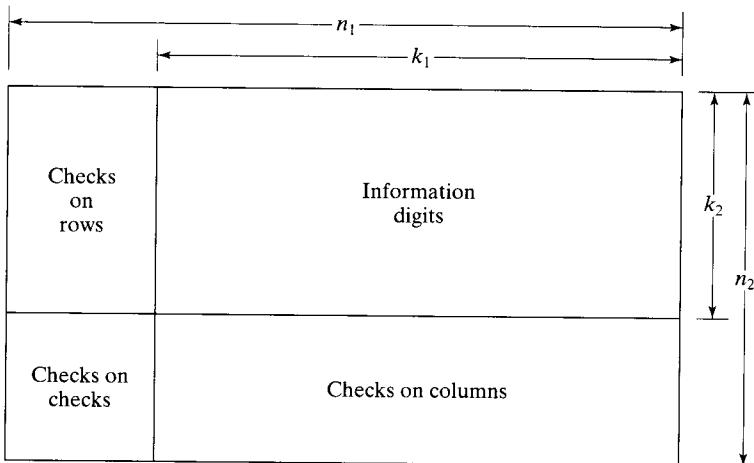


FIGURE 4.3: Code array for the product code $C_1 \times C_2$.

It is not easy to characterize the correctable error patterns for the product code; this depends on how the correction is done. One method involves using a two-step decoding. The decoding is first performed on rows and then on columns. In this case a pattern will be correctable if and only if the uncorrectable patterns on rows after row correction leave correctable patterns on the columns. It generally improves the correction to decode by rows, then columns, then columns and rows again. This method, of course, increases the decoding delay. This type of decoding is called *iterative decoding* [25].

The product code is capable of correcting any combination of $[(d_1 d_2 - 1)/2]$ errors, but the method described will not achieve this. For example, consider the product code of two Hamming single-error-correcting codes. The minimum distance of each is 3, so the minimum distance of the product is 9. A pattern of four errors at the corners of a rectangle gives two errors in each of the two rows and two columns and is therefore not correctable by simple correction on rows and columns. Nevertheless, simple correction on rows and columns, although nonoptimum, can be very effective. The complexity of the two-step decoding is roughly the sum of the complexities of the two component code decodings.

EXAMPLE 4.8

Consider the product of the $(5, 4)$ SPC code C_1 with itself. The product code $C_1 \times C_1$ is a $(25, 16)$ linear block code C with a minimum distance of 4. Suppose the message $\mathbf{u} = (1\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1)$ is to be encoded. This message is read into a storage buffer and arranged into a 4×4 information array, as shown in Figure 4.4. The first four information symbols form the first row of the information array, the second four information symbols form the second row, and so on. At the first step of encoding, a single (even) parity-check symbol is added to each row of the information array. This results in a 4×5 array, as shown in Figure 4.4. In the second step of encoding a single (even) parity-check symbol is added to each of the five columns of the array, as shown in Figure 4.4. Suppose this code array is transmitted column by column. At the received end, the received sequence is rearranged into a 5×5 code array column by column, called the *received array*. Suppose a single error occurs at the intersection of a row and a column. The erroneous row and column containing this single error are indicated by parity-check failures, then the error is corrected by complementing the received symbol (i.e., 0 to 1, and 1 to 0) at the intersection. All the single-error patterns can be corrected in this manner. Checking the row and column parity failures cannot correct any double-error pattern, but it can detect all the double-error patterns. When a double-error pattern occurs, there

1	1	0	1	1
1	0	0	0	1
0	0	1	0	1
1	1	1	0	1
1	0	0	1	0

FIGURE 4.4: A code array of the product of the $(5, 4)$ SPC code with itself.

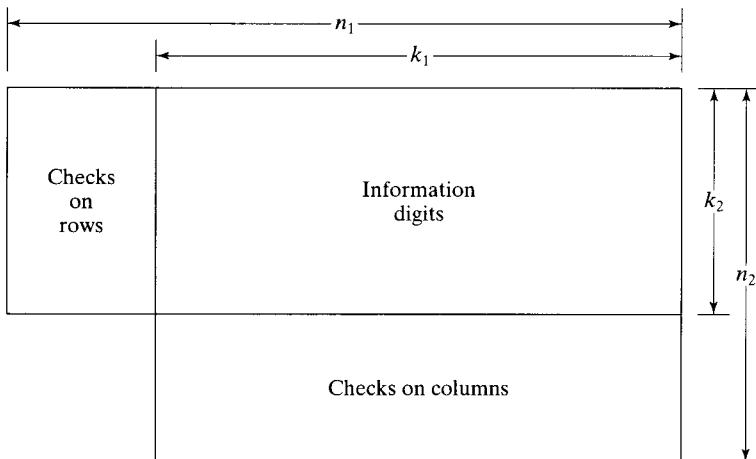


FIGURE 4.5: Incomplete product of two codes.

are three possible distributions of the two errors: (1) they are in the same row; (2) they are in the same column; or (3) they are in different rows and different columns. In the first case, there are two column parity failures but no row parity failure. Hence, errors are detected but they cannot be located. In the second case, there are two row parity failures but no column parity failure. Again, errors are detected but cannot be located. In the third case, there are two row parity failures and two column parity failures, so there are four intersecting locations. The two errors are situated at two opposite diagonal positions, but we cannot determine the positions.

In constructing a two-dimensional product code, if we do not form the $(n_1 - k_1) \times (n_2 - k_2)$ checks on checks in the lower left corner of Figure 4.3, we obtain an incomplete code array, as shown in Figure 4.5. This incomplete product of two codes results in a $(k_1 n_2 + k_2 n_1 - k_1 k_2, k_1 k_2)$ linear block code with a minimum distance of $d_1 + d_2 - 1$ (see Problem 4.22). The code has a higher rate but smaller minimum distance than the complete product code.

4.8 INTERLEAVED CODES

Given an (n, k) linear block code C , it is possible to construct a $(\lambda n, \lambda k)$ linear block code (i.e., a code λ times as long with λ times as many information symbols) by interleaving, that is, simply by arranging λ codewords in C into λ rows of a rectangular array and then transmitting the array column by column, as shown in Figure 4.6. The resulting code, denoted by C^λ , is called an *interleaved code*. The parameter λ is referred to as the *interleaving depth* (or *degree*). If the minimum distance of the base code C is d_{min} , the minimum distance of the interleaved code is also d_{min} .

The obvious way to implement an interleaved code is to set up the code array and operate on rows in encoding and decoding. In this way, a pattern of errors can be corrected for the whole array if and only if the pattern of errors in each row

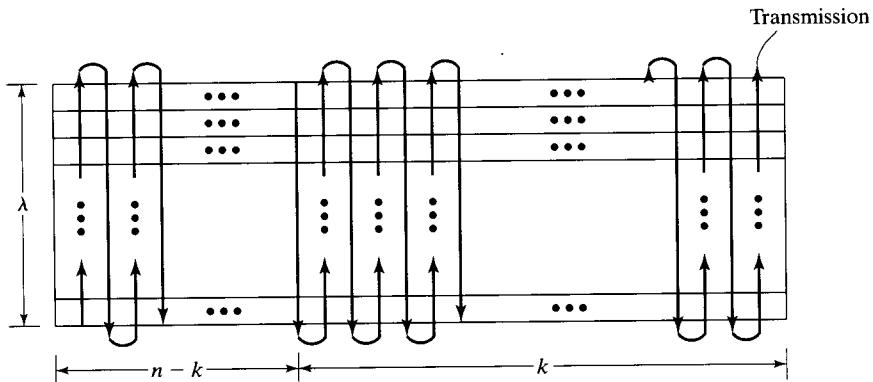


FIGURE 4.6: Transmission of an interleaved code.

is a correctable pattern for the original code C . The interleaving technique is very effective for deriving long, powerful codes for correcting errors that cluster to form *bursts*. This topic will be discussed in a later chapter.

Interleaving a single code can easily be generalized to interleaving several different codes of the same length. For $1 \leq i \leq \lambda$, let C_i be an (n, k_i) linear block code. Take λ codewords, one from each code, and arrange them as λ rows of a rectangular array as follows:

$$\begin{bmatrix} v_{1,0}, & v_{1,1}, & \cdots, & v_{1,n-1} \\ v_{2,0}, & v_{2,1}, & \cdots, & v_{2,n-1} \\ \vdots \\ v_{\lambda,0}, & v_{\lambda,1}, & \cdots, & v_{\lambda,n-1} \end{bmatrix}. \quad (4.80)$$

Then, transmit this array column by column. This interleaving of λ codes results in an $(\lambda n, k_1 + k_2 + \cdots + k_\lambda)$ linear block code, denoted by $C^\lambda = C_1 * C_2 * \cdots * C_\lambda$. Each column of the array given in (4.80) is a binary λ -tuple. If each column of (4.80) is regarded as an element in Galois field $GF(2^\lambda)$, then C^λ may be regarded as a linear block code with symbols from $GF(2^\lambda)$.

The interleaving technique presented here is called *block interleaving*. Other types of interleaving will be discussed in later chapters and can be found in [26].

PROBLEMS

- 4.1 Form a parity-check matrix for the $(15, 11)$ Hamming code. Devise a decoder for the code.
- 4.2 Show that Hamming codes achieve the Hamming bound (see Problem 3.15).
- 4.3 Show that the probability of an undetected error for Hamming codes of length $2^m - 1$ on a BSC with transition probability p satisfies the upper bound 2^{-m} for $p \leq 1/2$. (*Hint:* Use the inequality $(1 - 2p) \leq (1 - p)^2$.)

- 4.4** Compute the probability of an undetected error for the (15, 11) code on a BSC with transition probability $p = 10^{-2}$.
- 4.5** Devise a decoder for the (22, 16) SEC-DED code whose parity-check matrix is given in Figure 4.1(a).
- 4.6** Form the generator matrix of the first-order RM code RM(1, 3) of length 8. What is the minimum distance of the code? Determine its parity-check sums and devise a majority-logic decoder for the code. Decode the received vector $\mathbf{r} = (0\ 1\ 0\ 0\ 0\ 1\ 0\ 1)$.
- 4.7** Form the generator matrix of the first-order RM code RM(1, 4) of length 16. What is the minimum distance of the code? Determine its parity-check sums and devise a majority-logic decoder for the code. Decode the received vector $\mathbf{r} = (0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 1)$.
- 4.8** Find the parity-check sums for the second-order RM code RM(2, 5) of length 32. What is the minimum distance of the code? Form the parity-check sums for the code. Describe the decoding steps.
- 4.9** Prove that the $(m - r - 1)$ th-order RM code, RM($m - r - 1, m$), is the dual code of the r th-order RM code, RM(r, m).
- 4.10** Show that the RM(1, 3) and RM(2, 5) codes are self-dual.
- 4.11** Find a parity-check matrix for the RM(1, 4) code.
- 4.12** Construct the RM(2, 5) code of length 32 from RM codes of length 8 using $|\mathbf{u}|\mathbf{u} + \mathbf{v}|$ -construction.
- 4.13** Using the $|\mathbf{u}|\mathbf{u} + \mathbf{v}|$ -construction, decompose the RM(2, 5) code into component codes that are either repetition codes of dimension 1 or even parity-check codes of minimum distance 2.
- 4.14** Determine the Boolean polynomials that give the codewords of the RM(1, 3) code.
- 4.15** Use Boolean representation to show that the RM(r, m) code can be constructed from RM($r, m - 1$) and RM($r - 1, m - 1$) codes.
- 4.16** Construct the RM(2, 4) code from the RM(2, 3) and RM(1, 3) codes using one-level squaring construction. Find its generator matrix in the form of (4.53) or (4.68).
- 4.17** Using two-level squaring construction, express the generator matrix of the RM(2, 4) code in the forms of (4.60) and (4.61).
- 4.18** Prove that the (24, 12) Golay code is self-dual. (*Hint:* Show that $\mathbf{G} \cdot \mathbf{G}^T = 0$.)
- 4.19** Design an encoding circuit for the (24, 12) Golay code.
- 4.20** Suppose that the (24, 12) Golay code is used for error correction. Decode the following received sequences:
 - $\mathbf{r} = (1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 1)$,
 - $\mathbf{r} = (0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1)$.
- 4.21** Show that the digits for checking the parity-check digits of a product code array shown in Figure 4.3 are the same no matter whether they are formed by using the parity-check rules for C_2 on columns or the parity-check rules for C_1 on rows.
- 4.22** Prove that the minimum distance of the incomplete product of an (n_1, k_1, d_1) linear code and an (n_2, k_2, d_2) linear code is $d_1 + d_2 - 1$.
- 4.23** The incomplete product of the $(n_1, n_1 - 1, 2)$ and the $(n_2, n_2 - 1, 2)$ even parity-check codes has a minimum distance of 3. Devise a decoding algorithm for correcting a single error in the information part of a code array.

BIBLIOGRAPHY

1. R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell Syst. Tech. J.*, 29: 147–60, April 1950.
2. W. W. Peterson and E. J. Weldon, Jr., *Error Correcting Codes*, 2d ed., MIT Press, Cambridge, 1972.
3. M. J. E. Golay, "Notes on Digital Coding," *Proc. IEEE*, 37: 657, June 1949.
4. A. Tietavainen, "On the Nonexistence of Perfect Codes over Finite Fields," *SIAM J. Appl. Math.*, 24: 88–96, 1973.
5. V. Pless, *Introduction to the Theory of Error Correcting Codes*, 2d ed., John Wiley, New York, 1989.
6. S. K. Leung-Yan-Cheong and M. E. Hellman, "Concerning a Bound on Undetected Error Probability," *IEEE Trans. Inform. Theory*, IT-22: 235–37, March 1976.
7. T. Klove, *Error Detecting Codes*, Kluwer Academic, Boston, Mass., 1995.
8. M. Y. Hsiao, "A Class of Optimal Minimum Odd-Weight-Column SEC–DED Codes," *IBM J. Res. Dev.*, 14, July 1970.
9. D. E. Muller, "Applications of Boolean Algebra to Switching Circuits Design and to Error Detection," *IRE Trans.*, EC-3: 6–12, September 1954.
10. I. S. Reed, "A Class of Multiple-Error-Correcting Codes and the Decoding Scheme," *IRE Trans.*, IT-4: 38–49, September 1954.
11. J. L. Massey, *Threshold Decoding*, MIT Press, Cambridge, 1963.
12. F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North Holland, Amsterdam, 1977.
13. M. Plotkin, "Binary Codes with Specific Minimum Distance," *IEEE Trans. Inform. Theory*, IT-6: 445–50, November 1960.
14. V. P. Nelson, H. T. Nagle, B. D. Carroll, and J. D. Irwin, *Digital Logic Circuit Analysis & Design*, Prentice Hall, Englewood Cliffs, N.J., 1995.
15. G. D. Forney Jr. "Coset Codes II: Binary Lattices and Related Codes," *IEEE Trans. Inform. Theory*, IT-34: 1152–1187, September 1988.
16. S. Lin, T. Kasami, T. Fujiwara, and M. Fossorier, *Trellises and Trellis-Based Decoding Algorithms for Linear Block Codes*, Kluwer Academic, Boston, Mass., 1998.
17. S. Lin, "Some Codes Which Are Invariant under a Transitive Permutation Group and Their Connection with Balanced Incomplete Block Designs," chap. 24 in *Combinatorial Mathematics and Its Applications*, ed. R. C. Bose and T. A. Dowling, University of North Carolina Press, Chapel Hill, 1969.

18. T. Kasami, "The Weight Enumerators for Several Classes of Subcodes of the Second-Order Binary Reed-Muller Codes," *Inform. and Control*, 18: 369-94, 1971.
19. M. Sugino, Y. Ienaga, N. Tokura, and T. Kasami, "Weight Distribution of (128, 64) Reed-Muller Code," *IEEE Trans. Inform. Theory*, IT-17: 627-28, September 1971.
20. T. Kasami, N. Tokura, and S. Azumi, "On the Weight Enumeration of Weight Less than $2.5d$ of Reed-Muller Codes," *Inform. and Control*, 30: 380-95, April 1976.
21. T. Sugita, T. Kasami, and T. Fujiwara, "The Weight Distributions of the Third-Order Reed-Muller Code of Length 512," *IEEE Trans. Inform. Theory*, IT-42: 1622-25, September 1996.
22. R. G. Gallager, *Low-Density Parity-Check Codes*, MIT Press, Cambridge, 1960.
23. S. A. Vanstone and P. C. van Oorschot, *An Introduction to Error Correcting Codes with Applications*, Kluwer Academic, Boston, Mass., 1989.
24. S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, Englewood Cliffs, N.J., 1995.
25. P. Elias, "Error-Free Coding," *IRE Trans. Inform. Theory*, PGIT-4: 29-37, September 1954.
26. G. C. Clark Jr. and J. B. Cain, *Error-Correction Coding for Digital Communications*, Plenum, New York, 1981.

CHAPTER 5

Cyclic Codes

Cyclic codes form an important subclass of linear block codes. These codes are attractive for two reasons: first, encoding and syndrome computation can be implemented easily by employing shift registers with feedback connections (known as linear sequential circuits); and second, because they have considerable inherent algebraic structure, it is possible to devise various practical methods for decoding them. Cyclic codes are widely used in communication systems for error control. They are particularly efficient for error detection.

Cyclic codes were first studied by Eugene Prange in 1957 [1]. Since then, progress in the study of cyclic codes for both random-error correction and burst-error correction has been spurred by many algebraic coding theorists. Many classes of cyclic codes have been constructed over the years, including BCH codes, Reed–Solomon codes, Euclidean geometry codes, projective geometry codes, quadratic residue codes, and Fire codes, which will be discussed in later chapters. Excellent expositions of cyclic codes can be found in [2–5]. References [6–9] also provide good coverage of cyclic codes.

5.1 DESCRIPTION OF CYCLIC CODES

If we cyclically shift the components of an n -tuple $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ one place to the right, we obtain another n -tuple,

$$\mathbf{v}^{(1)} = (v_{n-1}, v_0, \dots, v_{n-2}),$$

which is called a *cyclic shift* of \mathbf{v} . If the components of \mathbf{v} are cyclically shifted i places to the right, the resultant n -tuple is

$$\mathbf{v}^{(i)} = (v_{n-i}, v_{n-i+1}, \dots, v_{n-1}, v_0, v_1, \dots, v_{n-i-1}).$$

Clearly, cyclically shifting \mathbf{v} i places to the right is equivalent to cyclically shifting \mathbf{v} $n - i$ places to the left.

DEFINITION 5.1 An (n, k) linear code C is called a *cyclic code* if every cyclic shift of a codeword in C is also a codeword in C .

The $(7, 4)$ linear code given in Table 5.1 is a cyclic code.

To develop the algebraic properties of a cyclic code, we treat the components of a codeword $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ as the coefficients of a polynomial as follows:

$$\mathbf{v}(X) = v_0 + v_1 X + v_2 X^2 + \dots + v_{n-1} X^{n-1}.$$

Thus, each codeword corresponds to a polynomial of degree $n - 1$ or less. If $v_{n-1} \neq 0$, the degree of $\mathbf{v}(X)$ is $n - 1$; if $v_{n-1} = 0$, the degree of $\mathbf{v}(X)$ is less than $n - 1$. The correspondence between the codeword \mathbf{v} and the polynomial $\mathbf{v}(X)$ is one-to-one. We

TABLE 5.1: A (7, 4) cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$.

Messages	Code vectors	Code polynomials
(0000)	0000000	$0 = 0 \cdot \mathbf{g}(X)$
(1000)	1101000	$1 + X + X^3 = 1 \cdot \mathbf{g}(X)$
(0100)	0110100	$X + X^2 + X^4 = X \cdot \mathbf{g}(X)$
(1100)	1011100	$1 + X^2 + X^3 + X^4 = (1 + X) \cdot \mathbf{g}(X)$
(0010)	0011010	$X^2 + X^3 + X^5 = X^2 \cdot \mathbf{g}(X)$
(1010)	1110010	$1 + X + X^2 + X^5 = (1 + X^2) \cdot \mathbf{g}(X)$
(0110)	0101110	$X + X^3 + X^4 + X^5 = (X + X^2) \cdot \mathbf{g}(X)$
(1110)	1000110	$1 + X^4 + X^5 = (1 + X + X^2) \cdot \mathbf{g}(X)$
(0001)	0001101	$X^3 + X^4 + X^6 = X^3 \cdot \mathbf{g}(X)$
(1001)	1100101	$1 + X + X^4 + X^6 = (1 + X^3) \cdot \mathbf{g}(X)$
(0101)	0111001	$X + X^2 + X^3 + X^6 = (X + X^3) \cdot \mathbf{g}(X)$
(1101)	1010001	$1 + X^2 + X^6 = (1 + X + X^3) \cdot \mathbf{g}(X)$
(0011)	0010111	$X^2 + X^4 + X^5 + X^6 = (X^2 + X^3) \cdot \mathbf{g}(X)$
(1011)	1111111	$1 + X + X^2 + X^3 + X^4 + X^5 + X^6 = (1 + X^2 + X^3) \cdot \mathbf{g}(X)$
(0111)	0100011	$X + X^5 + X^6 = (X + X^2 + X^3) \cdot \mathbf{g}(X)$
(1111)	1001011	$1 + X^3 + X^5 + X^6 = (1 + X + X^2 + X^3) \cdot \mathbf{g}(X)$

shall call $\mathbf{v}(X)$ the code polynomial of \mathbf{v} . Hereafter, we shall use the terms *codeword* and *code polynomial* interchangeably. The code polynomial that corresponds to codeword $\mathbf{v}^{(i)}$ is

$$\mathbf{v}^{(i)}(X) = v_{n-i} + v_{n-i+1}X + \cdots + v_{n-1}X^{i-1} + v_0X^i + v_1X^{i+1} + \cdots + v_{n-i-1}X^{n-1}.$$

There exists an interesting algebraic relationship between $\mathbf{v}(X)$ and $\mathbf{v}^{(i)}(X)$. Multiplying $\mathbf{v}(X)$ by X^i , we obtain

$$X^i \mathbf{v}(X) = v_0X^i + v_1X^{i+1} + \cdots + v_{n-i-1}X^{n-1} + \cdots + v_{n-1}X^{n+i-1}.$$

The preceding equation can be manipulated into the following form:

$$\begin{aligned} X^i \mathbf{v}(X) &= v_{n-i} + v_{n-i+1}X + \cdots + v_{n-1}X^{i-1} + v_0X^i + \cdots + v_{n-i-1}X^{n-1} \\ &\quad + v_{n-i}(X^n + 1) + v_{n-i+1}X(X^n + 1) + \cdots + v_{n-1}X^{i-1}(X^n + 1) \quad (5.1) \\ &= \mathbf{q}(X)(X^n + 1) + \mathbf{v}^{(i)}(X), \end{aligned}$$

where $\mathbf{q}(X) = v_{n-i} + v_{n-i+1}X + \cdots + v_{n-1}X^{i-1}$. From (5.1) we see that the code polynomial $\mathbf{v}^{(i)}(X)$ is simply the remainder resulting from dividing the polynomial $X^i \mathbf{v}(X)$ by $X^n + 1$.

Next, we prove a number of important algebraic properties of a cyclic code that make possible the simple implementation of encoding and syndrome computation.

THEOREM 5.1 The nonzero code polynomial of minimum degree in a cyclic code C is unique.

Proof. Let $\mathbf{g}(X) = g_0 + g_1X + \cdots + g_{r-1}X^{r-1} + X^r$ be a nonzero code polynomial of minimum degree in C . Suppose that $\mathbf{g}(X)$ is not unique. Then, there exists another code polynomial of degree r , say $\mathbf{g}'(X) = g'_0 + g'_1X + \cdots + g'_{r-1}X^{r-1} + X^r$. Because C is linear, $\mathbf{g}(X) + \mathbf{g}'(X) = (g_0 + g'_0) + (g_1 + g'_1)X + \cdots + (g_{r-1} + g'_{r-1})X^{r-1}$ is a code polynomial of degree less than r . If $\mathbf{g}(X) + \mathbf{g}'(X) \neq 0$, then $\mathbf{g}(X) + \mathbf{g}'(X)$ is a nonzero code polynomial of degree less than the minimum degree r . This is impossible. Therefore, $\mathbf{g}(X) + \mathbf{g}'(X) = 0$. This implies that $\mathbf{g}'(X) = \mathbf{g}(X)$. Hence, $\mathbf{g}(X)$ is unique. **Q.E.D.**

THEOREM 5.2 Let $\mathbf{g}(X) = g_0 + g_1X + \cdots + g_{r-1}X^{r-1} + X^r$ be the nonzero code polynomial of minimum degree in an (n, k) cyclic code C . Then, the constant term g_0 must be equal to 1.

Proof. Suppose that $g_0 = 0$. Then,

$$\begin{aligned}\mathbf{g}(X) &= g_1X + g_2X^2 + \cdots + g_{r-1}X^{r-1} + X^r \\ &= X(g_1 + g_2X + \cdots + g_{r-1}X^{r-2} + X^{r-1}).\end{aligned}$$

If we shift $\mathbf{g}(X)$ cyclically $n - 1$ places to the right (or one place to the left), we obtain a nonzero code polynomial, $g_1 + g_2X + \cdots + g_{r-1}X^{r-2} + X^{r-1}$, of degree less than r . This is a contradiction to the assumption that $\mathbf{g}(X)$ is the nonzero code polynomial with minimum degree. Thus, $g_0 \neq 0$. **Q.E.D.**

It follows from Theorem 5.2 that the nonzero code polynomial of minimum degree in an (n, k) cyclic code C is of the following form:

$$\mathbf{g}(X) = 1 + g_1X + g_2X^2 + \cdots + g_{r-1}X^{r-1} + X^r. \quad (5.2)$$

Consider the $(7, 4)$ cyclic code given in Table 5.1. The nonzero code polynomial of minimum degree is $\mathbf{g}(X) = 1 + X + X^3$.

Consider the polynomials $X\mathbf{g}(X)$, $X^2\mathbf{g}(X)$, \dots , $X^{n-r-1}\mathbf{g}(X)$, of degrees $r + 1, r + 2, \dots, n - 1$, respectively. It follows from (5.1) that $X\mathbf{g}(X) = \mathbf{g}^{(1)}(X)$, $X^2\mathbf{g}(X) = \mathbf{g}^{(2)}(X), \dots, X^{n-r-1}\mathbf{g}(X) = \mathbf{g}^{(n-r-1)}(X)$; that is, they are cyclic shifts of the code polynomial $\mathbf{g}(X)$. Therefore, they are code polynomials in C . Since C is linear, a linear combination of $\mathbf{g}(X), X\mathbf{g}(X), \dots, X^{n-r-1}\mathbf{g}(X)$,

$$\begin{aligned}\mathbf{v}(X) &= u_0\mathbf{g}(X) + u_1X\mathbf{g}(X) + \cdots + u_{n-r-1}X^{n-r-1}\mathbf{g}(X) \\ &= (u_0 + u_1X + \cdots + u_{n-r-1}X^{n-r-1})\mathbf{g}(X),\end{aligned} \quad (5.3)$$

is also a code polynomial, where $u_i = 0$ or 1. The following theorem characterizes an important property of a cyclic code.

THEOREM 5.3 Let $\mathbf{g}(X) = 1 + g_1X + \cdots + g_{r-1}X^{r-1} + X^r$ be the nonzero code polynomial of minimum degree in an (n, k) cyclic code C . A binary polynomial of degree $n - 1$ or less is a code polynomial if and only if it is a multiple of $\mathbf{g}(X)$.

Proof. Let $\mathbf{v}(X)$ be a binary polynomial of degree $n - 1$ or less. Suppose that $\mathbf{v}(X)$ is a multiple of $\mathbf{g}(X)$. Then,

$$\begin{aligned}\mathbf{v}(X) &= (a_0 + a_1X + \cdots + a_{n-r-1}X^{n-r-1})\mathbf{g}(X) \\ &= a_0\mathbf{g}(X) + a_1X\mathbf{g}(X) + \cdots + a_{n-r-1}X^{n-r-1}\mathbf{g}(X).\end{aligned}$$

Because $\mathbf{v}(X)$ is a linear combination of the code polynomials $\mathbf{g}(X), X\mathbf{g}(X), \dots, X^{n-r-1}\mathbf{g}(X)$, it is a code polynomial in C . This proves the first part of the theorem—that if a polynomial of degree $n - 1$ or less is a multiple of $\mathbf{g}(X)$, it is a code polynomial.

Now, let $\mathbf{v}(X)$ be a code polynomial in C . Dividing $\mathbf{v}(X)$ by $\mathbf{g}(X)$, we obtain

$$\mathbf{v}(X) = \mathbf{a}(X)\mathbf{g}(X) + \mathbf{b}(X),$$

where either $\mathbf{b}(X)$ is identical to zero, or the degree of $\mathbf{b}(X)$ is less than the degree of $\mathbf{g}(X)$. Rearranging the preceding equation, we have

$$\mathbf{b}(X) = \mathbf{v}(X) + \mathbf{a}(X)\mathbf{g}(X).$$

It follows from the first part of the theorem that $\mathbf{a}(X)\mathbf{g}(X)$ is a code polynomial. Because both $\mathbf{v}(X)$ and $\mathbf{a}(X)\mathbf{g}(X)$ are code polynomials, $\mathbf{b}(X)$ must also be a code polynomial. If $\mathbf{b}(X) \neq 0$, then $\mathbf{b}(X)$ is a nonzero code polynomial whose degree is less than the degree of $\mathbf{g}(X)$. This contradicts the assumption that $\mathbf{g}(X)$ is the nonzero code polynomial of minimum degree. Thus, $\mathbf{b}(X)$ must be identical to zero. This proves the second part of the theorem—that a code polynomial is a multiple of $\mathbf{g}(X)$. **Q.E.D.**

The number of binary polynomials of degree $n - 1$ or less that are multiples of $\mathbf{g}(X)$ is 2^{n-r} . It follows from Theorem 5.3 that these polynomials form all the code polynomials of the (n, k) cyclic code C . Because there are 2^k code polynomials in C , then 2^{n-r} must be equal to 2^k . As a result, we have $r = n - k$ [i.e., the degree of $\mathbf{g}(X)$ is $n - k$]. Hence, the nonzero code polynomial of minimum degree in an (n, k) cyclic code is of the following form:

$$\mathbf{g}(X) = 1 + \mathbf{g}_1 X + \mathbf{g}_2 X^2 + \dots + \mathbf{g}_{n-k-1} X^{n-k-1} + X^{n-k}.$$

Summarizing the preceding results, we have the following theorem:

THEOREM 5.4 In an (n, k) cyclic code, there exists one and only one code polynomial of degree $n - k$,

$$\mathbf{g}(X) = 1 + \mathbf{g}_1 X + \mathbf{g}_2 X^2 + \dots + \mathbf{g}_{n-k-1} X^{n-k-1} + X^{n-k}. \quad (5.4)$$

Every code polynomial is a multiple of $\mathbf{g}(X)$, and every binary polynomial of degree $n - 1$ or less that is a multiple of $\mathbf{g}(X)$ is a code polynomial.

It follows from Theorem 5.4 that every code polynomial $\mathbf{v}(X)$ in an (n, k) cyclic code can be expressed in the following form:

$$\begin{aligned} \mathbf{v}(X) &= \mathbf{u}(X)\mathbf{g}(X) \\ &= (u_0 + u_1 X + \dots + u_{k-1} X^{k-1})\mathbf{g}(X). \end{aligned}$$

If the coefficients of $\mathbf{u}(X), u_0, u_1, \dots, u_{k-1}$, are the k information digits to be encoded, $\mathbf{v}(X)$ is the corresponding code polynomial. Hence, the encoding can be achieved by multiplying the message $\mathbf{u}(X)$ by $\mathbf{g}(X)$. Therefore, an (n, k) cyclic code is completely specified by its nonzero code polynomial of minimum degree, $\mathbf{g}(X)$.

given by (5.4). The polynomial $\mathbf{g}(X)$ is called the *generator polynomial* of the code. The degree of $\mathbf{g}(X)$ is equal to the number of parity-check digits of the code. The generator polynomial of the $(7, 4)$ cyclic code given in Table 4.1 is $\mathbf{g}(X) = 1 + X + X^3$. We see that each code polynomial is a multiple of $\mathbf{g}(X)$.

The next important property of a cyclic code is given in the following theorem.

THEOREM 5.5 The generator polynomial $\mathbf{g}(X)$ of an (n, k) cyclic code is a factor of $X^n + 1$.

Proof. Multiplying $\mathbf{g}(X)$ by X^k results in a polynomial $X^k \mathbf{g}(X)$ of degree n . Dividing $X^k \mathbf{g}(X)$ by $X^n + 1$, we obtain

$$X^k \mathbf{g}(X) = (X^n + 1) + \mathbf{g}^{(k)}(X), \quad (5.5)$$

where $\mathbf{g}^{(k)}(X)$ is the remainder. It follows from (5.1) that $\mathbf{g}^{(k)}(X)$ is the code polynomial obtained by shifting $\mathbf{g}(X)$ to the right cyclically k times. Hence, $\mathbf{g}^{(k)}(X)$ is a multiple of $\mathbf{g}(X)$, say $\mathbf{g}^{(k)}(X) = \mathbf{a}(X) \mathbf{g}(X)$. From (5.5) we obtain

$$X^n + 1 = \{X^k + \mathbf{a}(X)\} \mathbf{g}(X).$$

Thus, $\mathbf{g}(X)$ is a factor of $X^n + 1$. Q.E.D.

At this point, a natural question is whether, for any n and k , there exists an (n, k) cyclic code. This question is answered by the following theorem.

THEOREM 5.6 If $\mathbf{g}(X)$ is a polynomial of degree $n - k$ and is a factor of $X^n + 1$, then $\mathbf{g}(X)$ generates an (n, k) cyclic code.

Proof. Consider the k polynomials $\mathbf{g}(X), X\mathbf{g}(X), \dots, X^{k-1}\mathbf{g}(X)$, all of degree $n - 1$ or less. A linear combination of these k polynomials,

$$\begin{aligned} \mathbf{v}(X) &= a_0 \mathbf{g}(X) + a_1 X \mathbf{g}(X) + \cdots + a_{k-1} X^{k-1} \mathbf{g}(X) \\ &= (a_0 + a_1 X + \cdots + a_{k-1} X^{k-1}) \mathbf{g}(X), \end{aligned}$$

is also a polynomial of degree $n - 1$ or less and is a multiple of $\mathbf{g}(X)$. There are a total of 2^k such polynomials, and they form an (n, k) linear code.

Let $\mathbf{v}(X) = v_0 + v_1 X + \cdots + v_{n-1} X^{n-1}$ be a code polynomial in this code. Multiplying $\mathbf{v}(X)$ by X , we obtain

$$\begin{aligned} X\mathbf{v}(X) &= v_0 X + v_1 X^2 + \cdots + v_{n-2} X^{n-1} + v_{n-1} X^n \\ &= v_{n-1} (X^n + 1) + (v_{n-1} + v_0 X + \cdots + v_{n-2} X^{n-1}) \\ &= v_{n-1} (X^n + 1) + \mathbf{v}^{(1)}(X), \end{aligned}$$

where $\mathbf{v}^{(1)}(X)$ is a cyclic shift of $\mathbf{v}(X)$. Since both $X\mathbf{v}(X)$ and $X^n + 1$ are divisible by $\mathbf{g}(X)$, $\mathbf{v}^{(1)}(X)$ must be divisible by $\mathbf{g}(X)$. Thus, $\mathbf{v}^{(1)}(X)$ is a multiple of $\mathbf{g}(X)$ and is a linear combination of $\mathbf{g}(X), X\mathbf{g}(X), \dots, X^{k-1}\mathbf{g}(X)$. Hence, $\mathbf{v}^{(1)}(X)$ is also a code polynomial. It follows from Definition 5.1 that the linear code generated by $\mathbf{g}(X), X\mathbf{g}(X), \dots, X^{k-1}\mathbf{g}(X)$ is an (n, k) cyclic code. Q.E.D.

Theorem 5.6 says that any factor of $X^n + 1$ with degree $n - k$ generates an (n, k) cyclic code. For large n , $X^n + 1$ may have many factors of degree $n - k$. Some of these polynomials generate good codes, and some generate bad codes. How to select generator polynomials to produce good cyclic codes is a very difficult problem, and coding theorists have expended much effort in searching for good cyclic codes. Several classes of good cyclic codes have been discovered, and they can be practically implemented.

EXAMPLE 5.1

The polynomial $X^7 + 1$ can be factored as follows:

$$X^7 + 1 = (1 + X)(1 + X + X^3)(1 + X^2 + X^3).$$

There are two factors of degree 3, and each generates a $(7, 4)$ cyclic code. The $(7, 4)$ cyclic code given by Table 5.1 is generated by $\mathbf{g}(X) = 1 + X + X^3$. This code has a minimum distance of 3 and it is a single-error-correcting code. Notice that the code is not in systematic form. Each code polynomial is the product of a message polynomial of degree 3 or less and the generator polynomial $\mathbf{g}(X) = 1 + X + X^3$. For example, let $\mathbf{u} = (1\ 0\ 1\ 0)$ be the message to be encoded. The corresponding message polynomial is $\mathbf{u}(X) = 1 + X^2$. Multiplying $\mathbf{u}(X)$ by $\mathbf{g}(X)$ gives us the following code polynomial:

$$\begin{aligned}\mathbf{v}(X) &= (1 + X^2)(1 + X + X^3) \\ &= 1 + X + X^2 + X^5,\end{aligned}$$

or the codeword $(1\ 1\ 1\ 0\ 0\ 1\ 0)$.

Given the generator polynomial $\mathbf{g}(X)$ of an (n, k) cyclic code, we can put the code into systematic form (i.e., the rightmost k digits of each codeword are the unaltered information digits, and the leftmost $n - k$ digits are parity-check digits). Suppose that the message to be encoded is $\mathbf{u} = (u_0, u_1, \dots, u_{k-1})$. The corresponding message polynomial is

$$\mathbf{u}(X) = u_0 + u_1X + \dots + u_{k-1}X^{k-1}.$$

Multiplying $\mathbf{u}(X)$ by X^{n-k} , we obtain a polynomial of degree $n - 1$ or less:

$$X^{n-k}\mathbf{u}(X) = u_0X^{n-k} + u_1X^{n-k+1} + \dots + u_{k-1}X^{n-1}.$$

Dividing $X^{n-k}\mathbf{u}(X)$ by the generator polynomial $\mathbf{g}(X)$, we have

$$X^{n-k}\mathbf{u}(X) = \mathbf{a}(X)\mathbf{g}(X) + \mathbf{b}(X) \tag{5.6}$$

where $\mathbf{a}(X)$ and $\mathbf{b}(X)$ are the quotient and the remainder, respectively. Because the degree of $\mathbf{g}(X)$ is $n - k$, the degree of $\mathbf{b}(X)$ must be $n - k - 1$ or less; that is,

$$\mathbf{b}(X) = b_0 + b_1X + \dots + b_{n-k-1}X^{n-k-1}.$$

Rearranging (5.6), we obtain the following polynomial of degree $n - 1$ or less:

$$\mathbf{b}(X) + X^{n-k}\mathbf{u}(X) = \mathbf{a}(X)\mathbf{g}(X). \quad (5.7)$$

This polynomial is a multiple of the generator polynomial $\mathbf{g}(X)$ and therefore it is a code polynomial of the cyclic code generated by $\mathbf{g}(X)$. Writing out $\mathbf{b}(X) + X^{n-k}\mathbf{u}(X)$, we have

$$\begin{aligned} \mathbf{b}(X) + X^{n-k}\mathbf{u}(X) &= b_0 + b_1X + \cdots + b_{n-k-1}X^{n-k-1} \\ &\quad + u_0X^{n-k} + u_1X^{n-k+1} + \cdots + u_{k-1}X^{n-1}, \end{aligned} \quad (5.8)$$

which corresponds to the codeword

$$(b_0, b_1, \dots, b_{n-k-1}, u_0, u_1, \dots, u_{k-1}).$$

We see that the codeword consists of k unaltered information digits $(u_0, u_1, \dots, u_{k-1})$ followed by $n - k$ parity-check digits. The $n - k$ parity-check digits are simply the coefficients of the remainder resulting from dividing the message polynomial $X^{n-k}\mathbf{u}(X)$ by the generator polynomial $\mathbf{g}(X)$. The preceding process yields an (n, k) cyclic code in systematic form. In connection with cyclic codes in systematic form, the following convention is used: the first $n - k$ symbols, the coefficients of $1, X, \dots, X^{n-k-1}$, are taken as parity-check digits, and the last k symbols, the coefficients of $X^{n-k}, X^{n-k+1}, \dots, X^{n-1}$, are taken as the information digits. In summary, encoding in systematic form consists of three steps:

- Step 1.** Premultiply the message $\mathbf{u}(X)$ by X^{n-k} .
- Step 2.** Obtain the remainder $\mathbf{b}(X)$ (the parity-check digits) from dividing $X^{n-k}\mathbf{u}(X)$ by the generator polynomial $\mathbf{g}(X)$.
- Step 3.** Combine $\mathbf{b}(X)$ and $X^{n-k}\mathbf{u}(X)$ to obtain the code polynomial $\mathbf{b}(X) + X^{n-k}\mathbf{u}(X)$.

EXAMPLE 5.2

Consider the $(7, 4)$ cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$. Let $\mathbf{u}(X) = 1 + X^3$ be the message to be encoded. Dividing $X^3\mathbf{u}(X) = X^3 + X^6$ by $\mathbf{g}(X)$,

$$\begin{array}{r} X^3 + X \quad (\text{quotient}) \\ \hline X^3 + X + 1 | X^6 & X^3 \\ X^6 & + X^4 + X^3 \\ \hline X^4 & \\ X^4 & + X^2 + X \\ \hline X^2 + X & \quad (\text{remainder}) \end{array}$$

we obtain the remainder $\mathbf{b}(X) = X + X^2$. Thus, the code polynomial is $\mathbf{v}(X) = \mathbf{b}(X) + X^3\mathbf{u}(X) = X + X^2 + X^3 + X^6$, and the corresponding codeword is $\mathbf{v} = (0\ 1\ 1\ 1\ 0\ 0\ 1)$, where the four rightmost digits are the information digits. The 16 codewords in systematic form are listed in Table 5.2.

TABLE 5.2: A (7, 4) cyclic code in systematic form generated by $\mathbf{g}(X) = 1 + X + X^3$.

Message	Codeword	
(0 0 0 0)	(0 0 0 0 0 0 0)	$0 = 0 \cdot \mathbf{g}(X)$
(1 0 0 0)	(1 1 0 1 0 0 0)	$1 + X + X^3 = \mathbf{g}(X)$
(0 1 0 0)	(0 1 1 0 1 0 0)	$X + X^2 + X^4 = X\mathbf{g}(X)$
(1 1 0 0)	(1 0 1 1 1 0 0)	$1 + X^2 + X^3 + X^4 = (1 + X)\mathbf{g}(X)$
(0 0 1 0)	(1 1 1 0 0 1 0)	$1 + X + X^2 + X^5 = (1 + X^2)\mathbf{g}(X)$
(1 0 1 0)	(0 0 1 1 0 1 0)	$X^2 + X^3 + X^5 = X^2\mathbf{g}(X)$
(0 1 1 0)	(1 0 0 0 1 1 0)	$1 + X^4 + X^5 = (1 + X + X^2)\mathbf{g}(X)$
(1 1 1 0)	(0 1 0 1 1 1 0)	$X + X^3 + X^4 + X^5 = (X + X^2)\mathbf{g}(X)$
(0 0 0 1)	(1 0 1 0 0 0 1)	$1 + X^2 + X^6 = (1 + X + X^3)\mathbf{g}(X)$
(1 0 0 1)	(0 1 1 1 0 0 1)	$X + X^2 + X^3 + X^6 = (X + X^3)\mathbf{g}(X)$
(0 1 0 1)	(1 1 0 0 1 0 1)	$1 + X + X^4 + X^6 = (1 + X^3)\mathbf{g}(X)$
(1 1 0 1)	(0 0 0 1 1 0 1)	$X^3 + X^4 + X^6 = X^3\mathbf{g}(X)$
(0 0 1 1)	(0 1 0 0 0 1 1)	$X + X^5 + X^6 = (X + X^2 + X^3)\mathbf{g}(X)$
(1 0 1 1)	(1 0 0 1 0 1 1)	$1 + X^3 + X^5 + X^6 = (1 + X + X^2 + X^3)\mathbf{g}(X)$
(0 1 1 1)	(0 0 1 0 1 1 1)	$X^2 + X^4 + X^5 + X^6 = (X^2 + X^3)\mathbf{g}(X)$
(1 1 1 1)	(1 1 1 1 1 1 1)	$1 + X + X^2 + X^3 + X^4 + X^5 + X^6 = (1 + X^2 + X^5)\mathbf{g}(X)$

5.2 GENERATOR AND PARITY-CHECK MATRICES OF CYCLIC CODES

Consider an (n, k) cyclic code C with generator polynomial $\mathbf{g}(X) = g_0 + g_1X + \cdots + g_{n-k}X^{n-k}$. In Section 5.1 we showed that the k code polynomials $\mathbf{g}(X), X\mathbf{g}(X), \dots, X^{k-1}\mathbf{g}(X)$ span C . If the n -tuples corresponding to these k code polynomials are used as the rows of a $k \times n$ matrix, we obtain the following generator matrix for C :

$$\mathbf{G} = \begin{bmatrix} g_0 & g_1 & g_2 & \cdot & \cdot & \cdot & \cdot & \cdot & g_{n-k} & 0 & 0 & 0 & \cdot & \cdot & 0 \\ 0 & g_0 & g_1 & g_2 & \cdot & \cdot & \cdot & \cdot & \cdot & g_{n-k} & 0 & 0 & \cdot & \cdot & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & \cdot & \cdot & \cdot & \cdot & \cdot & g_{n-k} & 0 & \cdot & \cdot & 0 \\ \cdot & & \cdot & & & & & & & & & & & \cdot & & \cdot \\ \cdot & & \cdot & & & & & & & & & & & \cdot & & \cdot \\ 0 & 0 & \cdot & \cdot & \cdot & 0 & g_0 & g_1 & g_2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & g_{n-k} \end{bmatrix} \quad (5.9)$$

(Note that $g_0 = g_{n-k} = 1$.) In general, \mathbf{G} is not in systematic form; however, we can put it into systematic form with row operations. For example, the (7, 4) cyclic code given in Table 5.1 with generator polynomial $\mathbf{g}(X) = 1 + X + X^3$ has the following matrix as a generator matrix:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Clearly, \mathbf{G} is not in systematic form. If we add the first row to the third row, and if we add the sum of the first two rows to the fourth row, we obtain the following matrix:

$$\mathbf{G}' = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix},$$

which is in systematic form. This matrix generates the same code as \mathbf{G} .

Recall that the generator polynomial $\mathbf{g}(X)$ is a factor of $X^n + 1$, say

$$X^n + 1 = \mathbf{g}(X)\mathbf{h}(X), \quad (5.10)$$

where the polynomial $\mathbf{h}(X)$ has degree k and is of the following form:

$$\mathbf{h}(X) = h_0 + h_1 X + \cdots + h_k X^k$$

with $h_0 = h_k = 1$. Next, we want to show that a parity-check matrix of C may be obtained from $\mathbf{h}(X)$. Let $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ be a codeword in C . Then, $\mathbf{v}(X) = \mathbf{a}(X)\mathbf{g}(X)$. Multiplying $\mathbf{v}(X)$ by $\mathbf{h}(X)$, we obtain

$$\begin{aligned} \mathbf{v}(X)\mathbf{h}(X) &= \mathbf{a}(X)\mathbf{g}(X)\mathbf{h}(X) \\ &= \mathbf{a}(X)(X^n + 1) \\ &= \mathbf{a}(X) + X^n\mathbf{a}(X). \end{aligned} \quad (5.11)$$

Because the degree of $\mathbf{a}(X)$ is $k - 1$ or less, the powers $X^k, X^{k+1}, \dots, X^{n-1}$ do not appear in $\mathbf{a}(X) + X^n\mathbf{a}(X)$. If we expand the product $\mathbf{v}(X)\mathbf{h}(X)$ on the left-hand side of (5.11), the coefficients of $X^k, X^{k+1}, \dots, X^{n-1}$ must be equal to zero. Therefore, we obtain the following $n - k$ equalities:

$$\sum_{i=0}^k h_i v_{n-i-j} = 0 \quad \text{for } 1 \leq j \leq n - k. \quad (5.12)$$

Now, we take the *reciprocal* of $\mathbf{h}(X)$, which is defined as follows:

$$X^k \mathbf{h}(X^{-1}) \triangleq h_k + h_{k-1} X + h_{k-2} X^2 + \cdots + h_0 X^k. \quad (5.13)$$

We can easily see that $X^k \mathbf{h}(X^{-1})$ is also a factor of $X^n + 1$. The polynomial $X^k \mathbf{h}(X^{-1})$ generates an $(n, n - k)$ cyclic code with the following $(n - k) \times n$ matrix as a generator matrix:

$$\mathbf{H} = \begin{bmatrix} h_k & h_{k-1} & h_{k-2} & \cdot & \cdot & \cdot & \cdot & \cdot & h_0 & 0 & \cdot & \cdot & \cdot & \cdot & 0 \\ 0 & h_k & h_{k-1} & h_{k-2} & \cdot & \cdot & \cdot & \cdot & h_0 & 0 & \cdot & \cdot & \cdot & \cdot & 0 \\ 0 & 0 & h_k & h_{k-1} & h_{k-2} & \cdot & \cdot & \cdot & \cdot & h_0 & \cdot & \cdot & \cdot & \cdot & 0 \\ \vdots & & & & & & & & & & & & & & & \vdots \\ 0 & 0 & \cdot & \cdot & \cdot & 0 & h_k & h_{k-1} & h_{k-2} & \cdot & \cdot & \cdot & \cdot & \cdot & h_0 \end{bmatrix}. \quad (5.14)$$

It follows from the $n - k$ equalities of (5.12) that any codeword \mathbf{v} in C is orthogonal to every row of \mathbf{H} . Therefore, \mathbf{H} is a parity-check matrix of the cyclic code C , and the row space of \mathbf{H} is the dual code of C . Since the parity-check matrix \mathbf{H} is obtained from the polynomial $\mathbf{h}(X)$, we call $\mathbf{h}(X)$ the *parity polynomial* of C . Hence, a cyclic code is also uniquely specified by its parity polynomial.

Besides deriving a parity-check matrix for a cyclic code, we have also proved another important property, which is stated in the following theorem.

THEOREM 5.7 Let C be an (n, k) cyclic code with generator polynomial $\mathbf{g}(X)$. The dual code of C is also cyclic and is generated by the polynomial $X^k \mathbf{h}(X^{-1})$, where $\mathbf{h}(X) = (X^n + 1)/\mathbf{g}(X)$.

EXAMPLE 5.3

Consider the $(7, 4)$ cyclic code given in Table 5.1 with generator polynomial $\mathbf{g}(X) = 1 + X + X^3$. The parity polynomial is

$$\begin{aligned}\mathbf{h}(X) &= \frac{X^7 + 1}{\mathbf{g}(X)} \\ &= 1 + X + X^2 + X^4.\end{aligned}$$

The reciprocal of $\mathbf{h}(X)$ is

$$\begin{aligned}X^4 \mathbf{h}(X^{-1}) &= X^4(1 + X^{-1} + X^{-2} + X^{-4}) \\ &= 1 + X^2 + X^3 + X^4.\end{aligned}$$

This polynomial $X^4 \mathbf{h}(X^{-1})$ divides $X^7 + 1$: $(X^7 + 1)/X^4 \mathbf{h}(X^{-1}) = 1 + X^2 + X^3$. If we construct all the codewords of the $(7, 3)$ code generated by $X^4 \mathbf{h}(X^{-1}) = 1 + X^2 + X^3 + X^4$, we will find that it has a minimum distance of 4. Hence, it is capable of correcting any single error and simultaneously detecting any combination of double errors.

We also can easily form the generator matrix in systematic form. Dividing X^{n-k+i} by the generator polynomial $\mathbf{g}(X)$ for $i = 0, 1, \dots, k-1$, we obtain

$$X^{n-k+i} = \mathbf{a}_i(X)\mathbf{g}(X) + \mathbf{b}_i(X), \quad (5.15)$$

where $\mathbf{b}_i(X)$ is the remainder with the following form:

$$\mathbf{b}_i(X) = b_{i0} + b_{i1}X + \dots + b_{i,n-k-1}X^{n-k-1}.$$

Because $\mathbf{b}_i(X) + X^{n-k+i}$ for $i = 0, 1, \dots, k-1$ are multiples of $\mathbf{g}(X)$, they are code polynomials. Arranging these k code polynomials as rows of a $k \times n$ matrix, we obtain

$$\mathbf{G} = \left[\begin{array}{ccccccccc} b_{00} & b_{01} & b_{02} & \cdots & b_{0,n-k-1} & 1 & 0 & 0 & \cdots & 0 \\ b_{10} & b_{11} & b_{12} & \cdots & b_{1,n-k-1} & 0 & 1 & 0 & \cdots & 0 \\ b_{20} & b_{21} & b_{22} & \cdots & b_{2,n-k-1} & 0 & 0 & 1 & \cdots & 0 \\ \vdots & & & & \vdots & & & & \ddots & \vdots \\ b_{k-1,0} & b_{k-1,1} & b_{k-1,2} & \cdots & b_{k-1,n-k-1} & 0 & 0 & 0 & \cdots & 1 \end{array} \right], \quad (5.16)$$

which is the generator matrix of C in systematic form. The corresponding parity-check matrix for C is

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & b_{00} & b_{10} & b_{20} & \cdots & b_{k-1,0} \\ 0 & 1 & 0 & \cdots & 0 & b_{01} & b_{11} & b_{21} & \cdots & b_{k-1,1} \\ 0 & 0 & 1 & \cdots & 0 & b_{02} & b_{12} & b_{22} & \cdots & b_{k-1,2} \\ \vdots & & & & & \vdots & \vdots & & & \vdots \\ 0 & 0 & 0 & \cdots & 1 & b_{0,n-k-1} & b_{1,n-k-1} & b_{2,n-k-1} & \cdots & b_{k-1,n-k-1} \end{bmatrix}. \quad (5.17)$$

EXAMPLE 5.4

Again, consider the $(7, 4)$ cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$. Dividing X^3, X^4, X^5 , and X^6 by $\mathbf{g}(X)$, we have

$$\begin{aligned} X^3 &= \mathbf{g}(X) + (1 + X), \\ X^4 &= X\mathbf{g}(X) + (X + X^2), \\ X^5 &= (X^2 + 1)\mathbf{g}(X) + (1 + X + X^2), \\ X^6 &= (X^3 + X + 1)\mathbf{g}(X) + (1 + X^2). \end{aligned}$$

Rearranging the preceding equations, we obtain the following four code polynomials:

$$\begin{aligned} \mathbf{v}_0(X) &= 1 + X + X^3, \\ \mathbf{v}_1(X) &= X + X^2 + X^4, \\ \mathbf{v}_2(X) &= 1 + X + X^2 + X^5, \\ \mathbf{v}_3(X) &= 1 + X^2 + X^6. \end{aligned}$$

Taking these four code polynomials as rows of a 4×7 matrix, we obtain the following generator matrix in systematic form for the $(7, 4)$ cyclic code:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

which is identical to the matrix G' obtained earlier in this section.

5.3 ENCODING OF CYCLIC CODES

As shown in Section 5.1, encoding of an (n, k) cyclic code in systematic form consists of three steps: (1) multiplying the message polynomial $\mathbf{u}(X)$ by X^{n-k} ; (2) dividing $X^{n-k}\mathbf{u}(X)$ by $\mathbf{g}(X)$ to obtain the remainder $\mathbf{b}(X)$; and (3) forming the codeword $\mathbf{b}(X) + X^{n-k}\mathbf{u}(X)$. All three steps can be accomplished with a division circuit that is a linear $(n - k)$ -stage shift register with feedback connections based on the generator polynomial $\mathbf{g}(X) = 1 + g_1X + g_2X^2 + \cdots + g_{n-k-1}X^{n-k-1} + X^{n-k}$. Such a circuit is shown in Figure 5.1. The encoding operation is carried out as follows:

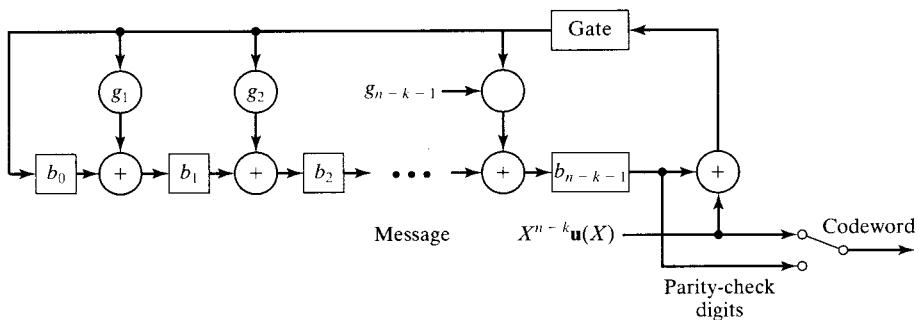


FIGURE 5.1: Encoding circuit for an (n, k) cyclic code with generator polynomial $\mathbf{g}(X) = 1 + g_1 X + \dots + g_{n-k-1} X^{n-k-1} + X^{n-k}$.

- Step 1.** Turn on the gate. The k information digits u_0, u_1, \dots, u_{k-1} [or $\mathbf{u}(X) = u_0 + u_1 X + \dots + u_{k-1} X^{k-1}$ in polynomial form] are shifted into the circuit and simultaneously into the communication channel. Shifting the message $\mathbf{u}(X)$ into the circuit from the front end is equivalent to premultiplying $\mathbf{u}(X)$ by X^{n-k} . As soon as the complete message has entered the circuit, the $n - k$ digits in the register form the remainder, and thus they are the parity-check digits.
- Step 2.** Break the feedback connection by turning off the gate.
- Step 3.** Shift the parity-check digits out and send them into the channel. These $n - k$ parity-check digits $b_0, b_1, \dots, b_{n-k-1}$, together with the k information digits, form a complete codeword.

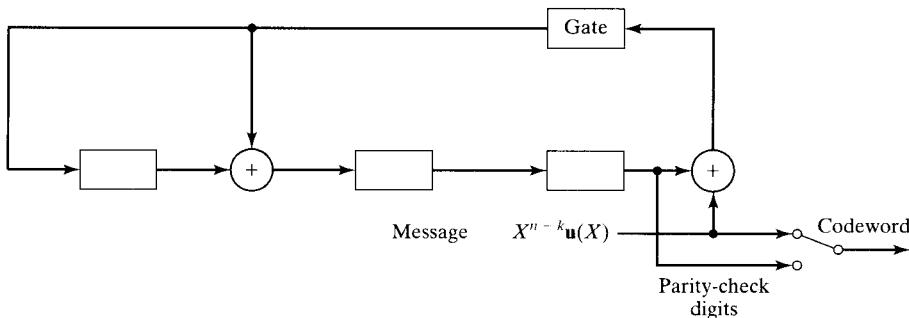
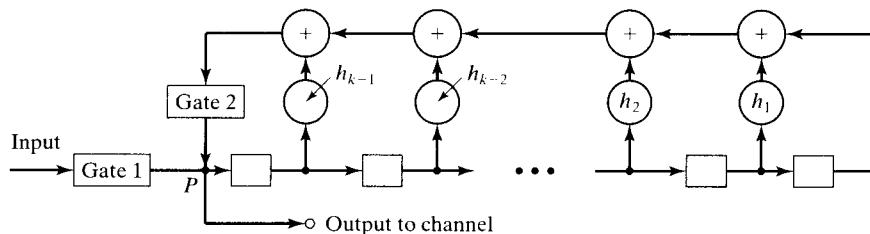
EXAMPLE 5.5

Consider the $(7, 4)$ cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$. The encoding circuit based on $\mathbf{g}(X)$ is shown in Figure 5.2. Suppose that the message $\mathbf{u} = (1011)$ is to be encoded. As the message digits are shifted into the register the contents of the register change as follows:

Input	Register contents
	0 0 0 (initial state)
1	1 1 0 (first shift)
1	1 0 1 (second shift)
0	1 0 0 (third shift)
1	1 0 0 (fourth shift)

After four shifts, the contents of the register are (100) . Thus, the complete codeword is (1001011) , and the code polynomial is $1 + X^3 + X^5 + X^6$.

Encoding of a cyclic code can also be accomplished by using its parity polynomial $\mathbf{h}(X) = h_0 + h_1 X + \dots + h_k X^k$. Let $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ be a codeword. We have shown in Section 5.2 that the components of \mathbf{v} satisfy the $n - k$ equalities

FIGURE 5.2: Encoder for the $(7, 4)$ cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$.FIGURE 5.3: Encoding circuit for an (n, k) cyclic code based on the parity polynomial $\mathbf{h}(X) = 1 + h_1X + \dots + X^k$.

of (5.12). Since $h_k = 1$, the equalities of (5.12) can be put into the following form:

$$v_{n-k-j} = \sum_{i=0}^{k-1} h_i v_{n-i-j} \quad \text{for } 1 \leq j \leq n-k \quad (5.18)$$

which is known as a *difference equation*. For a cyclic code in systematic form, the components $v_{n-k}, v_{n-k+1}, \dots, v_{n-1}$ of each codeword are the information digits. Given these k information digits, (5.18) is a rule for determining the $n - k$ parity-check digits, $v_0, v_1, \dots, v_{n-k-1}$. An encoding circuit based on (5.18) is shown in Figure 5.3. The feedback connections are based on the coefficients of the parity polynomial $\mathbf{h}(X)$. (Note that $h_0 = h_k = 1$.) The encoding operation can be described in the following steps:

- Step 1.** Initially, gate 1 is turned on and gate 2 is turned off. The k information digits $\mathbf{u}(X) = u_0 + u_1X + \dots + u_{k-1}X^{k-1}$ are shifted into the register and the communication channel simultaneously.
- Step 2.** As soon as the k information digits have entered the shift register, gate 1 is turned off and gate 2 is turned on. The first parity-check digit,

$$\begin{aligned} v_{n-k-1} &= h_0 v_{n-1} + h_1 v_{n-2} + \dots + h_{k-1} v_{n-k} \\ &= u_{k-1} + h_1 u_{k-2} + \dots + h_{k-1} u_0, \end{aligned}$$

is formed and appears at point P .

Step 3. The register is shifted once. The first parity-check digit is shifted into the channel and is also shifted into the register. Now, the second parity-check digit,

$$\begin{aligned} v_{n-k-2} &= h_0 v_{n-2} + h_1 v_{n-3} + \cdots + h_{k-1} v_{n-k-1} \\ &= u_{k-2} + h_1 u_{k-3} + \cdots + h_{k-2} u_0 + h_{k-1} v_{n-k-1}, \end{aligned}$$

is formed at P .

Step 4. Step 3 is repeated until $n - k$ parity-check digits have been formed and shifted into the channel. Then, gate 1 is turned on and gate 2 is turned off. The next message is now ready to be shifted into the register.

The preceding encoding circuit employs a k -stage shift register. Comparing the two encoding circuits presented in this section, we can make the following remark: for codes with more parity-check digits than message digits, the k -stage encoding circuit is more economical; otherwise, the $(n - k)$ -stage encoding circuit is preferable.

EXAMPLE 5.6

The parity polynomial of the $(7, 4)$ cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$ is

$$\mathbf{h}(X) = \frac{X^7 + 1}{1 + X + X^3} = 1 + X + X^2 + X^4.$$

The encoding circuit based on $\mathbf{h}(X)$ is shown in Figure 5.4. Each codeword is of the form $\mathbf{v} = (v_0, v_1, v_2, v_3, v_4, v_5, v_6)$, where v_3, v_4, v_5 , and v_6 are message digits, and v_0, v_1 , and v_2 are parity-check digits. The difference equation that determines the parity-check digits is

$$\begin{aligned} v_{3-j} &= 1 \cdot v_{7-j} + 1 \cdot v_{6-j} + 1 \cdot v_{5-j} + 0 \cdot v_{4-j} \\ &= v_{7-j} + v_{6-j} + v_{5-j} \quad \text{for } 1 \leq j \leq 3. \end{aligned}$$

Suppose that the message to be encoded is $(1\ 0\ 1\ 1)$. Then, $v_3 = 1, v_4 = 0, v_5 = 1, v_6 = 1$. The first parity-check digit is

$$v_2 = v_6 + v_5 + v_4 = 1 + 1 + 0 = 0.$$

The second parity-check digit is

$$v_1 = v_5 + v_4 + v_3 = 1 + 0 + 1 = 0.$$

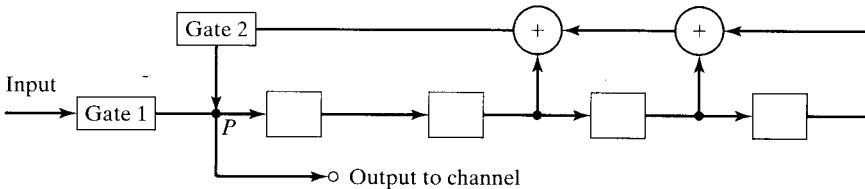


FIGURE 5.4: Encoding circuit for the $(7, 4)$ cyclic code based on its parity polynomial $\mathbf{h}(X) = 1 + X + X^2 + X^4$.

The third parity-check digit is

$$v_0 = v_4 + v_3 + v_2 = 0 + 1 + 0 = 1.$$

Thus, the codeword that corresponds to the message (1 0 1 1) is (1 0 0 1 0 1 1).

5.4 SYNDROME COMPUTATION AND ERROR DETECTION

Suppose that a codeword is transmitted. Let $\mathbf{r} = (r_0, r_1, \dots, r_{n-1})$ be the received vector. Because of the channel noise, the received vector may not be the same as the transmitted codeword. In the decoding of a linear code, the first step is to compute the syndrome $\mathbf{s} = \mathbf{r} \cdot \mathbf{H}^T$, where \mathbf{H} is the parity-check matrix. If the syndrome is zero, \mathbf{r} is a codeword, and the decoder accepts \mathbf{r} as the transmitted codeword. If the syndrome is not identical to zero, \mathbf{r} is not a codeword, and the presence of errors has been detected.

We have shown that for a linear systematic code the syndrome is simply the vector sum of the received parity-check digits and the parity-check digits recomputed from the received information digits. For a cyclic code in systematic form, the syndrome can be computed easily. The received vector \mathbf{r} is treated as a polynomial of degree $n - 1$ or less,

$$\mathbf{r}(X) = r_0 + r_1 X + r_2 X^2 + \dots + r_{n-1} X^{n-1}.$$

Dividing $\mathbf{r}(X)$ by the generator polynomial $\mathbf{g}(X)$, we obtain

$$\mathbf{r}(X) = \mathbf{a}(X)\mathbf{g}(X) + \mathbf{s}(X). \quad (5.19)$$

The remainder $\mathbf{s}(X)$ is a polynomial of degree $n - k - 1$ or less. The $n - k$ coefficients of $\mathbf{s}(X)$ form the syndrome \mathbf{s} . It is clear from Theorem 5.4 that $\mathbf{s}(X)$ is identical to zero if and only if the received polynomial $\mathbf{r}(X)$ is a code polynomial. Hereafter, we will simply call $\mathbf{s}(X)$ the syndrome. The syndrome can be computed with a division circuit as shown in Figure 5.5, which is identical to the $(n - k)$ -stage encoding circuit, except that the received polynomial $\mathbf{r}(X)$ is shifted into the register from the left end. The received polynomial $\mathbf{r}(X)$ is shifted into the register with all stages initially set to 0. As soon as the entire $\mathbf{r}(X)$ has been shifted into the register, the contents in the register form the syndrome $\mathbf{s}(X)$.

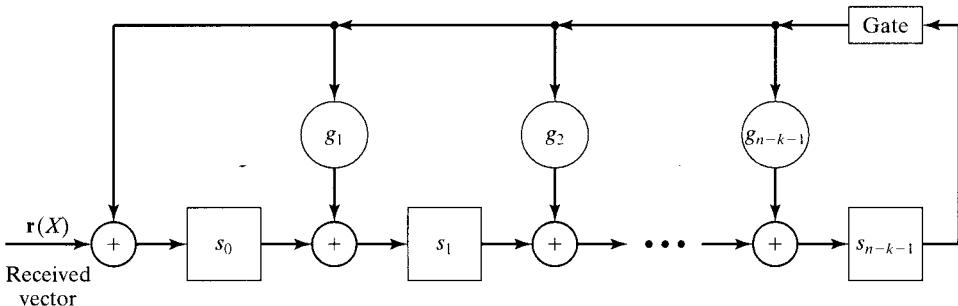


FIGURE 5.5: An $(n - k)$ -stage syndrome circuit with input from the left end.

Because of the cyclic structure of the code, the syndrome $s(X)$ has the following property.

THEOREM 5.8 Let $s(X)$ be the syndrome of a received polynomial $r(X) = r_0 + r_1X + \dots + r_{n-1}X^{n-1}$. Then, the remainder $s^{(1)}(X)$ resulting from dividing $Xs(X)$ by the generator polynomial $g(X)$ is the syndrome of $r^{(1)}(X)$, which is a cyclic shift of $r(X)$.

Proof. It follows from (5.1) that $r(X)$ and $r^{(1)}(X)$ satisfy the following relationship:

$$Xr(X) = r_{n-1}(X^n + 1) + r^{(1)}(X). \quad (5.20)$$

Rearranging (5.20), we have

$$r^{(1)}(X) = r_{n-1}(X^n + 1) + Xr(X). \quad (5.21)$$

Dividing both sides of (5.21) by $g(X)$ and using the fact that $X^n + 1 = g(X)h(X)$, we obtain

$$c(X)g(X) + \rho(X) = r_{n-1}g(X)h(X) + X[a(X)g(X) + s(X)], \quad (5.22)$$

where $\rho(X)$ is the remainder resulting from dividing $r^{(1)}(X)$ by $g(X)$. Then, $\rho(X)$ is the syndrome of $r^{(1)}(X)$.

Rearranging (5.22), we obtain the following relationship between $\rho(X)$ and $Xs(X)$:

$$Xs(X) = [c(X) + r_{n-1}h(X) + Xa(X)]g(X) + \rho(X). \quad (5.23)$$

From (5.23) we see that $\rho(X)$ is also the remainder resulting from dividing $Xs(X)$ by $g(X)$. Therefore, $\rho(X) = s^{(1)}(X)$. This completes the proof. **Q.E.D.**

It follows from Theorem 5.8 that the remainder $s^{(i)}(X)$ resulting from dividing $X^i s(X)$ by the generator polynomial $g(X)$ is the syndrome of $r^{(i)}(X)$, which is the i th cyclic shift of $r(X)$. This property is useful in decoding cyclic codes. We can obtain the syndrome $s^{(1)}(X)$ of $r^{(1)}(X)$ by shifting (or clocking) the syndrome register once with $s(X)$ as the initial contents and with the input gate disabled. This is because shifting the syndrome register once with $s(X)$ as the initial contents is equivalent to dividing $Xs(X)$ by $g(X)$. Thus, after the shift, the register contains $s^{(1)}(X)$. To obtain the syndrome $s^{(i)}(X)$ of $r^{(i)}(X)$, we simply shift the syndrome register i times with $s(X)$ as the initial contents.

EXAMPLE 5.7

A syndrome circuit for the $(7, 4)$ cyclic code generated by $g(X) = 1 + X + X^3$ is shown in Figure 5.6. Suppose that the received vector is $r = (0\ 0\ 1\ 0\ 1\ 1\ 0)$. The syndrome of r is $s = (1\ 0\ 1)$. Table 5.3 shows the contents in the register as the received vector is shifted into the circuit. At the end of the seventh shift, the register contains the syndrome $s = (1\ 0\ 1)$. If the register is shifted once more with the input gate disabled, the new contents will be $s^{(1)} = (1\ 0\ 0)$, which is the syndrome of $r^{(1)}(X) = (0\ 0\ 0\ 1\ 0\ 1\ 1)$, a cyclic shift of r .

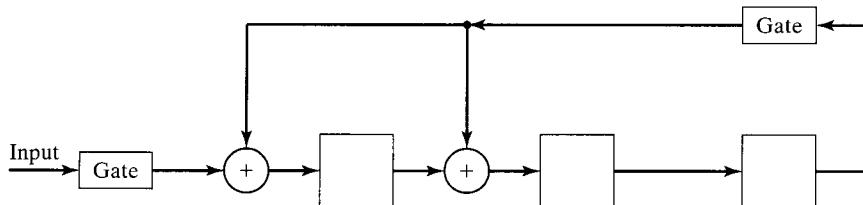


FIGURE 5.6: Syndrome circuit for the $(7, 4)$ cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$.

TABLE 5.3: Contents of the syndrome register shown in Figure 5.6 with $\mathbf{r} = (0010110)$ as input.

Shift	Input	Register contents
		000 (initial state)
1	0	000
2	1	100
3	1	110
4	0	011
5	1	011
6	0	111
7	0	101 (syndrome \mathbf{s})
8	—	100 (syndrome $\mathbf{s}^{(1)}$)
9	—	010 (syndrome $\mathbf{s}^{(2)}$)

We may shift the received vector $\mathbf{r}(X)$ into the syndrome register from the right end, as shown in Figure 5.7; however, after the entire $\mathbf{r}(X)$ has been shifted into the register, the contents in the register do not form the syndrome of $\mathbf{r}(X)$; rather, they form the syndrome $\mathbf{s}^{(n-k)}(X)$ of $\mathbf{r}^{(n-k)}(X)$, which is the $(n - k)$ th cyclic shift of $\mathbf{r}(X)$. To show this, we notice that shifting $\mathbf{r}(X)$ from the right end is equivalent to premultiplying $\mathbf{r}(X)$ by X^{n-k} . When the entire $\mathbf{r}(X)$ has entered the register, the register contains the remainder $\rho(X)$ resulting from dividing $X^{n-k}\mathbf{r}(X)$ by the generator polynomial $\mathbf{g}(X)$. Thus, we have

$$X^{n-k}\mathbf{r}(X) = \mathbf{a}(X)\mathbf{g}(X) + \rho(X). \quad (5.24)$$

It follows from (5.1) that $\mathbf{r}(X)$ and $\mathbf{r}^{(n-k)}(X)$ satisfy the following relation:

$$X^{n-k}\mathbf{r}(X) = \mathbf{b}(X)(X^n + 1) + \mathbf{r}^{(n-k)}(X). \quad (5.25)$$

Combining (5.24) and (5.25) and using the fact that $X^n + 1 = \mathbf{g}(X)\mathbf{h}(X)$, we have

$$\mathbf{r}^{(n-k)}(X) = [\mathbf{b}(X)\mathbf{h}(X) + \mathbf{a}(X)]\mathbf{g}(X) + \rho(X).$$

This equation says that when $\mathbf{r}^{(n-k)}(X)$ is divided by $\mathbf{g}(X)$, $\rho(X)$ is also the remainder. Therefore, $\rho(X)$ is indeed the syndrome of $\mathbf{r}^{(n-k)}(X)$.

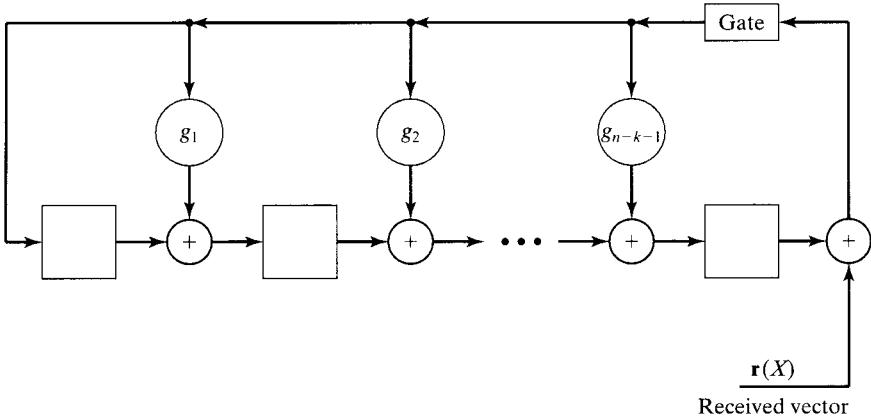


FIGURE 5.7: An $(n - k)$ -stage syndrome circuit with input from the right end.

Let $\mathbf{v}(X)$ be the transmitted codeword, and let $\mathbf{e}(X) = e_0 + e_1 X + \cdots + e_{n-1} X^{n-1}$ be the error pattern. Then, the received polynomial is

$$\mathbf{r}(X) = \mathbf{v}(X) + \mathbf{e}(X). \quad (5.26)$$

Because $\mathbf{v}(X)$ is a multiple of the generator polynomial $\mathbf{g}(X)$, combining (5.19) and (5.26), we have the following relationship between the error pattern and the syndrome:

$$\mathbf{e}(X) = [\mathbf{a}(X) + \mathbf{b}(X)]\mathbf{g}(X) + \mathbf{s}(X), \quad (5.27)$$

where $\mathbf{b}(X)\mathbf{g}(X) = \mathbf{v}(X)$. This shows that the syndrome is equal to the remainder resulting from dividing the error pattern by the generator polynomial. The syndrome can be computed from the received vector; however, the error pattern $\mathbf{e}(X)$ is unknown to the decoder. Therefore, the decoder has to estimate $\mathbf{e}(X)$ based on the syndrome $\mathbf{s}(X)$. If $\mathbf{e}(X)$ is a coset leader in the standard array and if table-lookup decoding is used, $\mathbf{e}(X)$ can correctly be determined from the syndrome.

From (5.27) we see that $\mathbf{s}(X)$ is identical to zero if and only if either the error pattern $\mathbf{e}(X) = 0$ or $\mathbf{e}(X)$ is identical to a codeword. If $\mathbf{e}(X)$ is identical to a code polynomial, $\mathbf{e}(X)$ is an undetectable error pattern. Cyclic codes are very effective for detecting errors, random or burst. The error-detection circuit is simply a syndrome circuit with an OR gate whose inputs are the syndrome digits. If the syndrome is not zero, the output of the OR gate is 1, and the presence of errors has been detected.

Now, we investigate the error-detecting capability of an (n, k) cyclic code. Suppose that the error pattern $\mathbf{e}(X)$ is a burst of length $n - k$ or less (i.e., errors are confined to $n - k$ or fewer consecutive positions). Then, we can express $\mathbf{e}(X)$ in the following form:

$$\mathbf{e}(X) = X^j \mathbf{B}(X),$$

where $0 \leq j \leq n - 1$, and $\mathbf{B}(X)$ is a polynomial of degree $n - k - 1$ or less. Because the degree of $\mathbf{B}(X)$ is less than the degree of the generator polynomial $\mathbf{g}(X)$, $\mathbf{B}(X)$ is not divisible by $\mathbf{g}(X)$. Since $\mathbf{g}(X)$ is a factor of $X^n + 1$, and X is not a factor of $\mathbf{g}(X)$, $\mathbf{g}(X)$ and X^j must be relatively prime. Therefore, $\mathbf{e}(X) = X^j \mathbf{B}(X)$ is not

divisible by $\mathbf{g}(X)$. As a result, the syndrome caused by $\mathbf{e}(X)$ is not equal to zero. This implies that an (n, k) cyclic code is capable of detecting any error burst of length $n - k$ or less. For a cyclic code, an error pattern with errors confined to i high-order positions and $l - i$ low-order positions is also regarded as a burst of length l or less. Such a burst is called an *end-around* burst. For example,

$$\mathbf{e} = \left(\begin{array}{cccccccccc} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array} \right) \quad \begin{array}{ccccccc} \leftarrow & \leftarrow & \leftarrow & & & & \\ & & & & \uparrow & \rightarrow & \rightarrow & \rightarrow \end{array}$$

is an end-around burst of length 7. An (n, k) cyclic code is also capable of detecting all the end-around error bursts of length $n - k$ or less (the proof of this is left as a problem). Summarizing the preceding results, we have the following property:

THEOREM 5.9 An (n, k) cyclic code is capable of detecting any error burst of length $n - k$ or less, including the end-around bursts.

In fact, a large percentage of error bursts of length $n - k + 1$ or longer can be detected. Consider the bursts of length $n - k + 1$ starting from the i th digit position and ending at the $(i + n - k)$ th digit position (i.e., errors are confined to digits $e_i, e_{i+1}, \dots, e_{i+n-k}$, with $e_i = e_{i+n-k} = 1$). There are 2^{n-k-1} such bursts. Among these bursts, the only one that cannot be detected is

$$\mathbf{e}(X) = X^i \mathbf{g}(X).$$

Therefore, the fraction of undetectable bursts of length $n - k + 1$ starting from the i th digit position is $2^{-(n-k-1)}$. This fraction applies to bursts of length $n - k + 1$ starting from any digit position (including the end-around case). Therefore, we have the following result:

THEOREM 5.10 The fraction of undetectable bursts of length $n - k + 1$ is $2^{-(n-k-1)}$.

For $l > n - k + 1$, there are 2^{l-2} bursts of length l starting from the i th digit position and ending at the $(i + l - 1)$ th digit position. Among these bursts, the undetectable ones must be of the following form:

$$\mathbf{e}(X) = X^i \mathbf{a}(X) \mathbf{g}(X)$$

where $\mathbf{a}(X) = a_0 + a_1 X + \cdots + a_{l-(n-k)-1} X^{l-(n-k)-1}$, with $a_0 = a_{l-(n-k)-1} = 1$. The number of such bursts is $2^{l-(n-k)-2}$. Therefore, the fraction of undetectable bursts of length l starting from the i th digit position is $2^{-(n-k)}$. Again, this fraction applies to bursts of length l starting from any digit position (including the end-around case), which leads to the following conclusion:

THEOREM 5.11 For $l > n - k + 1$, the fraction of undetectable error bursts of length l is $2^{-(n-k)}$.

The preceding analysis shows that cyclic codes are very effective for burst-error detection.

EXAMPLE 5.8

The $(7, 4)$ cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$ has a minimum distance of 3. It is capable of detecting any combination of two or fewer random errors or any burst of length 3 or less. It also detects many bursts of length greater than 3.

5.5 DECODING OF CYCLIC CODES

Decoding of cyclic codes consists of the same three steps as for decoding linear codes: syndrome computation, association of the syndrome with an error pattern, and error correction. It was shown in Section 5.4 that syndromes for cyclic codes can be computed with a division circuit whose complexity is linearly proportional to the number of parity-check digits (i.e., $n - k$). The error-correction step is simply adding (modulo-2) the error pattern to the received vector. This addition can be performed with a single EXCLUSIVE-OR gate if correction is carried out serially (i.e., one digit at a time); n EXCLUSIVE-OR gates are required if correction is carried out in parallel, as shown in Figure 3.8. The association of the syndrome with an error pattern can be completely specified by a decoding table. A straightforward approach to the design of a decoding circuit is via a combinational logic circuit that implements the table-lookup procedure; however, the limit to this approach is that the complexity of the decoding circuit tends to grow exponentially with the code length and with the number of errors that are going to be corrected. Cyclic codes have considerable algebraic and geometric properties. If these properties are properly used, decoding circuits can be simplified.

The cyclic structure of a cyclic code allows us to decode a received vector $\mathbf{r}(X) = r_0 + r_1X + r_2X^2 + \dots + r_{n-1}X^{n-1}$ serially. The received digits are decoded one at a time, and each digit is decoded with the same circuitry. As soon as the syndrome has been computed the decoding circuit checks whether the syndrome $\mathbf{s}(X)$ corresponds to a correctable error pattern $\mathbf{e}(X) = e_0 + e_1X + \dots + e_{n-1}X^{n-1}$ with an error at the highest-order position X^{n-1} (i.e., $e_{n-1} = 1$). If $\mathbf{s}(X)$ does not correspond to an error pattern with $e_{n-1} = 1$, the received polynomial (stored in a buffer register) and the syndrome register are cyclically shifted once simultaneously. Thus, we obtain $\mathbf{r}^{(1)}(X) = r_{n-1} + r_0X + \dots + r_{n-2}X^{n-1}$, and the new contents in the syndrome register form the syndrome $\mathbf{s}^{(1)}(X)$ of $\mathbf{r}^{(1)}(X)$. Now, the second digit r_{n-2} of $\mathbf{r}(X)$ becomes the first digit of $\mathbf{r}^{(1)}(X)$. The same decoding circuit will check whether $\mathbf{s}^{(1)}(X)$ corresponds to an error pattern with an error at location X^{n-1} .

If the syndrome $\mathbf{s}(X)$ of $\mathbf{r}(X)$ does correspond to an error pattern with an error at location X^{n-1} (i.e., $e_{n-1} = 1$), the first received digit r_{n-1} is an erroneous digit, and it must be corrected. The correction is carried out by taking the sum $r_{n-1} \oplus e_{n-1}$. This correction results in a modified received polynomial, denoted by $\mathbf{r}_1(X) = r_0 + r_1X + \dots + r_{n-2}X^{n-2} + (r_{n-1} \oplus e_{n-1})X^{n-1}$. The effect of the error digit e_{n-1} on the syndrome is then removed from the syndrome $\mathbf{s}(X)$, by adding the syndrome of $\mathbf{e}'(X) = X^{n-1}$ to $\mathbf{s}(X)$. This sum is the syndrome of the modified received polynomial $\mathbf{r}_1(X)$. Now, $\mathbf{r}_1(X)$ and the syndrome register are cyclically shifted once simultaneously. This shift results in a received polynomial $\mathbf{r}_1^{(1)}(X) = (r_{n-1} \oplus e_{n-1}) + r_0X + \dots + r_{n-2}X^{n-1}$. The syndrome $\mathbf{s}_1^{(1)}(X)$ of $\mathbf{r}_1^{(1)}(X)$ is the remainder resulting from dividing $X[\mathbf{s}(X) + X^{n-1}]$ by the generator polynomial

$g(X)$. Because the remainders resulting from dividing $Xs(X)$ and X^n by $g(X)$ are $s^{(1)}(X)$ and 1, respectively, we have

$$s_1^{(1)}(X) = s^{(1)}(X) + 1.$$

Therefore, if 1 is added to the left end of the syndrome register while it is shifted, we obtain $s_1^{(1)}(X)$. The decoding circuitry proceeds to decode the received digit r_{n-2} . The decoding of r_{n-2} and the other received digits is identical to the decoding of r_{n-1} . Whenever an error is detected and corrected, its effect on the syndrome is removed. The decoding stops after a total of n shifts. If $e(X)$ is a correctable error pattern, the contents of the syndrome register should be zero at the end of the decoding operation, and the received vector $r(X)$ has been correctly decoded. If the syndrome register does not contain all 0's at the end of the decoding process, an uncorrectable error pattern has been detected.

A general decoder for an (n, k) cyclic code is shown in Figure 5.8. It consists of three major parts: (1) a syndrome register, (2) an error-pattern detector, and (3) a buffer register to hold the received vector. The received polynomial is shifted into the syndrome register from the left end. To remove the effect of an error digit on the syndrome, we simply feed the error digit into the shift register from the left end through an EXCLUSIVE-OR gate. The decoding operation is as follows:

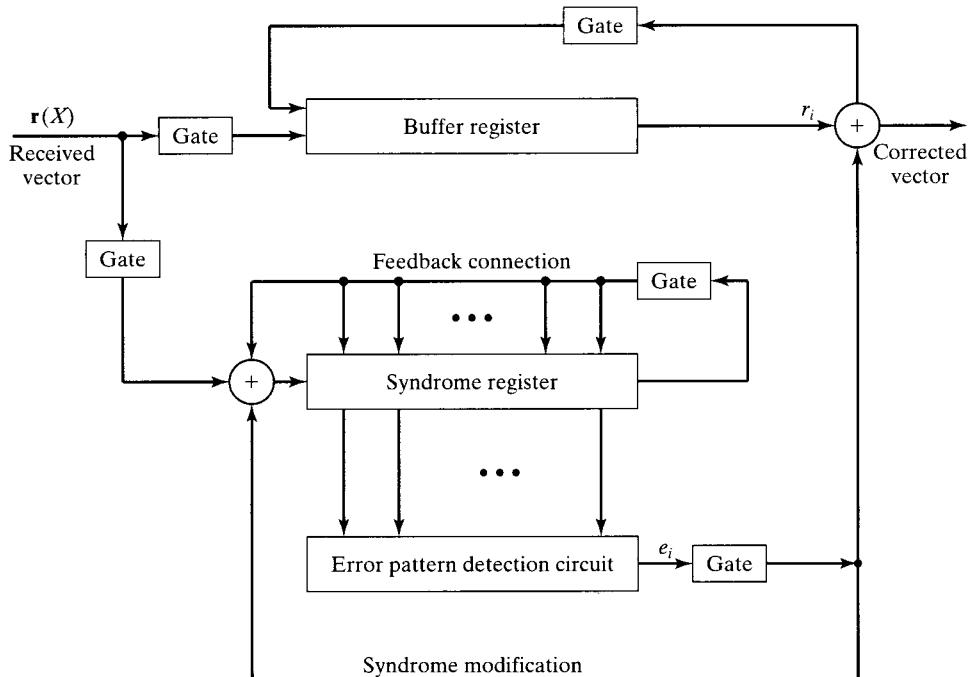


FIGURE 5.8: General cyclic code decoder with received polynomial $r(X)$ shifted into the syndrome register from the left end.

- Step 1.** The syndrome is formed by shifting the entire received vector into the syndrome register. The received vector is simultaneously stored in the buffer register.
- Step 2.** The syndrome is read into the detector and is tested for the corresponding error pattern. The detector is a combinational logic circuit that is designed in such a way that its output is 1 if and only if the syndrome in the syndrome register corresponds to a correctable error pattern with an error at the highest-order position X^{n-1} . That is, if a 1 appears at the output of the detector, the received symbol in the rightmost stage of the buffer register is assumed to be erroneous and must be corrected; if a 0 appears at the output of the detector, the received symbol at the rightmost stage of the buffer register is assumed to be error-free, and no correction is necessary. Thus, the output of the detector is the estimated error value for the symbol to come out of the buffer.
- Step 3.** The first received symbol is read out of the buffer. At the same time, the syndrome register is shifted once. If the first received symbol is detected to be an erroneous symbol, it is then corrected by the output of the detector. The output of the detector is also fed back to the syndrome register to modify the syndrome (i.e., to remove the error effect from the syndrome). This operation results in a new syndrome, which corresponds to the altered received vector shifted one place to the right.
- Step 4.** The new syndrome formed in step 3 is used to detect whether the second received symbol (now at the rightmost stage of the buffer register) is an erroneous symbol. The decoder repeats steps 2 and 3. The second received symbol is corrected in exactly the same manner as the first received symbol was corrected.
- Step 5.** The decoder decodes the received vector symbol by symbol in the manner outlined until the entire received vector is read out of the buffer register.

The preceding decoder is known as a Meggitt decoder [11], which applies in principle to any cyclic code. But whether it is practical depends entirely on its error-pattern detection circuit. In some cases the error-pattern detection circuits are simple. Several of these cases are discussed in subsequent chapters.

EXAMPLE 5.9

Consider the decoding of the (7, 4) cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$. This code has a minimum distance of 3 and is capable of correcting any single error over a block of seven digits. There are seven single-error patterns. These seven error patterns and the all-zero vector form all the coset leaders of the decoding table. Thus, they form all the correctable error patterns. Suppose that the received polynomial $\mathbf{r}(X) = r_0 + r_1X + r_2X^2 + r_3X^3 + r_4X^4 + r_5X^5 + r_6X^6$ is shifted into the syndrome register from the left end. The seven single-error patterns and their corresponding syndrome are listed in Table 5.4.

TABLE 5.4: Error patterns and their syndromes with the received polynomial $r(X)$ shifted into the syndrome register from the left end.

Error pattern $e(X)$	Syndrome $s(X)$	Syndrome vector (s_0, s_1, s_2)
$e_6(X) = X^6$	$s(X) = 1 + X^2$	(1 0 1)
$e_5(X) = X^5$	$s(X) = 1 + X + X^2$	(1 1 1)
$e_4(X) = X^4$	$s(X) = X + X^2$	(0 1 1)
$e_3(X) = X^3$	$s(X) = 1 + X$	(1 1 0)
$e_2(X) = X^2$	$s(X) = X^2$	(0 0 1)
$e_1(X) = X^1$	$s(X) = X$	(0 1 0)
$e_0(X) = X^0$	$s(X) = 1$	(1 0 0)

We see that $e_6(X) = X^6$ is the only error pattern with an error at location X^6 . When this error pattern occurs, the syndrome in the syndrome register will be (1 0 1) after the entire received polynomial $r(X)$ has entered the syndrome register. The detection of this syndrome indicates that r_6 is an erroneous digit and must be corrected. Suppose that the single error occurs at location X^i [i.e., $e_i(X) = X^i$] for $0 \leq i < 6$. After the entire received polynomial has been shifted into the syndrome register, the syndrome in the register will not be (1 0 1); however, after another $6 - i$ shifts, the contents in the syndrome register will be (1 0 1), and the next received digit to come out of the buffer register will be the erroneous digit. Therefore, only the syndrome (1 0 1) needs to be detected, and this can be accomplished with a single three-input AND gate. The complete decoding circuit is shown in Figure 5.9. Figure 5.10 illustrates the decoding process. Suppose that the codeword $v = (1001011)$ [or $v(X) = 1 + X^3 + X^5 + X^6$] is transmitted and $r = (1011011)$ [or $r(X) = 1 + X^2 + X^3 + X^5 + X^6$] is received. A single error occurs at location X^2 . When the entire received polynomial has been shifted into the syndrome and buffer registers, the syndrome register contains (0 0 1). In Figure 5.10, the contents in the syndrome register and the contents in the buffer register are recorded after each shift. Also, there is a pointer to indicate the error location after each shift. We see that after four more shifts the contents in the syndrome register are (1 0 1), and the erroneous digit r_2 is the next digit to come out from the buffer register.

The (7, 4) cyclic code considered in Example 5.9 is the same code considered in Example 3.9. Comparing the decoding circuit shown in Figure 3.9 with the decoding circuit shown in Figure 5.9, we see that the circuit shown in Figure 5.9 is simpler than the circuit shown in Figure 3.9. Thus, the cyclic structure does simplify the decoding circuit; however, the circuit shown in Figure 5.9 takes a longer time to decode a received vector because the decoding is carried out serially. In general, there is a trade-off between speed and simplicity, as they cannot be achieved at the same time.

The Meggitt decoder described decodes a received polynomial $r(X) = r_0 + r_1X + \dots + r_{n-1}X^{n-1}$ from the highest-order received digit r_{n-1} to the lowest-order received digit r_0 . After decoding the received digit r_i , both the buffer and syndrome

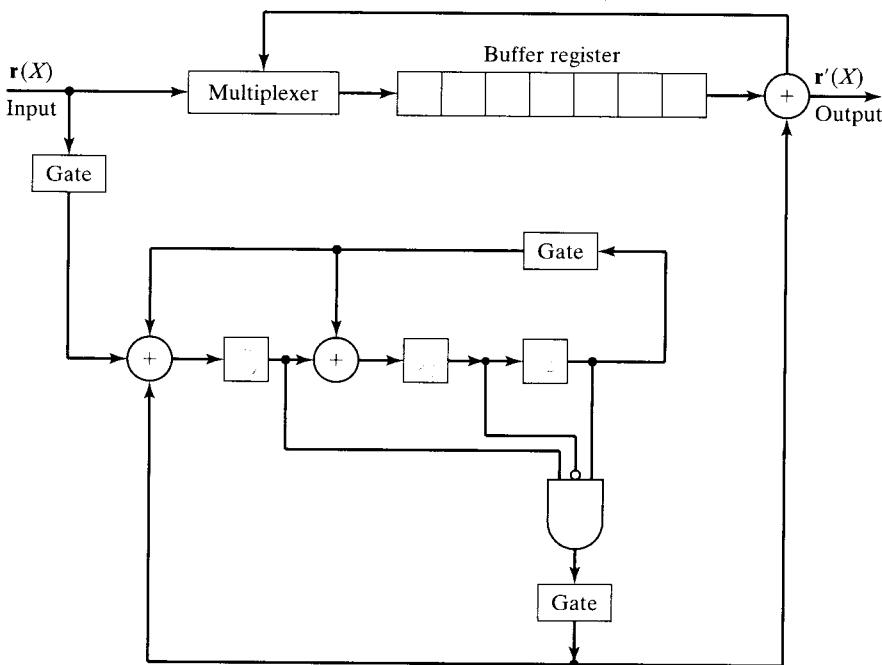


FIGURE 5.9: Decoding circuit for the $(7, 4)$ cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$.

registers are shifted once to the right. The next received digit to be decoded is r_{i-1} . It is possible to implement a Meggitt decoder to decode a received polynomial in the reverse order (i.e., to decode a received polynomial from the lowest-order received digit r_0 to the highest-order received digit r_{n-1}). After decoding the received digit r_i , both the buffer and syndrome registers are shifted once to the left. The next received digit to be decoded is r_{i+1} . The details of this decoding of a received polynomial in reverse order are left as an exercise.

To decode a cyclic code, the received polynomial $\mathbf{r}(X)$ may be shifted into the syndrome register from the right end for computing the syndrome. When $\mathbf{r}(X)$ has been shifted into the syndrome register, the register contains $\mathbf{s}^{(n-k)}(X)$, which is the syndrome of $\mathbf{r}^{(n-k)}(X)$, the $(n - k)$ th cyclic shift of $\mathbf{r}(X)$. If $\mathbf{s}^{(n-k)}(X)$ corresponds to an error pattern $\mathbf{e}(X)$ with $e_{n-1} = 1$, the highest-order digit r_{n-1} of $\mathbf{r}(X)$ is erroneous and must be corrected. In $\mathbf{r}^{(n-k)}(X)$, the digit r_{n-1} is at the location X^{n-k-1} . When r_{n-1} is corrected, the error effect must be removed from $\mathbf{s}^{(n-k)}(X)$. The new syndrome, denoted by $\mathbf{s}_1^{(n-k)}(X)$, is the sum of $\mathbf{s}^{(n-k)}(X)$ and the remainder $\rho(X)$ resulting from dividing X^{n-k-1} by the generator polynomial $\mathbf{g}(X)$. Because the degree of X^{n-k-1} is less than the degree of $\mathbf{g}(X)$,

$$\rho(X) = X^{n-k-1}.$$

Therefore,

$$\mathbf{s}_1^{(n-k)}(X) = \mathbf{s}^{(n-k)}(X) + X^{n-k-1},$$

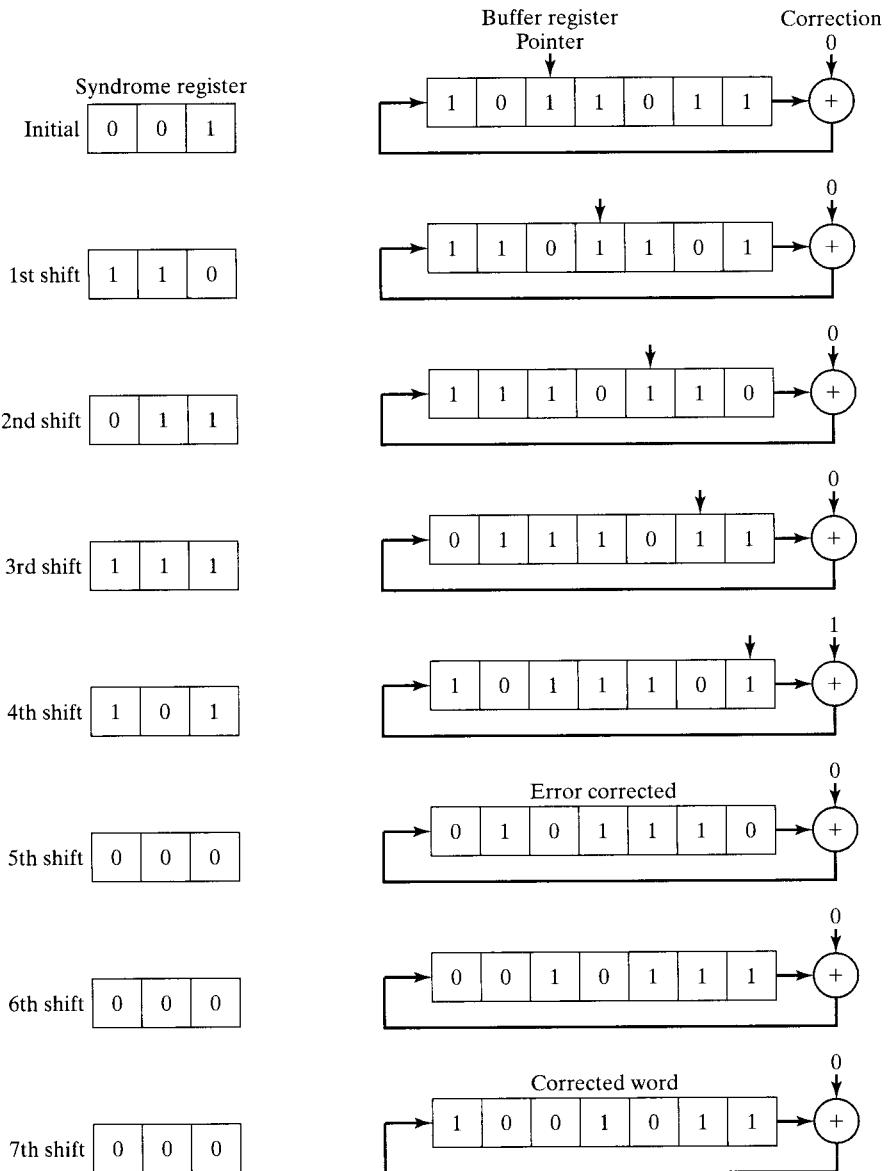


FIGURE 5.10: Error-correction process of the circuit shown in Figure 5.9.

which indicates that the effect of an error at the location X^{n-1} on the syndrome can be removed by feeding the error digit into the syndrome register from the right end through an EXCLUSIVE-OR gate, as shown in Figure 5.11. The decoding process of the decoder shown in Figure 5.11 is identical to the decoding process of the decoder shown in Figure 5.8.

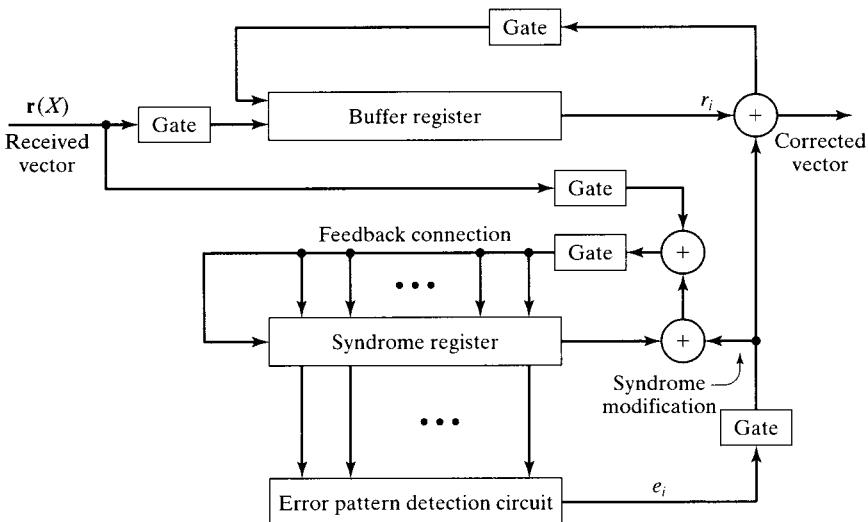


FIGURE 5.11: General cyclic code decoder with received polynomial $\mathbf{r}(X)$ shifted into the syndrome register from the right end.

EXAMPLE 5.10

Again, we consider the decoding of the $(7, 4)$ cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$. Suppose that the received polynomial $\mathbf{r}(X)$ is shifted into the syndrome register from the right end. The seven single-error patterns and their corresponding syndromes are listed in Table 5.5.

We see that only when $\mathbf{e}(X) = X^6$ occurs, the syndrome is $(0\ 0\ 1)$ after the entire polynomial $\mathbf{r}(X)$ has been shifted into the syndrome register. If the single error occurs at the location X^i with $i \neq 6$, the syndrome in the register will not be

TABLE 5.5: Error patterns and their syndromes with the received polynomial $\mathbf{r}(X)$ shifted into the syndrome register from the right end.

Error pattern $\mathbf{e}(X)$	Syndrome $\mathbf{s}^{(3)}(X)$	Syndrome vector (s_0, s_1, s_2)
$\mathbf{e}(X) = X^6$	$\mathbf{s}^{(3)}(X) = X^2$	$(0\ 0\ 1)$
$\mathbf{e}(X) = X^5$	$\mathbf{s}^{(3)}(X) = X$	$(0\ 1\ 0)$
$\mathbf{e}(X) = X^4$	$\mathbf{s}^{(3)}(X) = 1$	$(1\ 0\ 0)$
$\mathbf{e}(X) = X^3$	$\mathbf{s}^{(3)}(X) = 1 + X^2$	$(1\ 0\ 1)$
$\mathbf{e}(X) = X^2$	$\mathbf{s}^{(3)}(X) = 1 + X + X^2$	$(1\ 1\ 1)$
$\mathbf{e}(X) = X$	$\mathbf{s}^{(3)}(X) = X + X^2$	$(0\ 1\ 1)$
$\mathbf{e}(X) = X^0$	$\mathbf{s}^{(3)}(X) = 1 + X$	$(1\ 1\ 0)$

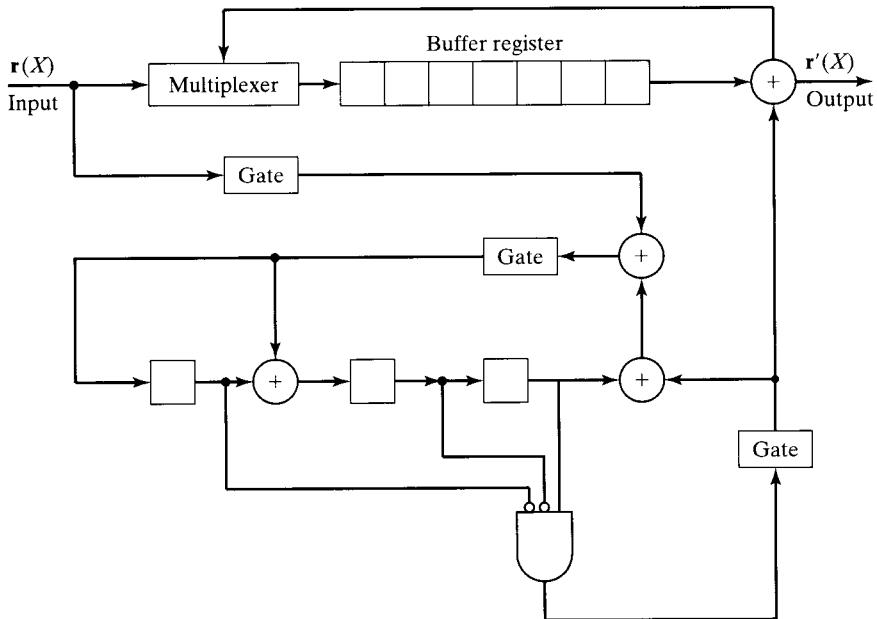


FIGURE 5.12: Decoding circuit for the $(7, 4)$ cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$.

(001) after the entire received polynomial $\mathbf{r}(X)$ has been shifted into the syndrome register; however, after another $6 - i$ shifts, the syndrome register will contain (001) . Based on this result, we obtain another decoding circuit for the $(7, 4)$ cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$, as shown in Figure 5.12. We see that the circuit shown in Figure 5.9 and the circuit shown in Figure 5.12 have the same complexity.

5.6 CYCLIC HAMMING CODES

The Hamming codes presented in Section 4.1 can be put in cyclic form. A cyclic Hamming code of length $2^m - 1$ with $m \geq 3$ is generated by a primitive polynomial $\mathbf{p}(X)$ of degree m .

In the following we show that the cyclic code defined previously is indeed a Hamming code, by examining its parity-check matrix in systematic form. The method presented in Section 5.2 is used to form the parity-check matrix. Dividing X^{m+i} by the generator polynomial $\mathbf{p}(X)$ for $0 \leq i < 2^m - m - 1$, we obtain

$$X^{m+i} = \mathbf{a}_i(X)\mathbf{p}(X) + \mathbf{b}_i(X), \quad (5.28)$$

where the remainder $\mathbf{b}_i(X)$ is of the form

$$\mathbf{b}_i(X) = b_{i0} + b_{i1}X + \cdots + b_{i,m-1}X^{m-1}.$$

Because X is not a factor of the primitive polynomial $\mathbf{p}(X)$, X^{m+i} and $\mathbf{p}(X)$ must be relatively prime. As a result, $\mathbf{b}_i(X) \neq 0$. Moreover, $\mathbf{b}_i(X)$ consists of at least two

terms. Suppose that $\mathbf{b}_i(X)$ has only one term, say X^j with $0 \leq j < m$. It follows from (5.28) that

$$X^{m+i} = \mathbf{a}_i(X)\mathbf{p}(X) + X^j.$$

Rearranging the preceding equation, we have

$$X^j(X^{m+i-j} + 1) = \mathbf{a}_i(X)\mathbf{p}(X).$$

Because X^j and $\mathbf{p}(X)$ are relatively prime, the foregoing equation implies that $\mathbf{p}(X)$ divides $X^{m+i-j} + 1$; however, this is impossible since $m + i - j < 2^m - 1$, and $\mathbf{p}(X)$ is a primitive polynomial of degree m . [Recall that the smallest positive integer n such that $\mathbf{p}(X)$ divides $X^n + 1$ is $2^m - 1$.] Therefore, for $0 \leq i < 2^m - m - 1$, the remainder $\mathbf{b}_i(X)$ contains at least two terms. Next we show that for $i \neq j$, $\mathbf{b}_i(X) \neq \mathbf{b}_j(X)$. From (5.28) we obtain

$$\mathbf{b}_i(X) + X^{m+i} = \mathbf{a}_i(X)\mathbf{p}(X),$$

$$\mathbf{b}_j(X) + X^{m+j} = \mathbf{a}_j(X)\mathbf{p}(X).$$

Suppose that $\mathbf{b}_i(X) = \mathbf{b}_j(X)$. Assuming that $i < j$ and combining the preceding two equations above we obtain the following relation:

$$X^{m+i}(X^{j-i} + 1) = [\mathbf{a}_i(X) + \mathbf{a}_j(X)]\mathbf{p}(X).$$

This equation implies that $\mathbf{p}(X)$ divides $X^{j-i} + 1$, but this is impossible, since $i \neq j$ and $j - i < 2^m - 1$. Therefore, $\mathbf{b}_i(X) \neq \mathbf{b}_j(X)$.

Let $\mathbf{H} = [\mathbf{I}_m \ \mathbf{Q}]$ be the parity-check matrix (in systematic form) of the cyclic code generated by $\mathbf{p}(X)$, where \mathbf{I}_m is an $m \times m$ identity matrix and \mathbf{Q} is an $m \times (2^m - m - 1)$ matrix. Let $\mathbf{b}_i = (b_{i0}, b_{i1}, \dots, b_{i,m-1})$ be the m -tuple corresponding to $\mathbf{b}_i(X)$. It follows from (5.17) that the matrix \mathbf{Q} has the $2^m - m - 1$ \mathbf{b}_i 's with $0 \leq i < 2^m - m - 1$ as all its columns. It follows from the preceding analysis that no two columns of \mathbf{Q} are alike, and each column has at least two 1's. Therefore, the matrix \mathbf{H} is indeed a parity-check matrix of a Hamming code, and $\mathbf{p}(X)$ generates this code.

The polynomial $\mathbf{p}(X) = 1 + X + X^3$ is a primitive polynomial. Therefore, the $(7, 4)$ cyclic code generated by $\mathbf{p}(X) = 1 + X + X^3$ is a Hamming code. A list of primitive polynomials with degree ≥ 3 is given in Table 2.7.

Cyclic Hamming codes can easily be decoded. To devise the decoding circuit, all we need to know is how to decode the first received digit. All the other received digits will be decoded in the same manner and with the same circuitry. Suppose that a single error has occurred at the highest-order position, X^{2^m-2} , of the received vector $\mathbf{r}(X)$ [i.e., the error polynomial is $\mathbf{e}(X) = X^{2^m-2}$]. Suppose that $\mathbf{r}(X)$ is shifted into the syndrome register from the right end. After the entire $\mathbf{r}(X)$ has entered the register, the syndrome in the register is equal to the remainder resulting from dividing $X^m \cdot X^{2^m-2}$ (the error polynomial preshifted m times) by the generator polynomial $\mathbf{p}(X)$. Because $\mathbf{p}(X)$ divides $X^{2^m-1} + 1$, the syndrome is of the following form:

$$\mathbf{s}(X) = X^{m-1}.$$

Therefore, if a single error occurs at the highest-order location of $\mathbf{r}(X)$, the resultant syndrome is $(0, 0, \dots, 0, 1)$. If a single error occurs at any other location of

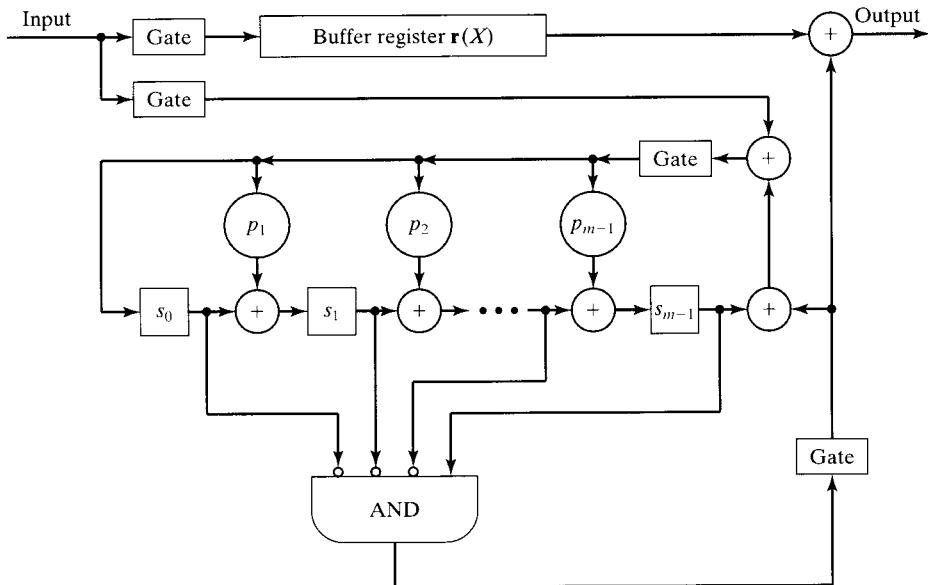


FIGURE 5.13: Decoder for a cyclic Hamming code.

$\mathbf{r}(X)$, the resultant syndrome will be different from $(0, 0, \dots, 0, 1)$. Based on this result only a single m -input AND gate is needed to detect the syndrome pattern $(0, 0, \dots, 0, 1)$. The inputs to this AND gate are $s_0', s_1', \dots, s_{m-2}'$ and s_{m-1}' , where s_i is a syndrome digit and s_i' denotes its complement. A complete decoding circuit for a cyclic Hamming code is shown in Figure 5.13. The decoding operation is described in the following steps:

- Step 1.** The syndrome is obtained by shifting the entire received vector into the syndrome register. At the same time, the received vector is stored in the buffer register. If the syndrome is zero, the decoder assumes that no error has occurred, and no correction is necessary. If the syndrome is not zero, the decoder assumes that a single error has occurred.
- Step 2.** The received word is read out of the buffer register digit by digit. As each digit is read out of the buffer register the syndrome register is shifted cyclically once. As soon as the syndrome in the register is $(0, 0, 0, \dots, 0, 1)$ the next digit to come out of the buffer is the erroneous digit, and the output of the m -input AND gate is 1.
- Step 3.** The erroneous digit is read out of the buffer register and is corrected by the output of the m -input AND gate. The correction is accomplished by an EXCLUSIVE-OR gate.
- Step 4.** The syndrome register is reset to zero after the entire received vector is read out of the buffer.

The cyclic Hamming code presented here can be modified to correct any single error and simultaneously to detect any combination of double errors. Let

$\mathbf{g}(X) = (X + 1)\mathbf{p}(X)$, where $\mathbf{p}(X)$ is a primitive polynomial of degree m . Because both $X + 1$ and $\mathbf{p}(X)$ divide $X^{2^m-1} + 1$ and since they are relatively prime, $\mathbf{g}(X)$ must also divide $X^{2^m-1} + 1$. A single-error-correcting and double-error-detecting cyclic Hamming code of length $2^m - 1$ is generated by $\mathbf{g}(X) = (X + 1)\mathbf{p}(X)$. The code has $m + 1$ parity-check digits. We show next that the minimum distance of this code is 4.

For convenience, we denote the single-error-correcting cyclic Hamming code by C_1 and denote the cyclic code generated by $\mathbf{g}(X) = (X + 1)\mathbf{p}(X)$ by C_2 . In fact, C_2 consists of the even-weight codewords of C_1 as all its codewords, because any odd-weight code polynomial in C_1 does not have $X + 1$ as a factor. Therefore, an odd-weight code polynomial of C_1 is not divisible by $\mathbf{g}(X) = (X + 1)\mathbf{p}(X)$, and it is not a code polynomial of C_2 ; however, an even-weight code polynomial of C_1 has $X + 1$ as a factor. Therefore, it is divisible by $\mathbf{g}(X) = (X + 1)\mathbf{p}(X)$, and it is also a code polynomial in C_2 . As a result, the minimum weight of C_2 is at least 4.

Next, we show that the minimum weight of C_2 is exactly 4. Let i, j , and k be three distinct nonnegative integers less than $2^m - 1$ such that $X^i + X^j + X^k$ is not divisible by $\mathbf{p}(X)$. Such integers do exist. For example, we first choose i and j . Dividing $X^i + X^j$ by $\mathbf{p}(X)$, we obtain

$$X^i + X^j = \mathbf{a}(X)\mathbf{p}(X) + \mathbf{b}(X),$$

where $\mathbf{b}(X)$ is the remainder with degree $m - 1$ or less. Because $X^i + X^j$ is not divisible by $\mathbf{p}(X)$, $\mathbf{b}(X) \neq 0$. Now, we choose an integer k such that when X^k is divided by $\mathbf{p}(X)$, the remainder is not equal to $\mathbf{b}(X)$. Therefore, $X^i + X^j + X^k$ is not divisible by $\mathbf{p}(X)$. Dividing this polynomial by $\mathbf{p}(X)$, we have

$$X^i + X^j + X^k = \mathbf{c}(X)\mathbf{p}(X) + \mathbf{d}(X). \quad (5.29)$$

Next, we choose a nonnegative integer l less than $2^m - 1$ such that when X^l is divided by $\mathbf{p}(X)$, the remainder is $\mathbf{d}(X)$; that is,

$$X^l = \mathbf{f}(X)\mathbf{p}(X) + \mathbf{d}(X). \quad (5.30)$$

The integer l cannot be equal to any of the three integers i, j , or k . Suppose that $l = i$. From (5.29) and (5.30) we would obtain

$$X^j + X^k = [\mathbf{c}(X) + \mathbf{f}(X)]\mathbf{p}(X).$$

This result implies that $\mathbf{p}(X)$ divides $X^{k-j} + 1$ (assuming that $j < k$), which is impossible, since $k - j < 2^m - 1$, and $\mathbf{p}(X)$ is a primitive polynomial. Therefore, $l \neq i$. Similarly, we can show that $l \neq j$ and $l \neq k$. Using this fact and combining (5.29) and (5.30), we obtain

$$X^i + X^j + X^k + X^l = [\mathbf{c}(X) + \mathbf{f}(X)]\mathbf{p}(X).$$

Because $X + 1$ is a factor of $X^i + X^j + X^k + X^l$ and it is not a factor of $\mathbf{p}(X)$, $\mathbf{c}(X) + \mathbf{f}(X)$ must be divisible by $X + 1$. As a result, $X^i + X^j + X^k + X^l$ is divisible by $\mathbf{g}(X) = (X + 1)\mathbf{p}(X)$. Therefore, it is a code polynomial in the code generated by $\mathbf{g}(X)$. It has weight 4. This proves that the cyclic code C_2 generated by $\mathbf{g}(X) = (X + 1)\mathbf{p}(X)$ has a minimum weight (or distance) of 4. Hence, it is capable of correcting any single error and simultaneously detecting any combination of double errors.

Because the distance-4 Hamming code C_2 of length $2^m - 1$ consists of the even-weight codewords of the distance-3 Hamming code C_1 of length $2^m - 1$ as its codewords, the weight enumerator $A_2(z)$ for C_2 can be determined from the weight enumerator $A_1(z)$ for C_1 . $A_2(z)$ consists of only the even power terms of $A_1(z)$. Therefore,

$$A_2(z) = \frac{1}{2}[A_1(z) + A_1(-z)] \quad (5.31)$$

(see Problem 5.8). Since $A_1(Z)$ is known and is given by (4.1), $A_2(z)$ can be determined from (4.1) and (5.31):

$$A_2(z) = \frac{1}{2(n+1)} \{(1+z)^n + (1-z)^n + 2n(1-z^2)^{(n-1)/2}\}, \quad (5.32)$$

where $n = 2^m - 1$. The dual of a distance-4 cyclic Hamming code is a $(2^m - 1, m + 1)$ cyclic code that has the following weight distribution:

$$B_0 = 1, \quad B_{2^{m-1}-1} = 2^m - 1, \quad B_{2^{m-1}} = 2^m - 1, \quad B_{2^m-1} = 1.$$

Therefore, the weight enumerator for the dual of a distance-4 cyclic Hamming code is

$$B_2(z) = 1 + (2^m - 1)z^{2^{m-1}-1} + (2^m - 1)z^{2^{m-1}} + z^{2^m-1}. \quad (5.33)$$

If a distance-4 cyclic Hamming code is used for error detection on a BSC, its probability of an undetected error, $P_u(E)$, can be computed from (3.33) and (5.32) or from (3.36) and (5.33). Computing $P_u(E)$ from (3.36) and (5.33), we obtain the following expression:

$$P_u(E) = 2^{-(m+1)} \{1 + 2(2^m - 1)(1-p)(1-2p)^{2^{m-1}-1} + (1-2p)^{2^m-1}\} - (1-p)^{2^m-1}. \quad (5.34)$$

Again, from (5.34), we can show that the probability of an undetected error for the distance-4 cyclic Hamming codes satisfies the upper bound $2^{-(n-k)} = 2^{-(m+1)}$ (see Problem 5.21).

Distance-3 and distance-4 cyclic Hamming codes are often used in communication systems for error detection.

5.7 ERROR-TRAPPING DECODING

In principle, the general decoding method of Meggitt's applies to any cyclic code, but refinements are necessary for practical implementation. If we put some restrictions on the error patterns that we intend to correct, the Meggitt decoder can be practically implemented. Consider an (n, k) cyclic code with generator polynomial $\mathbf{g}(X)$. Suppose that a code polynomial $\mathbf{v}(X)$ is transmitted and is corrupted by an error pattern $\mathbf{e}(X)$. Then, the received polynomial is $\mathbf{r}(X) = \mathbf{v}(X) + \mathbf{e}(X)$. We showed in Section 5.4 that the syndrome $\mathbf{s}(X)$ computed from $\mathbf{r}(X)$ is equal to the remainder resulting from dividing the error pattern $\mathbf{e}(X)$ by the generator $\mathbf{g}(X)$ [i.e., $\mathbf{e}(X) = \mathbf{a}(X)\mathbf{g}(X) + \mathbf{s}(X)$]. Suppose that errors are confined to the $n - k$ high-order positions, $X^k, X^{k+1}, \dots, X^{n-1}$ of $\mathbf{r}(X)$ [i.e., $\mathbf{e}(X) = e_k X^k + e_{k+1} X^{k+1} + \dots + e_{n-1} X^{n-1}$]. If $\mathbf{r}(X)$ is cyclically shifted $n - k$ times, the errors will be confined to $n - k$ low-order parity positions, $X^0, X^1, \dots, X^{n-k-1}$ of $\mathbf{r}^{(n-k)}(X)$. The corresponding error pattern is then

$$\mathbf{e}^{(n-k)}(X) = e_k + e_{k+1} X + \dots + e_{n-1} X^{n-k-1}.$$

Because the syndrome $\mathbf{s}^{(n-k)}(X)$ of $\mathbf{r}^{(n-k)}(X)$ is equal to the remainder resulting from dividing $\mathbf{e}^{(n-k)}(X)$ by $\mathbf{g}(X)$ and since the degree of $\mathbf{e}^{(n-k)}(X)$ is less than $n - k$, we obtain the following equality:

$$\mathbf{s}^{(n-k)}(X) = \mathbf{e}^{(n-k)}(X) = e_k + e_{k+1}X + \cdots + e_{n-1}X^{n-k-1}.$$

Multiplying $\mathbf{s}^{(n-k)}(X)$ by X^k , we have

$$\begin{aligned} X^k \mathbf{s}^{(n-k)}(X) &= \mathbf{e}(X) \\ &= e_k X^k + e_{k+1} X^{k+1} + \cdots + e_{n-1} X^{n-1}. \end{aligned}$$

This result says that if errors are confined to the $n - k$ high-order positions of the received polynomial $\mathbf{r}(X)$, the error pattern $\mathbf{e}(X)$ is identical to $X^k \mathbf{s}^{(n-k)}(X)$, where $\mathbf{s}^{(n-k)}(X)$ is the syndrome of $\mathbf{r}^{(n-k)}(X)$, the $(n - k)$ th cyclic shift of $\mathbf{r}(X)$. When this event occurs, we simply compute $\mathbf{s}^{(n-k)}(X)$ and add $X^k \mathbf{s}^{(n-k)}(X)$ to $\mathbf{r}(X)$. The resultant polynomial is the transmitted code polynomial.

Suppose that errors are not confined to the $n - k$ high-order positions but are confined to $n - k$ consecutive positions, say $X^i, X^{i+1}, \dots, X^{(n-k)+i-1}$ of $\mathbf{r}(X)$ (including the end-around case). If $\mathbf{r}(X)$ is cyclically shifted $n - i$ times to the right, errors will be confined to the $n - k$ low-order position of $\mathbf{r}^{(n-i)}(X)$, and the error pattern will be identical to $X^i \mathbf{s}^{(n-i)}(X)$, where $\mathbf{s}^{(n-i)}(X)$ is the syndrome of $\mathbf{r}^{(n-i)}(X)$.

Now, suppose that we shift the received polynomial $\mathbf{r}(X)$ into the syndrome register from the right end. Shifting $\mathbf{r}(X)$ into the syndrome register from the right end is equivalent to multiplying $\mathbf{r}(X)$ by X^{n-k} . After the entire $\mathbf{r}(X)$ has been shifted into the syndrome register, the contents of the syndrome register form the syndrome $\mathbf{s}^{(n-k)}(X)$ of $\mathbf{r}^{(n-k)}(X)$. If the errors are confined to $n - k$ high-order positions, $X^k, X^{k+1}, \dots, X^{n-1}$ of $\mathbf{r}(X)$, they are identical to $\mathbf{s}^{(n-k)}(X)$; however, if the errors are confined to $n - k$ consecutive positions (including end-around) other than the $n - k$ high-order positions of $\mathbf{r}(X)$, after the entire $\mathbf{r}(X)$ has been shifted into the syndrome register, the syndrome register must be shifted a certain number of times before its contents are identical to the error digits. This shifting of the syndrome register until its contents are identical to the error digits is called *error trapping* [14]. If errors are confined to $n - k$ consecutive positions of $\mathbf{r}(X)$ and if we can detect when the errors are trapped in the syndrome register, errors can be corrected simply by adding the contents of the syndrome register to the received digits at the $n - k$ proper positions.

Suppose that a t -error-correcting cyclic code is used. To detect the event that the errors are trapped in the syndrome register, we may simply test the *weight* of the syndrome after each shift of the syndrome register. As soon as the weight of the syndrome becomes t or less, we assume that errors are trapped in the syndrome register. If the number of errors in $\mathbf{r}(X)$ is t or less and if they are confined to $n - k$ consecutive positions, the errors are trapped in the syndrome register only when the weight of the syndrome in the register becomes t or less. This result can be shown as follows. An error pattern $\mathbf{e}(X)$ with t or fewer errors that are confined to $n - k$ consecutive positions must be of the form $\mathbf{e}(X) = X^j \mathbf{B}(X)$, where $\mathbf{B}(X)$ has t or fewer terms and has degree $n - k - 1$ or less. (For the end-around case, the same form would be obtained after a certain number of cyclic shifts of $\mathbf{e}(X)$.) Dividing $\mathbf{e}(X)$ by the generator polynomial $\mathbf{g}(X)$, we have

$$X^j \mathbf{B}(X) = \mathbf{a}(X) \mathbf{g}(X) + \mathbf{s}(X),$$

where $s(X)$ is the syndrome of $X^j \mathbf{B}(X)$. Because $s(X) + X^j \mathbf{B}(X)$ is a multiple of $\mathbf{g}(X)$, it is a code polynomial. The syndrome $s(X)$ cannot have weight t or less unless $s(X) = X^j \mathbf{B}(X)$. Suppose that the weight of $s(X)$ is t or less, and $s(X) \neq X^j \mathbf{B}(X)$. Then, $s(X) + X^j \mathbf{B}(X)$ is a nonzero code polynomial with weight less than $2t + 1$. This is impossible, since a t -error-correcting code must have a minimum weight of at least $2t + 1$. Therefore, we conclude that the errors are trapped in the syndrome register only when the weight of the syndrome becomes t or less.

Based on the error-trapping concept and the test just described, an error-trapping decoder can be implemented as shown in Figure 5.14. The decoding operation can be described in the following steps:

- Step 1.** The received polynomial $r(X)$ is shifted into the buffer and syndrome registers simultaneously with gates 1 and 3 turned on and all the other gates turned off. Because we are interested only in the recovery of the k information digits, the buffer register has to store only the k received information digits.
- Step 2.** As soon as the entire $r(X)$ has been shifted into the syndrome register, the weight of the syndrome in the register is tested by an $(n - k)$ -input threshold gate whose output is 1 when t or fewer of its inputs are 1; otherwise, it is zero.
 - a. If the weight of the syndrome is t or less, the syndrome digits in the syndrome register are identical to the error digits at the $n - k$ high-order positions $X^k, X^{k+1}, \dots, X^{n-1}$ of $r(X)$. Now, gates 2 and 4 are turned on and the other gates are turned off. The received vector is read out of the buffer register one digit at a time and is corrected by the error digits shifted out from the syndrome register.

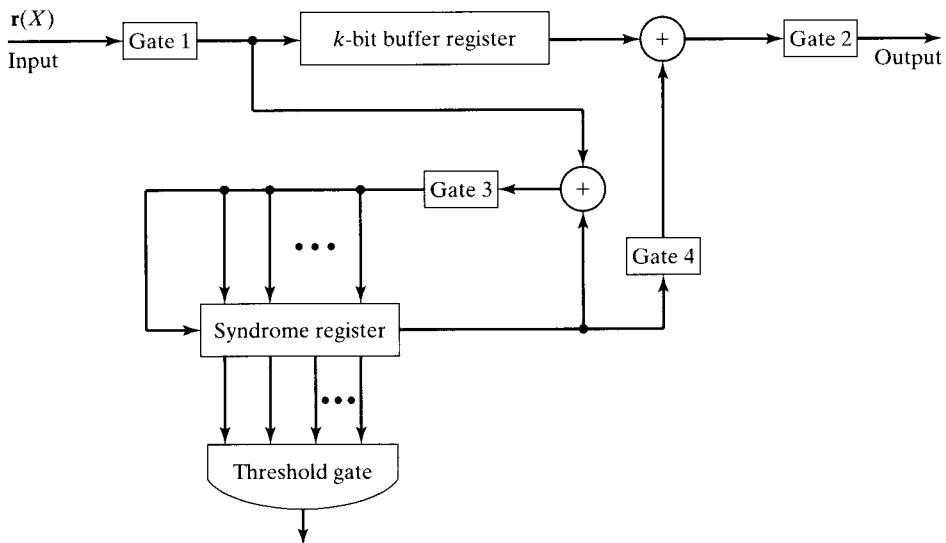


FIGURE 5.14: Error-trapping decoder.

- b.** If the weight of the syndrome is greater than t , the errors are *not* confined to the $n - k$ high-order positions of $\mathbf{r}(X)$, and they have not been trapped in the syndrome register. Go to step 3.
- Step 3.** The syndrome register is cyclically shifted once with gate 3 turned on and other gates turned off. The weight of the new syndrome is tested. (a) If it is t or less, the errors are confined to the locations $X^{k-1}, X^k, \dots, X^{n-2}$ of $\mathbf{r}(X)$, and the contents in the syndrome register are identical to the errors at these locations. Because the first received digit r_{n-1} is *error-free*, it is read out of the buffer register with gate 2 turned on. As soon as r_{n-1} has been read out, gate 4 is turned on and gate 3 is turned off. The contents in the syndrome register are shifted out and are used to correct the next $n - k$ received digits to come out from the buffer register. (b) If the weight of the syndrome is greater than t , the syndrome register is shifted once more with gate 3 turned on.
- Step 4.** The syndrome register is continuously shifted until the weight of its contents drops to t or less. If the weight goes down to t or less at the end of the i th shift, for $1 \leq i \leq k$, the first i received digits, $r_{n-i}, r_{n-i+1}, \dots, r_{n-1}$, in the buffer are *error-free*, and the contents in the syndrome register are identical to the errors at the locations $X^{k-i}, X^{k-i+1}, \dots, X^{n-i-1}$. As soon as the i error-free received digits have been read out of the buffer register, the contents in the syndrome register are shifted out and are used to correct the next $n - k$ received digit to come out from the buffer register. When the k received information digits have been read out of the buffer register and have been corrected, gate 2 is turned off. Any nonzero digits left in the syndrome register are errors in the parity part of $\mathbf{r}(X)$, and they will be ignored.
- Step 5.** If the weight of the syndrome never goes down to t or less by the time the syndrome register has been shifted k times, either an error pattern with errors confined to $n - k$ consecutive end-around locations has occurred or an uncorrectable error pattern has occurred. The syndrome register keeps being shifted. Suppose that the weight of its contents becomes t or less at the end of $k + l$ shifts with $1 \leq l \leq n - k$. Then, errors are confined to the $n - k$ consecutive end-around locations, $X^{n-l}, X^{n-l+1}, \dots, X^{n-1}, X^0, X^1, \dots, X^{n-k-l-1}$ of $\mathbf{r}(X)$. The l digits in the l leftmost stages of the syndrome register match the errors at the l high-order locations $X^{n-l}, X^{n-l+1}, \dots, X^{n-1}$ of $\mathbf{r}(X)$. Because the errors at the $n - k - l$ parity locations are not needed, the syndrome register is shifted $n - k - l$ times with all the gates turned off. Now, the l errors at the locations $X^{n-l}, X^{n-l+1}, \dots, X^{n-1}$ of $\mathbf{r}(X)$ are contained in the l rightmost stages of the syndrome register. With gates 2 and 4 turned on and other gates turned off, the received digits in the buffer register are read out and are corrected by the corresponding error digits shifted out from the syndrome register. This completes the decoding operation.

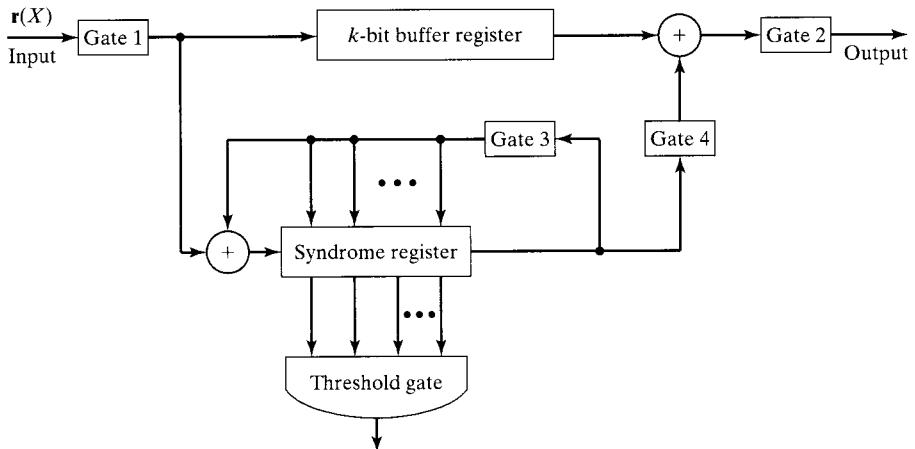


FIGURE 5.15: Another error-trapping decoder.

If the weight of the syndrome never drops to t or less by the time the syndrome register has been shifted a total of n times, either an uncorrectable error pattern has occurred or errors are not confined to $n - k$ consecutive positions. In either case, errors are detected. Except when errors are confined to the $n - k$ consecutive end-around positions of $\mathbf{r}(X)$, the received information digits can be read out of the buffer register, corrected, and delivered to the data sink after at most k cyclic shifts of the syndrome register. When an error pattern with errors confined to $n - k$ consecutive end-around locations of $\mathbf{r}(X)$ occurs, a total of n cyclic shifts of the syndrome register is required before the received message can be read out of the buffer register for corrections. For large n and $n - k$, the number of correctable end-around error patterns becomes big, which causes an undesirably long decoding delay.

It is possible to implement the error-trapping decoding in a different manner so that the error patterns with errors confined to $n - k$ consecutive end-around locations can be corrected as fast as possible. This can be done by shifting the received vector $\mathbf{r}(X)$ into the syndrome register from the *left* end, as shown in Figure 5.15. This variation is based on the following facts. If the errors are confined to $n - k$ low-order parity positions $X^0, X^1, \dots, X^{n-k-1}$ of $\mathbf{r}(X)$, then after the entire $\mathbf{r}(X)$ has entered the syndrome register, the contents in the register are identical to the error digits at the locations $X^0, X^1, \dots, X^{n-k-1}$ of $\mathbf{r}(X)$. Suppose that the errors are not confined to the $n - k$ low-order positions of $\mathbf{r}(X)$ but are confined to $n - k$ consecutive locations (including the end-around case), say $X^i, X^{i+1}, \dots, X^{(n-k)+i-1}$. After $n - i$ cyclic shifts of $\mathbf{r}(X)$, the errors will be shifted to the $n - k$ low-order positions of $\mathbf{r}^{(n-i)}(X)$, and the syndrome of $\mathbf{r}^{(n-i)}(X)$ will be identical to the errors confined to positions $X^i, X^{i+1}, \dots, X^{(n-k)+i-1}$ of $\mathbf{r}(X)$. The operation of the decoder shown in Figure 5.15 is as follows:

Step 1. Gates 1 and 3 are turned on and the other gates are turned off. The received vector $\mathbf{r}(X)$ is shifted into the syndrome register and simultaneously into the buffer register (only the k received information digits

are stored in the buffer register). As soon as the entire $\mathbf{r}(X)$ has been shifted into the syndrome register, the contents of the register form the syndrome $\mathbf{s}(X)$ of $\mathbf{r}(X)$.

Step 2. The weight of the syndrome is tested. (a) If the weight is t or less, the errors are confined to the $(n - k)$ low-order parity positions $X^0, X^1, \dots, X^{n-k-1}$ of $\mathbf{r}(X)$. Thus, the k received information digits in the buffer register are *error-free*. Gate 2 is then turned on, and the error-free information digits are read out of the buffer with gate 4 turned off. (b) If the weight of the syndrome is greater than t , the syndrome register is then shifted once with gate 3 turned on and the other gates turned off. Go to step 3.

Step 3. The weight of the new contents in the syndrome register is tested. (a) If the weight is t or less, the errors are confined to the position $X^{n-1}, X^0, X^1, \dots, X^{n-k-2}$ of $\mathbf{r}(X)$ (end-around case). The leftmost digit in the syndrome register is identical to the error at the position X^{n-1} of $\mathbf{r}(X)$; the other $n - k - 1$ digits in the syndrome register match the errors at parity positions $X^0, X^1, \dots, X^{n-k-2}$ of $\mathbf{r}(X)$. The output of the threshold gate turns gate 3 off and sets a clock to count from 2. The syndrome register is then shifted (in step with the clock) with gate 3 turned off. As soon as the clock has counted to $n - k$, the contents of the syndrome register will be $(0\ 0 \dots\ 0\ 1)$. The rightmost digit matches the error at position X^{n-1} of $\mathbf{r}(X)$. The k received information digits are then read out of the buffer, and the first received information digit is corrected by the 1 coming out from the syndrome register. The decoding is thus completed. (b) If the weight of the contents in the syndrome register is greater than t , the syndrome register is shifted once again with gate 3 turned on and other gates turned off. Go to step 4.

Step 4. Step 3(b) repeats until the weight of the contents of the syndrome register drop to t or less. If the weight drops to t or less after the i th shift, for $1 \leq i \leq n - k$, the clock starts to count from $i + 1$. At the same time the syndrome register is shifted with gate 3 turned off. As soon as the clock has counted to $n - k$, the rightmost i digits in the syndrome register match the errors in the first i received information digits in the buffer register. The other information digits are error-free. Gates 2 and 4 are then turned on. The received information digits are read out of the buffer for correction.

Step 5. If the weight of the contents of the syndrome register never drops to t or less by the time that the syndrome register has been shifted $n - k$ times (with gate 3 turned on), gate 2 is then turned on, and the received information digits are read out of the buffer one at a time. At the same time the syndrome register is shifted with gate 3 turned on. As soon as the weight of the contents of the syndrome register drops to t or less, the contents match the errors in the next $n - k$ digits to come out of the buffer. Gate 4 is then turned on, and the erroneous information digits are corrected by the digits coming out from the syndrome register with gate 3 turned off. Gate 2 is turned off as soon as k information digits have been read out of the buffer.

With the implementation of the error-trapping decoding just described, the received information digits can be read out of the buffer register after at most $n - k$ shifts of the syndrome register. For large $n - k$, this implementation provides faster decoding than the decoder shown in Figure 5.14; however, when $n - k$ is much smaller than k , the first implementation of error trapping is more advantageous in decoding speed than the one shown in Figure 5.15.

The decoding of cyclic Hamming codes presented in Section 5.6 is actually an error-trapping decoding. The syndrome register is cyclically shifted until the single error is trapped in the rightmost stage of the register. Error-trapping decoding is most effective for decoding single-error-correcting block codes and burst-error-correcting codes (decoding of burst-error-correcting codes is discussed in Chapter 20). It is also effective for decoding some short double-error-correcting codes. When error-trapping decoding is applied to long and high-rate codes (small $n - k$) with large error-correcting capability, it becomes very ineffective, and much of the error-correcting capability is sacrificed. Several refinements of this simple decoding technique [12]–[19] have been devised to extend its application to multiple-error-correcting codes. One of the refinements is presented in the next section.

EXAMPLE 5.11

Consider the $(15, 7)$ cyclic code generated by $\mathbf{g}(X) = 1 + X^4 + X^6 + X^7 + X^8$. This code has a minimum distance of $d_{min} = 5$, which will be proved in Chapter 6. Hence, the code is capable of correcting any combination of two or fewer errors over a block of 15 digits. Suppose that we decode this code with an error-trapping decoder.

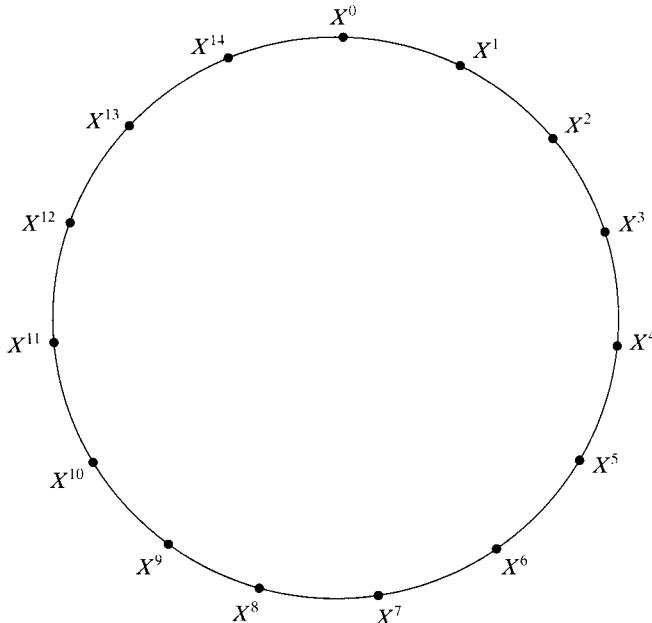


FIGURE 5.16: Ring arrangement of code digit positions.

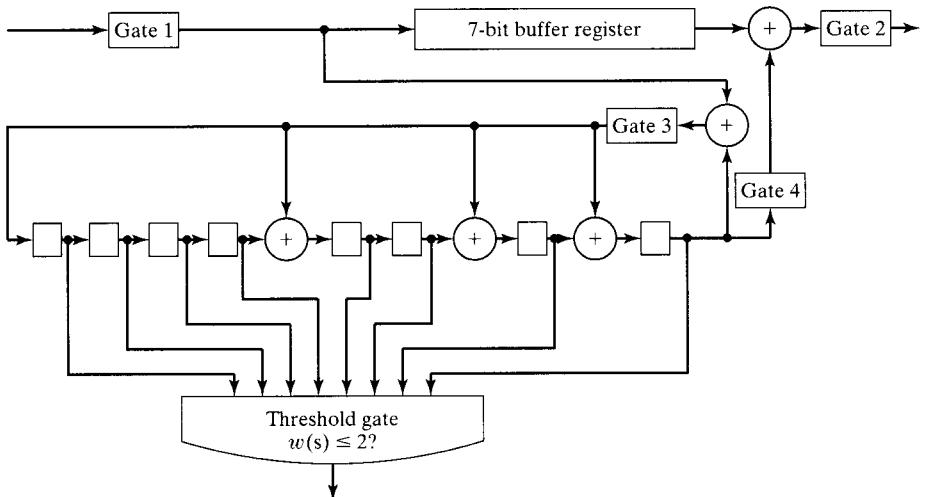


FIGURE 5.17: Error-trapping decoder for the $(15, 7)$ cyclic code generated by $\mathbf{g}(X) = 1 + X^4 + X^6 + X^7 + X^8$.

Clearly, any single error is confined to $n - k = 8$ consecutive positions. Therefore, any single error can be trapped and corrected. Now, consider any double errors over a span of 15 digits. If we arrange the 15 digit positions X^0 to X^{14} as a ring, as shown in Figure 5.16, we see that any double errors are confined to eight consecutive positions. Hence, any double errors can be trapped and corrected. An error-trapping decoder for the $(15, 7)$ cyclic code generated by $\mathbf{g}(X) = 1 + X^4 + X^6 + X^7 + X^8$ is shown in Figure 5.17.

5.8 IMPROVED ERROR-TRAPPING DECODING

The error-trapping decoding discussed in Section 5.7 can be improved to correct error patterns such that, for each error pattern, most errors are confined to $n - k$ consecutive positions, and fewer errors are outside the $(n - k)$ -digit span. This improvement needs additional circuitry. The complexity of the additional circuitry depends on how many errors outside an $(n - k)$ -digit span are to be corrected. An improvement proposed by Kasami [17] is discussed here.

The error pattern $\mathbf{e}(X) = e_0 + e_1 X + e_2 X^2 + \dots + e_{n-1} X^{n-1}$ that corrupted the transmitted codeword can be divided into two parts:

$$\mathbf{e}_p(X) = e_0 + e_1 X + \dots + e_{n-k-1} X^{n-k-1}$$

$$\mathbf{e}_I(X) = e_{n-k} X^{n-k} + \dots + e_{n-1} X^{n-1},$$

where $\mathbf{e}_I(X)$ contains the errors in the message section of the received vector, and $\mathbf{e}_p(X)$ contains the errors in the parity section of the received vector. Dividing $\mathbf{e}_I(X)$ by the code generator polynomial $\mathbf{g}(X)$, we obtain

$$\mathbf{e}_I(X) = \mathbf{q}(X)\mathbf{g}(X) + \boldsymbol{\rho}(X), \quad (5.35)$$

where $\rho(X)$ is the remainder with degree $n - k - 1$ or less. Adding $\mathbf{e}_p(X)$ to both sides of (5.35), we obtain

$$\mathbf{e}(X) = \mathbf{e}_p(X) + \mathbf{e}_I(X) = \mathbf{q}(X)\mathbf{g}(X) + \rho(X) + \mathbf{e}_p(X). \quad (5.36)$$

Because $\mathbf{e}_p(X)$ has degree $n - k - 1$ or less, $\rho(X) + \mathbf{e}_p(X)$ must be the remainder resulting from dividing the error pattern $\mathbf{e}(X)$ by the generator polynomial. Thus, $\rho(X) + \mathbf{e}_p(X)$ is equal to the syndrome of the received vector $\mathbf{r}(X)$,

$$\mathbf{s}(X) = \rho(X) + \mathbf{e}_p(X). \quad (5.37)$$

Rearranging (5.37), we have

$$\mathbf{e}_p(X) = \mathbf{s}(X) + \rho(X). \quad (5.38)$$

That is, if the error pattern $\mathbf{e}_I(X)$ in the message positions is known, the error pattern $\mathbf{e}_p(X)$ in the parity positions can be found.

Kasami's error-trapping decoding requires finding a set of polynomials $[\phi_j(X)]_{j=1}^N$ of degree $k - 1$ or less such that, for any correctable error pattern $\mathbf{e}(X)$, there is one polynomial $\phi_j(X)$ such that $X^{n-k}\phi_j(X)$ matches the message section of $\mathbf{e}(X)$ or the message section of a cyclic shift of $\mathbf{e}(X)$. The polynomials $\phi_j(X)$'s are called the *covering polynomials*. Let $\rho_j(X)$ be the remainder resulting from dividing $X^{n-k}\phi_j(X)$ by the generator polynomial $\mathbf{g}(X)$ of the code.

The decoding procedure is as follows:

- Step 1.** Calculate the syndrome $\mathbf{s}(X)$ by entering the entire received vector into the syndrome register.
- Step 2.** Calculate the weight of the sum $\mathbf{s}(X) + \rho_j(X)$ for each $j = 1, 2, \dots, N$ (i.e., $w[\mathbf{s}(X) + \rho_j(X)]$ for $j = 1, 2, \dots, N$).
- Step 3.** If, for some l ,

$$w[\mathbf{s}(X) + \rho_l(X)] \leq t - w[\phi_l(X)],$$

then $X^{n-k}\phi_l(X)$ matches the error pattern in the message section of $\mathbf{e}(X)$, and $\mathbf{s}(X) + \rho_l(X)$ matches the error pattern in the parity section of $\mathbf{e}(X)$. Thus,

$$\mathbf{e}(X) = \mathbf{s}(X) + \rho_l(X) + X^{n-k}\phi_l(X).$$

The correction is then performed by taking the modulo-2 sum $\mathbf{r}(X) + \mathbf{e}(X)$. This step requires $N(n - k)$ -input threshold gates to test the weights of $\mathbf{s}(X) + \rho_j(X)$ for $j = 1, 2, \dots, N$.

- Step 4.** If $w[\mathbf{s}(X) + \rho_j(X)] > t - w[\phi_j(X)]$ for all $j = 1, 2, \dots, N$, both syndrome and buffer registers are shifted cyclically once. Then, the new contents of the syndrome register, $\mathbf{s}^{(1)}(X)$, is the syndrome corresponding to $\mathbf{e}^{(1)}(X)$, which is obtained by shifting the error pattern $\mathbf{e}(X)$ cyclically one place to the right.
- Step 5.** The weight of $\mathbf{s}^{(1)}(X) + \rho_j(X)$ is computed for $j = 1, 2, \dots, N$. If, for some l ,

$$w[\mathbf{s}^{(1)}(X) + \rho_l(X)] \leq t - w[\phi_l(X)],$$

then $X^{n-k}\phi_l(X)$ matches the errors in the message section of $\mathbf{e}^{(1)}(X)$, and $\mathbf{s}^{(1)}(X) + \rho_l(X)$ matches the errors in the parity section of $\mathbf{e}^{(1)}(X)$. Thus,

$$\mathbf{e}^{(1)}(X) = \mathbf{s}^{(1)}(X) + \rho_l(X) + X^{n-k}\phi_l(X).$$

The correction is then performed by taking the modulo-2 sum $\mathbf{r}^{(1)}(X) + \mathbf{e}^{(1)}(X)$. If

$$w[\mathbf{s}^{(1)}(X) + \rho_j(X)] > t - w[\phi_j(X)]$$

for all $j = 1, 2, \dots, N$, both syndrome and buffer registers are shifted cyclically once again.

- Step 6.** The syndrome and buffer registers are continuously shifted until $\mathbf{s}^{(i)}(X)$ (the syndrome after the i th shift) is found such that, for some l ,

$$w[\mathbf{s}^{(i)}(X) + \rho_l(X)] \leq t - w[\phi_l(X)].$$

Then,

$$\mathbf{e}^{(i)}(X) = \mathbf{s}^{(i)}(X) + \rho_l(X) + X^{n-k}\phi_l(X),$$

where $\mathbf{e}^{(i)}(X)$ is the i th cyclic shift of $\mathbf{e}(X)$. If the weight $w[\mathbf{s}^{(i)}(X) + \rho_j(X)]$ never drops to $t - w[\phi_j(X)]$ or less for all j by the time that the syndrome and buffer registers have been cyclically shifted $n - 1$ times, an uncorrectable error pattern is detected.

The complexity of a decoder that employs the decoding method just described depends on N , the number of covering polynomials in $\{\phi_j(X)\}_{j=1}^N$. The combinational logical circuitry consists of $N(n-k)$ -input threshold gates. To find the set of covering polynomials $\{\phi_j(X)\}_{j=1}^N$ for a specific code is not an easy problem. Several methods for finding this set can be found in [17], [20], and [21].

This improved error-trapping method is applicable to many double- and triple-error-correcting codes, however, it is still applicable only to relatively short and low-rate codes. When the code length n and error-correcting capability t become large, the number of threshold gates required in the error-detecting logical circuitry becomes very large and impractical.

Other variations of error-trapping decoding can be found in [15], [16], and [18].

5.9 THE (23, 12) GOLAY CODE

As pointed out in Section 4.6, the (23, 12) Golay code [22] with a minimum distance of 7 is the only known multiple-error-correcting binary perfect code. This code can be put in cyclic form, and hence it can be encoded and decoded based on its cyclic structure. It is generated either by

$$\mathbf{g}_1(X) = 1 + X^2 + X^4 + X^5 + X^6 + X^{10} + X^{11}$$

or by

$$\mathbf{g}_2(X) = 1 + X + X^5 + X^6 + X^7 + X^9 + X^{11}.$$

Both $\mathbf{g}_1(X)$ and $\mathbf{g}_2(X)$ are factors of $X^{23} + 1$ and $X^{23} + 1 = (1 + X)\mathbf{g}_1(X)\mathbf{g}_2(X)$. The encoding can be accomplished by an 11-stage shift register with feedback

connections according to either $\mathbf{g}_1(X)$ or $\mathbf{g}_2(X)$. If the simple error-trapping scheme described in Section 5.7 is used for decoding this code, some of the double-error patterns and many of the triple-error patterns cannot be trapped. For example, consider the double-error patterns $\mathbf{e}(X) = X^{11} + X^{22}$. The two errors are never confined to $n - k = 11$ consecutive positions, no matter how many times $\mathbf{e}(X)$ is cyclically shifted. Therefore, they can never be trapped in the syndrome register and cannot be corrected. We can also readily see that the triple-error pattern $\mathbf{e}(X) = X^5 + X^{11} + X^{22}$ cannot be trapped. Therefore, if the simple error-trapping scheme for decoding the Golay code is used, some of its error-correcting capability will be lost; however, the decoding circuitry is simple.

There are several practical ways to decode the (23, 12) Golay code up to its error-correcting capability $t = 3$. Two of the best are discussed in this section. Both are refined error-trapping schemes.

5.9.1 Kasami Decoder [17]

The Golay code can easily be decoded by Kasami's error-trapping technique. The set of polynomials $\{\phi_j(X)\}_{j=1}^N$ is chosen as follows:

$$\phi_1(X) = 0, \quad \phi_2(X) = X^5, \quad \phi_3(X) = X^6.$$

Let $\mathbf{g}_1(X) = 1 + X^2 + X^4 + X^5 + X^6 + X^{10} + X^{11}$ be the generator polynomial. Dividing $X^{11}\phi_j(X)$ by $\mathbf{g}_1(X)$ for $j = 1, 2, 3$, we obtain the following remainders:

$$\rho_1(X) = 0,$$

$$\rho_2(X) = X + X^2 + X^5 + X^6 + X^8 + X^9,$$

$$\rho_3(X) = X\rho_2(X) = X^2 + X^3 + X^6 + X^7 + X^9 + X^{10}.$$

A decoder based on Kasami's error-trapping scheme is shown in Figure 5.18. The received vector is shifted into the syndrome register from the rightmost stage; this is equivalent to preshifting the received vector 11 times cyclically. After the entire received vector has entered the syndrome register, the syndrome in the register corresponds to $\mathbf{r}^{(11)}(X)$, which is the eleventh cyclic shift of $\mathbf{r}(X)$. In this case, if the errors are confined to the first 11 high-order positions $X^{12}, X^{13}, \dots, X^{22}$ of $\mathbf{r}(X)$, the syndrome matches the errors in those positions. The error-correction procedure of this decoder is as follows:

- Step 1.** Gates 1, 3, and 5 are turned on; gates 2 and 4 are turned off. The received vector $\mathbf{r}(X)$ is read into the syndrome register and simultaneously into the buffer register. The syndrome $\mathbf{s}(X) = s_0 + s_1X + \dots + s_{10}X^{10}$ is formed and is read into three threshold gates.
- Step 2.** Gates 1, 4, and 5 are turned off; gates 2 and 3 are turned on. The syndrome is tested for correctable error patterns as follows:
 - a. If the weight $w[\mathbf{s}(X)] \leq 3$, all the errors are confined to the 11 high-order positions of $\mathbf{r}(X)$, and $\mathbf{s}(X)$ matches the errors. Thus, the erroneous symbols are the next 11 digits to come out of the buffer register. The output of the threshold gate T_0 turns gate 4 on and

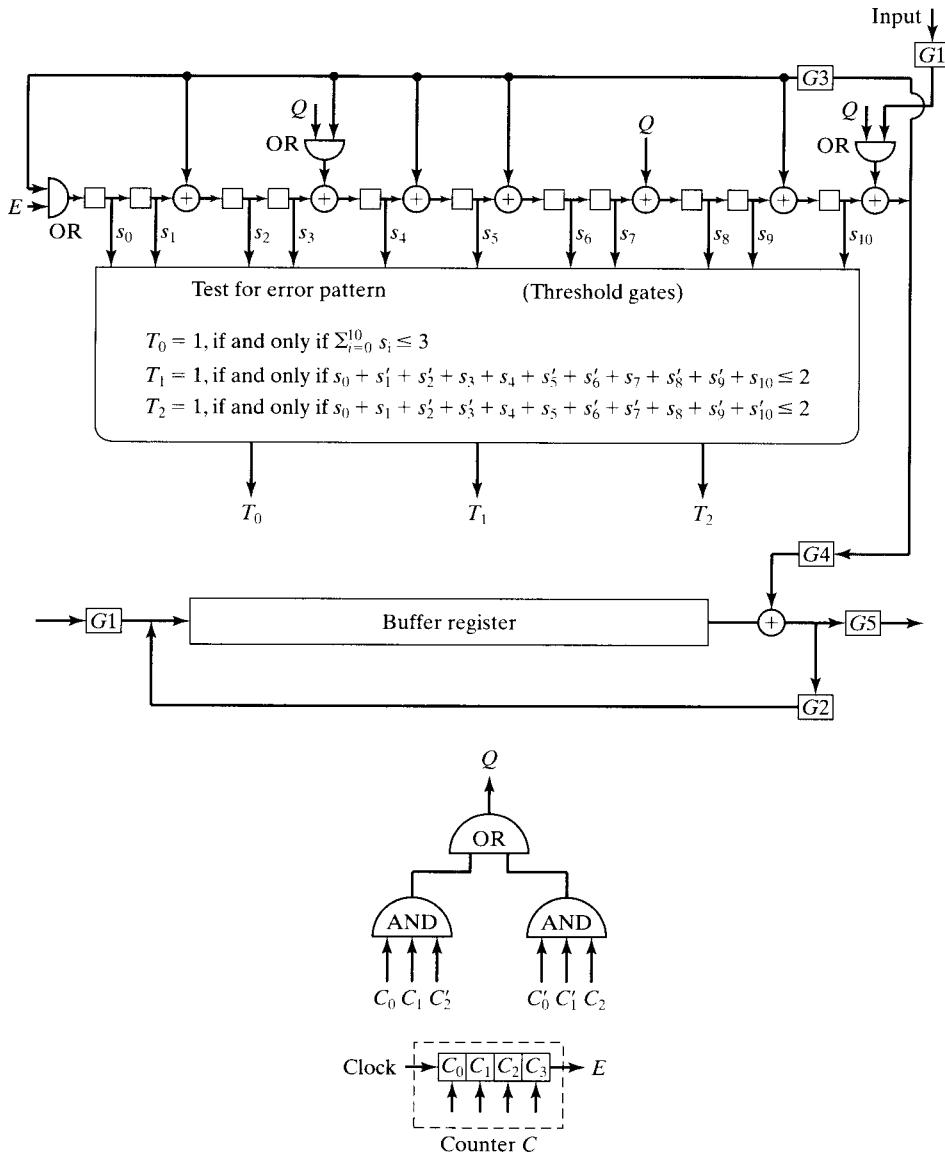


FIGURE 5.18: An error-trapping decoder for the (23, 12) Golay code.

gate 3 off. Digits are read out one at a time from the buffer register. The digit coming out of the syndrome register is added (modulo-2) to the digit coming out of the buffer. This corrects the errors.

- b. If $w[s(X)] > 3$, the weight of $s(X) + \rho_2(X)$ is tested. If $w[s(X) + \rho_2(X)] \leq 2$, then $s(X) + \rho_2(X) = s_0 + s'_1 X + s'_2 X^2 + s_3 X^3 + s_4 X^4 + s'_5 X^5 + s'_6 X^6 + s_7 X^7 + s'_8 X^8 + s'_9 X^9 + s_{10} X^{10}$ is identical to the error

pattern in the 11 high-order positions of the received word, and a single error occurs at location X^5 , where s'_i is the complement of s_i . Gate 4 is turned on, and gate 3 is turned off. The counter C starts to count from 2. At the same time, the syndrome register is shifted without feedback. The output \mathbf{Q} , which is 1 when and only when C counts 3 and 4, is fed into the syndrome register to form the error pattern $\mathbf{s}(X) + \rho_2(X)$. When the counter C counts 8, its output E is 1, and the leftmost stage of the syndrome register is set to 1. This 1 is used for correcting the error at location X^5 in the received vector $\mathbf{r}(X)$. The digits coming out of the buffer are then corrected by the digits coming out of the syndrome register.

- c. If $w[\mathbf{s}(X)] > 3$ and $w[\mathbf{s}(X) + \rho_2(X)] > 2$, the weight of $\mathbf{s}(X) + \rho_3(X)$ is tested. If $w[\mathbf{s}(X) + \rho_3(X)] \leq 2$, then $\mathbf{s}(X) + \rho_3(X) = s_0 + s_1X + s'_2X^2 + s'_3X^3 + s_4X^4 + s_5X^5 + s'_6X^6 + s'_7X^7 + s_8X^8 + s'_9X^9 + s'_{10}X^{10}$ is identical to the error pattern in the 11 high-order positions of the received word and a single error occurs at positions X^6 . The correction is the same as step (b), except that counter C starts to count from 3. If $w[\mathbf{s}(X)] > 3$, $w[\mathbf{s}(X) + \rho_2(X)] > 2$, and $w[\mathbf{s}(X) + \rho_3(X)] > 2$, then the decoder moves to step 3.

Step 3. Both the syndrome and buffer registers are cyclically shifted once with gates 1, 4, and 5 turned off and gates 2 and 3 turned on. The new contents of the syndrome register are $\mathbf{s}^{(1)}(X)$. Step 2 is then repeated.

Step 4. The decoding operation is completed as soon as the buffer register has been cyclically shifted 46 times. Gate 5 is then turned on and the vector in the buffer is shifted out to the data sink.

If there are three or fewer errors in the received vector, the vector in the buffer at the end of decoding will be the transmitted codeword. If there are more than three errors in the received vector, the vector in the buffer at the end of decoding will not be the transmitted codeword.

5.9.2 Systematic Search Decoder [23]

This decoding method is based on the fact that every pattern of three or fewer errors in a block of 23 digits can be cyclically shifted so that at most one of the errors lies outside a specified 11-digit section of the word. The decoding procedure is as follows:

- Step 1.** Compute the syndrome from the received vector.
- Step 2.** Shift the syndrome and the received vector 23 times, checking whether the weight of the syndrome ever falls to 3 or less. If it does, the syndrome with weight 3 or less matches the error pattern and correction can be made.
- Step 3.** If it does not, the first received information digit is inverted and step 2 is repeated, checking for a syndrome of weight of 2 or less. If one is found, the first received information digit was incorrect and the other two errors are specified by the syndrome. This completes the decoding.

- Step 4.** If no syndrome of weight 2 or less is found in step 3, the first information digit was originally correct. In this case, this bit must be reinverted.
- Step 5.** Repeat step 3 by inverting the second, third, . . . , and twelfth information digits. Because not all the errors are in the parity-check section, an error must be corrected in this manner.

In every pattern of three or fewer errors, there is at least one error that if corrected, will leave the remaining error or errors within 11 successive positions. When the digit corresponding to this error is inverted, the remaining errors are corrected as in ordinary error trapping.

Compared with the Kasami decoder, the systematic search decoder has the advantage that only one weight-sensing (threshold) gate is required; however, it has the disadvantage that the clock and timing circuitry is more complex than that of the Kasami decoder, since 12 different digits must be inverted sequentially. Also, the Kasami decoder operates faster than the systematic search decoder.

This systematic search technique can be generalized for decoding other multiple-error-correcting cyclic codes.

The weight enumerator for the (23, 12) Golay code is

$$A(Z) = 1 + 253z^7 + 506z^8 + 1288z^{11} + 1288z^{12} + 506z^{15} + 253z^{16} + z^{23}.$$

If this code is used for error detection on a BSC, its probability of an undetected error $P_u(E)$ can be computed from (3.19). Moreover, $P_u(E)$ satisfies the upper bound 2^{-11} [i.e., $P_u(E) \leq 2^{-11}$] [24]. Therefore, the (23, 12) Golay code is a good error-detecting code.

5.10 SHORTENED CYCLIC CODES

In system design, if a code of suitable natural length or suitable number of information digits cannot be found, it may be desirable to shorten a code to meet the requirements. A technique for shortening a cyclic code is presented in this section. This technique leads to simple implementation of the encoding and decoding for the shortened code.

Given an (n, k) cyclic code C consider the set of codewords for which the l leading high-order information digits are identical to zero. There are 2^{k-l} such codewords, and they form a linear subcode of C . If the l zero information digits are deleted from each of these codewords, we obtain a set of 2^{k-l} vectors of length $n - l$. These 2^{k-l} shortened vectors form an $(n - l, k - l)$ linear code. This code is called a *shortened cyclic code* (or *polynomial code*), and it is not cyclic. A shortened cyclic code has at least the same error-correcting capability as the code from which it is derived.

The encoding and decoding for a shortened cyclic code can be accomplished by the same circuits as those employed by the original cyclic code. This is so because the deleted l leading-zero information digits do not affect the parity-check and syndrome computations; however, in decoding the shortened cyclic code after the entire received vector has been shifted into the syndrome register, the syndrome register must be cyclically shifted l times to generate the proper syndrome for decoding the first received digit r_{n-l-1} . For large l , these extra l shifts of the syndrome register

cause undesirable decoding delay; they can be eliminated by modifying either the connections of the syndrome register or the error-pattern detection circuit.

Let $\mathbf{r}(X) = r_0 + r_1 X + \cdots + r_{n-l-1} X^{n-l-1}$ be the received polynomial. Suppose that $\mathbf{r}(X)$ is shifted into the syndrome register from the right end. If the decoding circuit for the original cyclic code is used for decoding the shortened code, the proper syndrome for decoding the received digit r_{n-l-1} is equal to the remainder resulting from dividing $X^{n-k+l}\mathbf{r}(X)$ by the generator polynomial $\mathbf{g}(X)$. Because shifting $\mathbf{r}(X)$ into the syndrome register from the right end is equivalent to premultiplying $\mathbf{r}(X)$ by X^{n-k} , the syndrome register must be cyclically shifted another l times after the entire $\mathbf{r}(X)$ has been shifted into the register. Now, we want to show how these extra l shifts can be eliminated by modifying the connections of the syndrome register. Dividing $X^{n-k+l}\mathbf{r}(X)$ by $\mathbf{g}(X)$, we obtain

$$X^{n-k+l}\mathbf{r}(X) = \mathbf{a}_1(X)\mathbf{g}(X) + \mathbf{s}^{(n-k+l)}(X), \quad (5.39)$$

where $\mathbf{s}^{(n-k+l)}(X)$ is the remainder and the syndrome for decoding the received digit r_{n-l-1} . Next, we divide X^{n-k+l} by $\mathbf{g}(X)$. Let $\rho(X) = \rho_0 + \rho_1 X + \cdots + \rho_{n-k-1} X^{n-k-1}$ be the remainder resulting from this division. Then, we have the following relation:

$$\rho(X) = X^{n-k+l} + \mathbf{a}_2(X)\mathbf{g}(X). \quad (5.40)$$

Multiplying both sides of (5.40) by $\mathbf{r}(X)$ and using the equality of (5.39), we obtain the following relation between $\rho(X)\mathbf{r}(X)$ and $\mathbf{s}^{(n-k+l)}(X)$:

$$\rho(X)\mathbf{r}(X) = [\mathbf{a}_1(X) + \mathbf{a}_2(X)\mathbf{r}(X)]\mathbf{g}(X) + \mathbf{s}^{(n-k+l)}(X). \quad (5.41)$$

The foregoing equality suggests that we can obtain the syndrome $\mathbf{s}^{(n-k+l)}(X)$ by multiplying $\mathbf{r}(X)$ by $\rho(X)$ and dividing the product $\rho(X)\mathbf{r}(X)$ by $\mathbf{g}(X)$. Computing $\mathbf{s}^{(n-k+l)}(X)$ in this way, we can avoid the extra l shifts of the syndrome register. Simultaneously multiplying $\mathbf{r}(X)$ by $\rho(X)$ and dividing $\rho(X)\mathbf{r}(X)$ by $\mathbf{g}(X)$ can be accomplished by the circuit shown in Figure 5.19. As soon as the received polynomial $\mathbf{r}(X)$ has been shifted into the register, the contents in the register form the syndrome $\mathbf{s}^{(n-k+l)}(X)$, and the first received digit is ready to be decoded.

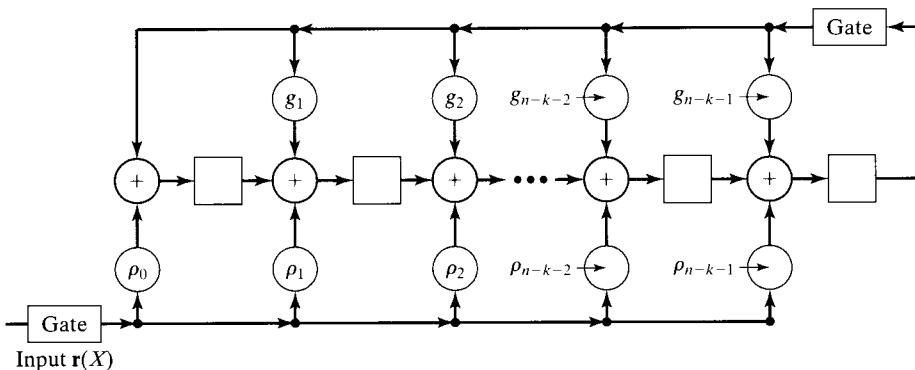


FIGURE 5.19: Circuit for multiplying $\mathbf{r}(X)$ by $\rho(X) = \rho_0 + \rho_1 X + \cdots + \rho_{n-k-1} X^{n-k-1}$ and dividing $\rho(X)\mathbf{r}(X)$ by $\mathbf{g}(X) = 1 + g_1 X + \cdots + X^{n-k}$.

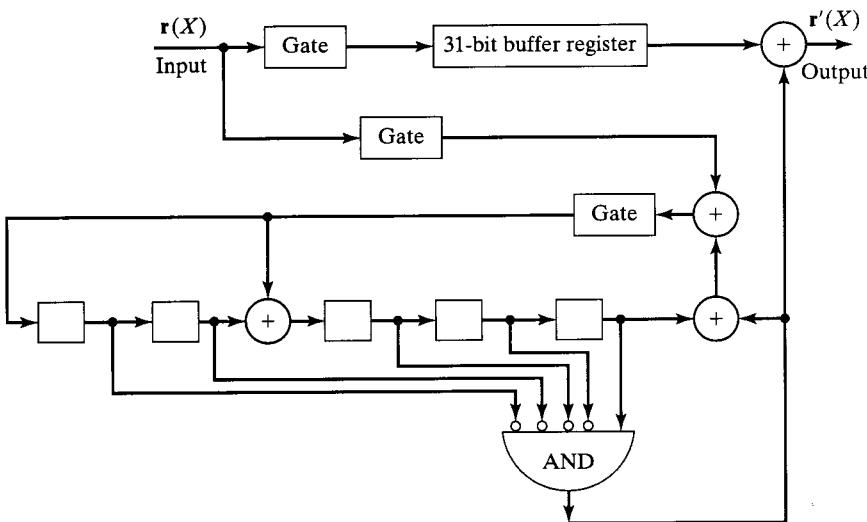


FIGURE 5.20: Decoding circuit for the (31, 26) cyclic Hamming code generated by $g(X) = 1 + X^2 + X^5$.

EXAMPLE 5.12

For $m = 5$, there exists a (31, 26) cyclic Hamming code generated by $g(X) = 1 + X^2 + X^5$. Suppose that it is shortened by three digits. The resultant shortened code is a (28, 23) linear code. The decoding circuit for the (31, 26) cyclic code is shown in Figure 5.20.

This circuit can be used to decode the (28, 23) shortened code. To eliminate the extra shifts of the syndrome register, we need to modify the connections of the syndrome register. First, we need to determine the polynomial $\rho(X)$. Dividing X^{n-k+3} by $g(X) = 1 + X^2 + X^5$, we have

$$\begin{array}{r} X^3 + 1 \\ \hline X^5 + X^2 + 1 | X^8 \\ X^8 + X^5 + X^3 \\ \hline X^5 + X^3 \\ X^5 + X^2 + 1 \\ \hline X^3 + X^2 + 1 \end{array}$$

and $\rho(X) = 1 + X^2 + X^3$. The modified decoding circuit for the (28, 23) shortened code is shown in Figure 5.21.

The extra l shifts of the syndrome register for decoding the shortened cyclic code can also be avoided by modifying the error-pattern detection circuit of the decoder for the original cyclic code.

The error-pattern detection circuit is redesigned to check whether the syndrome in the syndrome register corresponds to a correctable error pattern $e(X)$.

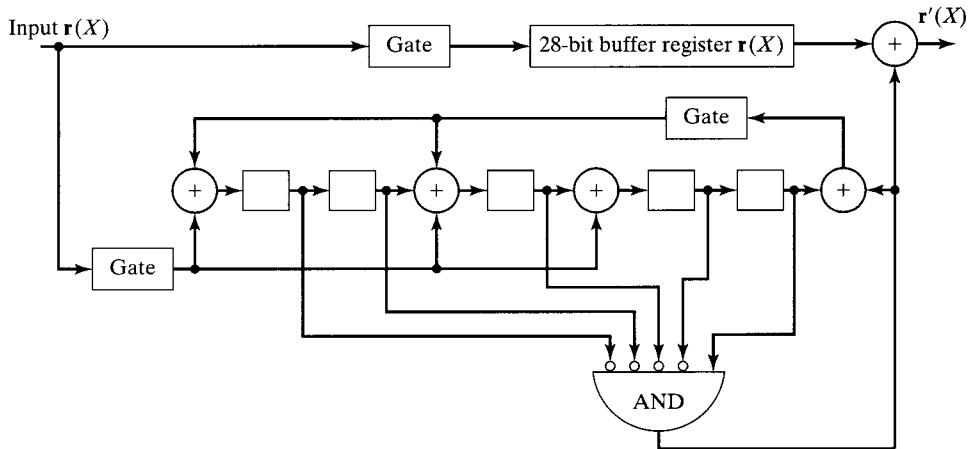


FIGURE 5.21: Decoding circuit for the (28, 23) shortened cyclic code generated by $\mathbf{g}(X) = 1 + X^2 + X^5$.

with an error at position X^{n-l-1} (i.e., $e_{n-l-1} = 1$). When the received digit r_{n-l-1} is corrected, the effect of the error digit e_{n-l-1} on the syndrome should be removed. Suppose that the received vector is shifted into the syndrome register from the right end. Let $\rho(X) = \rho_0 + \rho_1 X + \dots + \rho_{n-k-1} X^{n-k-1}$ be the remainder resulting from dividing $X^{n-l-1} \cdot X^{n-k} = X^{2n-k-l-1}$ by the generator polynomial $\mathbf{g}(X)$. Then, the effect of the error digit e_{n-l-1} on the syndrome is removed by adding $\rho(X)$ to the syndrome in the syndrome register.

EXAMPLE 5.13

Consider the (28, 23) shortened cyclic code obtained by deleting three digits from the (31, 26) cyclic Hamming code generated by $\mathbf{g}(X) = 1 + X^2 + X^5$. Suppose that in decoding this code the received vector is shifted into the syndrome register from the right end. If a single error occurs at the position X^{27} [or $\mathbf{e}(X) = X^{27}$], the syndrome corresponding to this error pattern is the remainder resulting from dividing $X^5 \mathbf{e}(X) = X^{32}$ by $\mathbf{g}(X) = 1 + X^2 + X^5$. The resultant syndrome is (01000). Thus, in decoding the (28, 23) shortened Hamming code, the error-pattern detection circuit may be designed to check whether the syndrome in the syndrome register is (01000). Doing so avoids the extra three shifts of the syndrome register. The resultant decoding circuit with syndrome resetting is shown in Figure 5.22.

Shortened cyclic codes for error detection in conjunction with ARQ protocols are widely used for error control, particularly in computer communications. In these applications they are often called *cyclic redundancy check* (CRC) codes. A CRC code is, in general, generated by either a primitive polynomial $\mathbf{p}(X)$ or a polynomial $\mathbf{g}(X) = (X + 1)\mathbf{p}(X)$. A number of CRC codes have become international standards for error detection in various contexts. A few standard CRC codes follow:

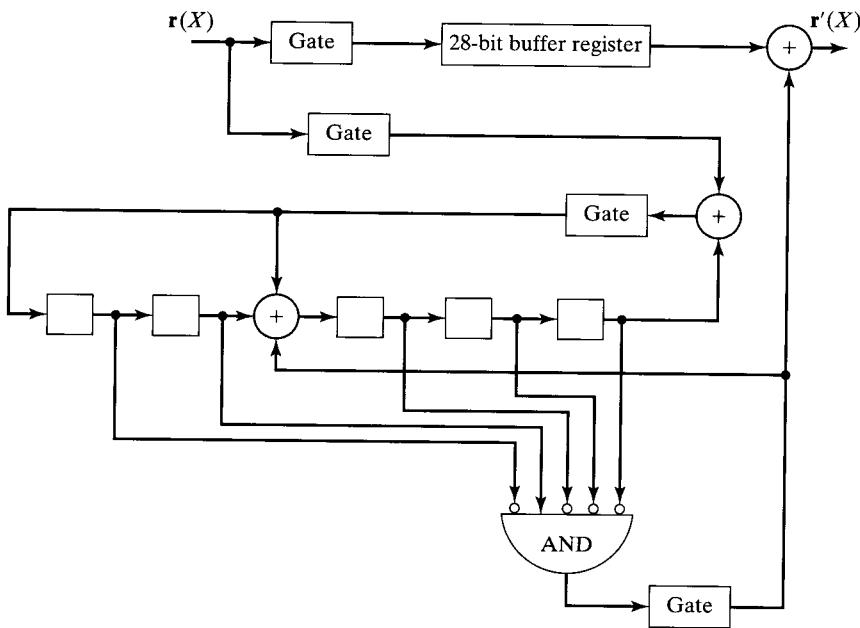


FIGURE 5.22: Another decoding circuit for the (28, 23) shortened Hamming code generated by $\mathbf{g}(X) = 1 + X^2 + X^5$.

CCITT X-25 (Consultative Committee for International Telegraphy and Telephony, Recommendation X-25)

$$\begin{aligned}\mathbf{g}(X) &= (X + 1)(X^{15} + X^{14} + X^{13} + X^{12} + X^4 + X^3 + X^2 + X + 1) \\ &= X^{16} + X^{12} + X^5 + 1,\end{aligned}$$

ANSI (American National Standards Institute)

$$\mathbf{g}(X) = (X + 1)(X^{15} + X + 1) = X^{16} + X^{15} + X^2 + 1,$$

IBM-SDLC (IBM Synchronous Data Link Control)

$$\begin{aligned}\mathbf{g}(X) &= (X + 1)^2(X^{14} + X^{13} + X^{12} + X^{10} + X^8 + X^6 + X^5 + X^4 + X^3 + X + 1) \\ &= X^{16} + X^{15} + X^{13} + X^7 + X^4 + X^2 + X + 1,\end{aligned}$$

IEC TC57

$$\begin{aligned}\mathbf{g}(X) &= (X + 1)^2(X^{14} + X^{10} + X^9 + X^8 + X^5 + X^3 + X^2 + X + 1) \\ &= X^{16} + X^{14} + X^{11} + X^8 + X^6 + X^5 + X^4 + 1,\end{aligned}$$

IEEE Standard 802.3

$$\begin{aligned}\mathbf{g}(X) &= X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} \\ &\quad + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1.\end{aligned}$$

5.11 CYCLIC PRODUCT CODES

The obvious way to implement a product code is to set up the code array and operate on rows and then on columns (or columns and then rows) in encoding and decoding, but there is an alternative that can be extremely attractive. In many cases the product code of cyclic codes is cyclic, and cyclic code implementation is much simpler.

If the component codes C_1 and C_2 are cyclic, and if their lengths, n_1 and n_2 , are relatively prime, the product code $C_1 \times C_2$ is cyclic if the code digits of a code array are transmitted in a proper order [3, 25, 26]. We start with the upper right corner and move down and to the left on a 45° diagonal, as shown in Figure 5.23. When we reach the end of a column, we move to the top of the next column. When we reach the end of a row, we move to the rightmost digit of the next row.

Because n_1 and n_2 are relatively prime, there exists a pair of integers a and b such that

$$an_1 + bn_2 = 1.$$

Let $\mathbf{g}_1(X)$ and $\mathbf{h}_1(X)$ be the generator and parity polynomials of the (n_1, k_1) cyclic code C_1 , and let $\mathbf{g}_2(X)$ and $\mathbf{h}_2(X)$ be the generator and parity polynomials of the (n_2, k_2) cyclic code C_2 . Then, it is possible to show [25, 26] that the generator polynomial $\mathbf{g}(X)$ of the cyclic product code of C_1 and C_2 is the greatest common divisor (GCD) of $X^{n_1 n_2} - 1$ and $\mathbf{g}_1(X^{bn_2})\mathbf{g}_2(X^{an_1})$; that is,

$$\mathbf{g}(X) = \text{GCD}[X^{n_1 n_2} - 1, \mathbf{g}_1(X^{bn_2})\mathbf{g}_2(X^{an_1})], \quad (5.42)$$

and the parity polynomial $\mathbf{h}(X)$ of the cyclic product code is the greatest common divisor of $\mathbf{h}_1(X^{bn_2})$ and $\mathbf{h}_2(X^{an_1})$; that is,

$$\mathbf{h}(X) = \text{GCD}[\mathbf{h}_1(X^{bn_2}), \mathbf{h}_2(X^{an_1})]. \quad (5.43)$$

The complexity of the decoder for cyclic product codes is comparable to the complexity of the decoders for both the (n_1, k_1) code and the (n_2, k_2) code. At the receiving end of the channel, the received vector may again be rearranged as a rectangular array. Thus, the decoder can decode each of the row (or column) codewords separately and then decode each of the column (or row) codewords.

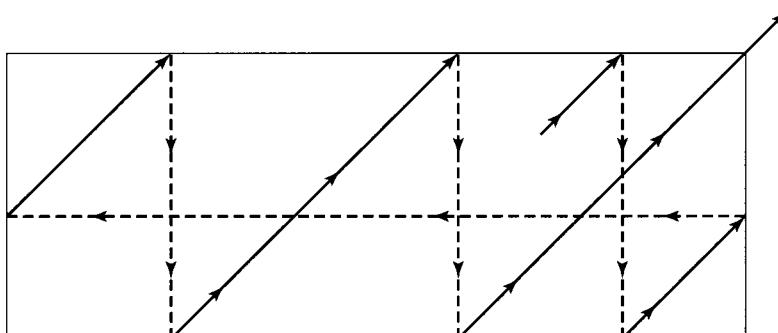


FIGURE 5.23: Transmission of a cyclic product code.

Alternatively, in the transmitted codeword, the set of n_1 digits formed by selecting every (n_2) th digit are the n_1 digits of a codeword of C_1 permuted in a fixed way. They can be permuted back to their original form and corrected by a Meggitt-type decoder. The digits in the permuted form are a codeword in a related code and can be decoded directly in this form by a Meggitt-type decoder. Similarly, the column code C_2 can be corrected by selecting every (n_1) th digit from the large codeword. Thus, the total equipment required is roughly that required to decode the two individual codes.

5.12 QUASI-CYCLIC CODES

Cyclic codes possess full cyclic symmetry; that is, cyclically shifting a codeword any number of symbol positions, either to the right or to the left, results in another codeword. This cyclic symmetry structure makes it possible to implement the encoding and decoding of cyclic codes with simple shift registers and logic circuits. There are other linear block codes that do not possess full cyclic symmetry but do have partial cyclic structure, namely, quasi-cyclic codes.

A *quasi-cyclic code* is a linear code for which cyclically shifting a codeword a fixed number $n_0 \neq 1$ (or a multiple of n_0) of symbol positions either to the right or to the left results in another codeword. It is clear that for $n_0 = 1$, a quasi-cyclic code is a cyclic code. The integer n_0 is called the *shifting constraint*. It is clear that the dual of a quasi-cyclic code is also quasi-cyclic.

As an example, consider the $(9, 3)$ code generated by the following generator matrix:

$$\mathbf{G} = \begin{bmatrix} 111 & 100 & 110 \\ 110 & 111 & 100 \\ 100 & 110 & 111 \end{bmatrix}.$$

The eight codewords of this code are listed in Table 5.6. Suppose we cyclically shift the fifth codeword in Table 5.6, (001011010) three symbol positions to the right. This shift results in the seventh codeword (010001011) in Table 5.6. If we cyclically shift the fifth codeword one and two positions to the right, we obtain two vectors, (000101101) and (100010110) , respectively, which are not codewords given in Table 5.6. Therefore, the $(9, 3)$ code given in Table 5.6 is a quasi-cyclic code with shifting constraint $n_0 = 3$. This code also can be encoded with a shift register as shown in Figure 5.24. Let (c_0, c_1, c_2) be the message to be encoded. As soon as the

TABLE 5.6: The codewords of the $(9, 3)$ quasi-cyclic code.

000	000	000
111	100	110
110	111	100
100	110	111
001	011	010
011	010	001
010	001	011
101	101	101

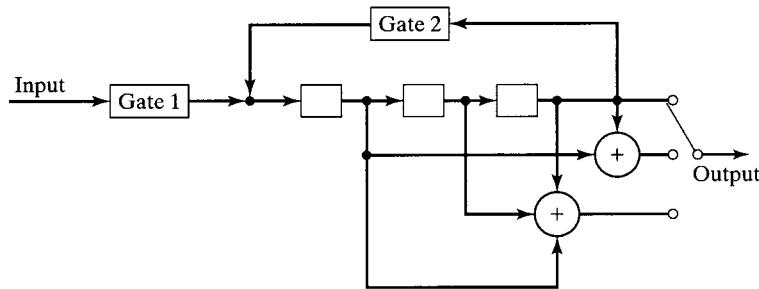


FIGURE 5.24: An encoding circuit for a (9, 3) quasi-cyclic code.

three information symbols have been shifted into the register, gate 1 is deactivated and gate 2 is activated. The information symbol c_2 and two parity-check symbols $p_2^{(1)}$ and $p_2^{(2)}$ appear at the output terminals and then are shifted into the channel. The two parity-check symbols are given by

$$\begin{aligned} p_2^{(1)} &= c_0 + c_2, \\ p_2^{(2)} &= c_0 + c_1 + c_2. \end{aligned}$$

Next, the register is shifted once. The content of the register is now (c_2, c_0, c_1) . The information symbol c_1 and two parity-check symbols $p_1^{(1)}$ and $p_1^{(2)}$ appear at the output terminals, and they are shifted into the channel. The parity-check symbols $p_1^{(1)}$ and $p_1^{(2)}$ are given by

$$\begin{aligned} p_1^{(1)} &= c_1 + c_2, \\ p_1^{(2)} &= c_0 + c_1 + c_2. \end{aligned}$$

At this point, the register is shifted once again. The content of the register is now (c_1, c_2, c_0) , and the information symbol c_0 and two parity-check symbols $p_0^{(1)}$ and $p_0^{(2)}$ appear at the output terminals.

These three symbols are then shifted into the channel. The two parity-check symbols are given by

$$\begin{aligned} p_0^{(1)} &= c_0 + c_1, \\ p_0^{(2)} &= c_0 + c_1 + c_2. \end{aligned}$$

This completes the encoding. The codeword has the form

$$\mathbf{v} = (p_0^{(2)}, p_0^{(1)}, c_0, p_1^{(2)}, p_1^{(1)}, c_1, p_2^{(2)}, p_2^{(1)}, c_2),$$

which consists of three blocks, each of which consists of one unaltered information symbol and two parity-check symbols. This form may also be regarded as a systematic form.

For an (mn_0, mk_0) quasi-cyclic code with shifting constraint n_0 , the generator matrix in systematic form is

$$\mathbf{G} = \begin{bmatrix} \mathbf{P}_0 \mathbf{I} & \mathbf{P}_1 \mathbf{0} & \cdots & \mathbf{P}_{m-1} \mathbf{0} \\ \mathbf{P}_{m-1} \mathbf{0} & \mathbf{P}_0 \mathbf{I} & \cdots & \mathbf{P}_{m-2} \mathbf{0} \\ \vdots & \vdots & & \vdots \\ \mathbf{P}_1 \mathbf{0} & \mathbf{P}_2 \mathbf{0} & \cdots & \mathbf{P}_0 \mathbf{I} \end{bmatrix}, \quad (5.44)$$

where \mathbf{I} and $\mathbf{0}$ represent the $k_0 \times k_0$ identity and zero matrices, respectively, and \mathbf{P}_i is an arbitrary $k_0 \times (n_0 - k_0)$ matrix. Each row (as a sequence of $m k_0 \times n_0$ matrices) is the cyclic shift (to the right) of the row immediately above it, and the row at the top is the cyclic shift of the bottom row. Each column of \mathbf{G} is the downward cyclic shift of the column on its left (or the upward cyclic shift of the column on its right). A message for the code consists of m k_0 -bit blocks, and a codeword consists of m n_0 -bit blocks. Each of these m n_0 -bit blocks consists of k_0 unaltered information symbols and $n_0 - k_0$ parity-check symbols. The parity-check matrix corresponding to the generator matrix given by (5.44) is

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} \mathbf{P}_0^T & \mathbf{0} \mathbf{P}_{m-1}^T & \cdots & \mathbf{0} \mathbf{P}_1^T \\ \mathbf{0} \mathbf{P}_1^T & \mathbf{I} \mathbf{P}_0^T & \cdots & \mathbf{0} \mathbf{P}_2^T \\ \vdots & \vdots & & \vdots \\ \mathbf{0} \mathbf{P}_{m-1}^T & \mathbf{0} \mathbf{P}_{m-2}^T & \cdots & \mathbf{I} \mathbf{P}_0^T \end{bmatrix}, \quad (5.45)$$

where \mathbf{I} and $\mathbf{0}$ represent the $(n_0 - k_0) \times (n_0 - k_0)$ identity and zero matrices, respectively, and \mathbf{P}_i^T is the transpose of \mathbf{P}_i . Consider the $(9, 3)$ quasi-cyclic code given previously for which $k_0 = 1$ and $n_0 = 3$. The parity-check matrix in systematic form is

$$\mathbf{H} = \begin{bmatrix} 101 & 001 & 001 \\ 011 & 001 & 000 \\ 001 & 101 & 001 \\ 000 & 011 & 001 \\ 001 & 001 & 101 \\ 001 & 000 & 011 \end{bmatrix}.$$

A more general form for the generator matrix of an (mn_0, mk_0) quasi-cyclic code is

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_{m-1} \\ \mathbf{G}_{m-1} & \mathbf{G}_0 & \cdots & \mathbf{G}_{m-2} \\ \vdots & \vdots & & \vdots \\ \mathbf{G}_2 & \mathbf{G}_3 & \cdots & \mathbf{G}_1 \\ \mathbf{G}_1 & \mathbf{G}_2 & \cdots & \mathbf{G}_0 \end{bmatrix}, \quad (5.46)$$

where each \mathbf{G}_i is a $k_0 \times n_0$ submatrix. We see that \mathbf{G} given by (5.46) displays the cyclic structure among the rows and columns in terms of the submatrices \mathbf{G}_i 's. For $0 \leq j < m$, let $\mathbf{M}_j \stackrel{\Delta}{=} [\mathbf{G}_j, \mathbf{G}_{j-1}, \dots, \mathbf{G}_{j+1}]^T$ denote the j th column of \mathbf{G} (with $\mathbf{G}_m = \mathbf{G}_0$). \mathbf{M}_j is an $mk_0 \times n_0$ submatrix. Then, we can put \mathbf{G} in the following form:

$$\mathbf{G} = [\mathbf{M}_0, \mathbf{M}_1, \dots, \mathbf{M}_{m-1}].$$

For $0 \leq l < n_0$, let \mathbf{Q}_l be the $mk_0 \times m$ submatrix that is formed by taking the l th columns from $\mathbf{M}_0, \mathbf{M}_1, \dots, \mathbf{M}_{m-1}$. Then, we can put \mathbf{G} in the following form:

$$\mathbf{G}_c = [\mathbf{Q}_0, \mathbf{Q}_1, \dots, \mathbf{Q}_{n_0-1}].$$

Each column of \mathbf{Q}_l consists of mk_0 bits that are regarded as $m k_0$ -bit bytes (a *byte* is a group of k_0 binary digits). In terms of bytes, \mathbf{Q}_l is regarded as an $m \times m$ matrix that has the following cyclic structure: (1) each row is the cyclic shift (to the right) of the row immediately above it, and the top row is the cyclic shift of the bottom row; (2) each column is the downward cyclic shift of the column on its left, and the leftmost column is the downward cyclic shift of the rightmost column. The matrix \mathbf{Q}_l is called a *circulant*. Therefore, \mathbf{G}_c consists of n_0 circulants. Most often, quasi-cyclic codes are studied in circulant form.

EXAMPLE 5.14

Consider the $(15, 5)$ quasi-cyclic code with parameters $m = 5$, $n_0 = 3$, and $k_0 = 1$ that is generated by the following generator matrix:

$$\mathbf{G} = \begin{bmatrix} 001 & 100 & 010 & 110 & 110 \\ 110 & 001 & 100 & 010 & 110 \\ 110 & 110 & 001 & 100 & 010 \\ 010 & 110 & 110 & 001 & 100 \\ 100 & 010 & 110 & 110 & 001 \end{bmatrix}.$$

$\mathbf{M}_0 \quad \mathbf{M}_1 \quad \mathbf{M}_2 \quad \mathbf{M}_3 \quad \mathbf{M}_4$

This quasi-cyclic code has a minimum distance of 7. In circulant form, the generator matrix takes the following form:

$$\mathbf{G} = \begin{bmatrix} 01011 & 00111 & 10000 \\ 10101 & 10011 & 01000 \\ 11010 & 11001 & 00100 \\ 01101 & 11100 & 00010 \\ 10110 & 01110 & 00001 \end{bmatrix}.$$

$\mathbf{Q}_0 \quad \mathbf{Q}_1 \quad \mathbf{Q}_2$

PROBLEMS

- 5.1** Consider the $(15, 11)$ cyclic Hamming code generated by $\mathbf{g}(X) = 1 + X + X^4$.
- Determine the parity polynomial $\mathbf{h}(X)$ of this code.
 - Determine the generator polynomial of its dual code.
 - Find the generator and parity matrices in systematic form for this code.
- 5.2** Devise an encoder and a decoder for the $(15, 11)$ cyclic Hamming code generated by $\mathbf{g}(X) = 1 + X + X^4$.

- 5.3** Show that $\mathbf{g}(X) = 1 + X^2 + X^4 + X^6 + X^7 + X^{10}$ generates a (21, 11) cyclic code. Devise a syndrome computation circuit for this code. Let $\mathbf{r}(X) = 1 + X^5 + X^{17}$ be a received polynomial. Compute the syndrome of $\mathbf{r}(X)$. Display the contents of the syndrome register after each digit of \mathbf{r} has been shifted into the syndrome computation circuit.
- 5.4** Shorten this (15, 11) cyclic Hamming by deleting the seven leading high-order message digits. The resultant code is an (8, 4) shortened cyclic code. Design a decoder for this code that eliminates the extra shifts of the syndrome register.
- 5.5** Shorten the (31, 26) cyclic Hamming code by deleting the 11 leading high-order message digits. The resultant code is a (20, 15) shortened cyclic code. Devise a decoding circuit for this code that requires no extra shifts of the syndrome register.
- 5.6** Let $\mathbf{g}(X)$ be the generator polynomial of a binary cyclic code of length n .
- Show that if $\mathbf{g}(X)$ has $X + 1$ as a factor, the code contains no codewords of odd weight.
 - If n is odd and $X + 1$ is not a factor of $\mathbf{g}(X)$, show that the code contains a codeword consisting of all 1's.
 - Show that the code has a minimum weight of at least 3 if n is the smallest integer such that $\mathbf{g}(X)$ divides $X^n + 1$.
- 5.7** Consider a binary (n, k) cyclic code C generated by $\mathbf{g}(X)$. Let

$$\mathbf{g}^*(X) = X^{n-k} \mathbf{g}(X^{-1})$$

be the reciprocal polynomial of $\mathbf{g}(X)$.

- Show that $\mathbf{g}^*(X)$ also generates an (n, k) cyclic code.
- Let C^* denote the cyclic code generated by $\mathbf{g}^*(X)$. Show that C and C^* have the same weight distribution.

(Hint: Show that

$$\mathbf{v}(X) = v_0 + v_1 X + \cdots + v_{n-2} X^{n-2} + v_{n-1} X^{n-1}$$

is a code polynomial in C if and only if

$$X^{n-1} \mathbf{v}(X^{-1}) = v_{n-1} + v_{n-2} X + \cdots + v_1 X^{n-2} + v_0 X^{n-1}$$

is a code polynomial in C^* .)

- 5.8** Consider a cyclic code C of length n that consists of both odd-weight and even-weight codewords. Let $\mathbf{g}(X)$ and $A(z)$ be the generator polynomial and weight enumerator for this code. Show that the cyclic code generated by $(X + 1)\mathbf{g}(X)$ has weight enumerator

$$A_1(z) = \frac{1}{2}[A(z) + A(-z)].$$

- 5.9** Suppose that the (15, 10) cyclic Hamming code of minimum distance 4 is used for error detection over a BSC with transition probability $p = 10^{-2}$. Compute the probability of an undetected error, $P_u(E)$, for this code.
- 5.10** Consider the $(2^m - 1, 2^m - m - 2)$ cyclic Hamming code C generated by $\mathbf{g}(X) = (X + 1)\mathbf{p}(X)$, where $\mathbf{p}(X)$ is a primitive polynomial of degree m . An error pattern of the form

$$\mathbf{e}(X) = X^i + X^{i+1}$$

is called a *double-adjacent-error pattern*. Show that no two double-adjacent-error patterns can be in the same coset of a standard array for C . Therefore, the code is capable of correcting all the single-error patterns and all the double-adjacent-error patterns.

- 5.11** Devise a decoding circuit for the $(7, 3)$ Hamming code generated by $\mathbf{g}(X) = (X + 1)(X^3 + X + 1)$. The decoding circuit corrects all the single-error patterns and all the double-adjacent-error patterns (see Problem 5.10).
- 5.12** For a cyclic code, if an error pattern $\mathbf{e}(X)$ is detectable, show that its i th cyclic shift $e^{(i)}(X)$ is also detectable.
- 5.13** In the decoding of an (n, k) cyclic code, suppose that the received polynomial $\mathbf{r}(X)$ is shifted into the syndrome register from the right end, as shown in Figure 5.11. Show that when a received digit r_i is detected in error and is corrected, the effect of error digit e_i on the syndrome can be removed by feeding e_i into the syndrome register from the right end, as shown in Figure 5.11.
- 5.14** Let $\mathbf{v}(X)$ be a code polynomial in a cyclic code of length n . Let l be the smallest integer such that

$$\mathbf{v}^{(l)}(X) = \mathbf{v}(X).$$

Show that if $l \neq 0$, l is a factor of n .

- 5.15** Let $\mathbf{g}(X)$ be the generator polynomial of an (n, k) cyclic code C . Suppose C is interleaved to a depth of λ . Prove that the interleaved code C^λ is also cyclic and its generator polynomial is $\mathbf{g}(X^\lambda)$.
- 5.16** Construct all the binary cyclic codes of length 15. (*Hint:* Using the fact that $X^{15} + 1$ has all the nonzero elements of $GF(2^4)$ as roots and using Table 2.9, factor $X^{15} + 1$ as a product of irreducible polynomials.)
- 5.17** Let β be a nonzero element in the Galois field $GF(2^m)$, and $\beta \neq 1$. Let $\phi(X)$ be the minimum polynomial of β . Is there a cyclic code with $\phi(X)$ as the generator polynomial? If your answer is yes, find the shortest cyclic code with $\phi(X)$ as the generator polynomial.
- 5.18** Let β_1 and β_2 be two distinct nonzero elements in $GF(2^m)$. Let $\phi_1(X)$ and $\phi_2(X)$ be the minimal polynomials of β_1 and β_2 , respectively. Is there a cyclic code with $\mathbf{g}(X) = \phi_1(X) \cdot \phi_2(X)$ as the generator polynomial? If your answer is yes, find the shortest cyclic code with $\mathbf{g}(X) = \phi_1(X) \cdot \phi_2(X)$ as the generator polynomial.
- 5.19** Consider the Galois field $GF(2^m)$, which is constructed based on the primitive polynomial $\mathbf{p}(X)$ of degree m . Let α be a primitive element of $GF(2^m)$ whose minimal polynomial is $\mathbf{p}(X)$. Show that every code polynomial in the Hamming code generated by $\mathbf{p}(X)$ has α and its conjugates as roots. Show that any binary polynomial of degree $2^m - 2$ or less that has α as a root is a code polynomial in the Hamming code generated by $\mathbf{p}(X)$.
- 5.20** Let C_1 and C_2 be two cyclic codes of length n that are generated by $\mathbf{g}_1(X)$ and $\mathbf{g}_2(X)$, respectively. Show that the code polynomials common to both C_1 and C_2 also form a cyclic code C_3 . Determine the generator polynomial of C_3 . If d_1 and d_2 are the minimum distances of C_1 and C_2 , respectively, what can you say about the minimum distance of C_3 ?
- 5.21** Show that the probability of an undetected error for the distance-4 cyclic Hamming codes is upper bounded by $2^{-(m+1)}$.
- 5.22** Let C be a $(2^m - 1, 2^m - m - 1)$ Hamming code generated by a primitive polynomial $\mathbf{p}(X)$ of degree m . Let C_d be the dual code of C . Then, C_d is a $(2^m - 1, m)$ cyclic code generated by

$$\mathbf{h}^*(X) = X^{2^m - m - 1} \mathbf{h}(X^{-1}),$$

where

$$\mathbf{h}(X) = \frac{X^{2^m-1} + 1}{\mathbf{p}(X)}.$$

- a. Let $\mathbf{v}(X)$ be a codeword in C_d and let $\mathbf{v}^{(i)}(X)$ be the i th cyclic shift of $\mathbf{v}(X)$. Show that for $1 \leq i \leq 2^m - 2$, $\mathbf{v}^{(i)}(X) \neq \mathbf{v}(X)$.

- b. Show that C_d contains the all-zero codeword and $2^m - 1$ codewords of weight 2^{m-1} .

(Hint: For part (a), use (5.1) and the fact that the smallest integer n such that $X^n + 1$ is divisible by $\mathbf{p}(X)$ is $2^m - 1$. For part (b), use the result of Problem 3.6(b).)

- 5.23 For an (n, k) cyclic code, show that the syndrome of an end-around burst of length $n - k$ cannot be zero.

- 5.24 Design a Meggitt decoder that decodes a received polynomial $\mathbf{r}(X) = r_0 + r_1 X + \dots + r_{n-1} X^{n-1}$ from the lowest-order received digit r_0 to the highest-order received digit r_{n-1} . Describe the decoding operation and the syndrome modification after each correction.

- 5.25 Consider the $(15, 5)$ cyclic code generated by the following polynomial:

$$\mathbf{g}(X) = 1 + X + X^2 + X^4 + X^5 + X^8 + X^{10}.$$

This code has been proved to be capable of correcting any combination of three or fewer errors. Suppose that this code is to be decoded by the simple error-trapping decoding scheme.

- a. Show that all the double errors can be trapped.

- b. Can all the error patterns of three errors be trapped? If not, how many error patterns of three errors cannot be trapped?

- c. Devise a simple error-trapping decoder for this code.

- 5.26 a. Devise a simple error-trapping decoder for the $(23, 12)$ Golay code.

- b. How many error patterns of double errors cannot be trapped?

- c. How many error patterns of three errors cannot be trapped?

- 5.27 Suppose that the $(23, 12)$ Golay code is used only for error correction on a BSC with transition probability p . If Kasami's decoder of Figure 5.18 is used for decoding this code, what is the probability of a decoding error? (Hint: Use the fact that the $(23, 12)$ Golay code is a perfect code.)

- 5.28 Use the decoder of Figure 5.18 to decode the following received polynomials:

a. $\mathbf{r}(X) = X^5 + X^{19}$

b. $\mathbf{r}(X) = X^4 + X^{11} + X^{21}$

At each step in the decoding process, write down the contents of the syndrome register.

- 5.29 Consider the following binary polynomial:

$$\mathbf{g}(X) = (X^3 + 1)\mathbf{p}(X),$$

where $(X^3 + 1)$ and $\mathbf{p}(X)$ are relatively prime, and $\mathbf{p}(X)$ is an irreducible polynomial of degree m with $m \geq 3$. Let n be the smallest integer such that $\mathbf{g}(X)$ divides $X^n + 1$. Thus, $\mathbf{g}(X)$ generates a cyclic code of length n .

- a. Show that this code is capable of correcting all the single-error, double-adjacent-error, and triple-adjacent-error patterns. (*Hint:* Show that these error patterns can be used as coset leaders of a standard array for the code.)
 - b. Devise an error-trapping decoder for this code. The decoder must be capable of correcting all the single-error, double-adjacent-error, and triple-adjacent-error patterns. Design a combinational logic circuit whose output is 1 when the errors are trapped in the appropriate stages of the syndrome register.
 - c. Suppose that $p(X) = 1 + X + X^4$, which is a primitive polynomial of degree 4. Determine the smallest integer n such that $g(X) = (X^3 + 1)p(X)$ divides $X^n + 1$.
- 5.30** Let C_1 be the $(3, 1)$ cyclic code generated by $g_1(X) = 1 + X + X^2$, and let C_2 be the $(7, 3)$ maximum-length code generated by $g_2(X) = 1 + X + X^2 + X^4$. Find the generator and parity polynomials of the cyclic product of C_1 and C_2 . What is the minimum distance of this product code? Discuss its error-correcting capability.
- 5.31** Devise an encoding circuit for the $(15, 5)$ quasi-cyclic code given in Example 5.14.

BIBLIOGRAPHY

1. E. Prange, "Cyclic Error-Correcting Codes in Two Symbols," *AFCRC-TN-57, 103*, Air Force Cambridge Research Center, Cambridge, Mass., September 1957.
2. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968. (Rev. ed., Aegean Park Press, Laguna Hills, Calif., 1984.)
3. W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2d ed., MIT Press, Cambridge, Mass., 1972.
4. F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North Holland, Amsterdam, 1977.
5. R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, Reading, Mass., 1984.
6. R. J. McEliece, *The Theory of Information and Coding*, Addison-Wesley, Reading, Mass., 1977.
7. G. Clark and J. Cain, *Error-Correction Codes for Digital Communications*, Plenum, New York, 1981.
8. S. A. Vanstone and P. C. van Oorschot, *An Introduction to Error Correcting Codes with Applications*, Kluwer Academic, Boston, Mass., 1989.
9. S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, Englewood Cliffs, N.J., 1995.
10. W. W. Peterson, "Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes," *IRE Trans. Inform. Theory*, IT-6, 459–70, September 1960.
11. J. E. Meggitt, "Error Correcting Codes and Their Implementation," *IRE Trans. Inform. Theory*, IT-7: 232–44, October 1961.

12. T. Kasami, "A Decoding Method for Multiple-Error-Correcting Cyclic Codes by Using Threshold Logics," *Conf. Rec. Inform. Process. Soc. Jap.* (in Japanese), Tokyo, November 1961.
13. M. E. Mitchell et al., "Coding and Decoding Operation Research," *G. E. Advanced Electronics Final Report on Contract AF 19 (604)-6183*, Air Force Cambridge Research Labs., Cambridge, Mass., 1961.
14. M. E. Mitchell, "Error-Trap Decoding of Cyclic Codes," *G. E. Report No. 62MCD3*, General Electric Military Communications Dept., Oklahoma City, December 1962.
15. E. Prange, "The Use of Information Sets in Decoding Cyclic Codes," *IEEE Trans. Inform. Theory*, IT-8: 85–80, September 1962.
16. L. Rudolph, "Easily Implemented Error-Correction Encoding-Decoding," *G. E. Report No. 62MCD2*, General Electric Corporation, Oklahoma City, December 1962.
17. T. Kasami, "A Decoding Procedure For Multiple-Error-Correction Cyclic Codes," *IEEE Trans. Inform. Theory*, IT-10: 134–39, April 1964.
18. F. J. MacWilliams, "Permutation Decoding of Systematic Codes," *Bell Syst. Tech. J.*, 43 (p. 1): 485–505, January 1964.
19. L. Rudolph and M. E. Mitchell, "Implementation of Decoders for Cyclic Codes," *IEEE Trans. Inform. Theory*, IT-10: 259–60, July 1964.
20. D. C. Foata, "On a Program for Ray-Chaudhuri's Algorithm for a Minimum Cover of an Abstract Complex," *Commun. ACM*, 4: 504–6, November 1961.
21. I. B. Pyne and E. J. McCluskey, "The Reduction of Redundancy in Solving Prime Implicant Tables," *IRE Trans. Electron. Comput.*, EC-11: 473–82, August 1962.
22. M. J. E. Golay, "Notes on Digital Coding," *Proc. IRE*, 37: 657, June 1949.
23. E. J. Weldon, Jr., "A Comparison of an Interleaved Golay Code and a Three-Dimensional Product Code," *Final Report, USNELC Contract N0095368M5345*, San Diego, CA, August 1968.
24. S. K. Leung-Yan-Cheong, E. R. Barnes, and D. U. Friedman, "On Some Properties of the Undetected Error Probability of Linear Codes," *IEEE Trans. Inform. Theory*, IT-25 (1): 110–12, January 1979.
25. H. O. Burton and E. J. Weldon, Jr., "Cyclic Product Codes," *IEEE Trans. Inform. Theory*, IT-11: 433–40, July 1965.
26. S. Lin and E. J. Weldon, Jr., "Further Results on Cyclic Product Codes," *IEEE Trans. Inform. Theory*, IT-16: 452–59, July 1970.
27. W. C. Gore, "Further Results on Product Codes," *IEEE Trans. Inform. Theory*, IT-16: 446–51, July 1970.
28. N. M. Abramson, "Cascade Decoding of Cyclic Product Codes," *IEEE Trans. Commun. Technol.*, COM-16: 398–402, 1968.

CHAPTER 6

Binary BCH Codes

The Bose, Chaudhuri, and Hocquenghem (BCH) codes form a large class of powerful random error-correcting cyclic codes. This class of codes is a remarkable generalization of the Hamming codes for multiple-error correction. Binary BCH codes were discovered by Hocquenghem in 1959 [1] and independently by Bose and Chaudhuri in 1960 [2]. The cyclic structure of these codes was proved by Peterson in 1960 [3]. Binary BCH codes were generalized to codes in p^m symbols (where p is a prime) by Gorenstein and Zierler in 1961 [4]. Among the nonbinary BCH codes, the most important subclass is the class of Reed–Solomon (RS) codes. The RS codes were discovered by Reed and Solomon in 1960 [5] independently of the work by Hocquenghem, Bose, and Chaudhuri.

The first decoding algorithm for binary BCH codes was devised by Peterson in 1960 [3]. Then, Peterson's algorithm was generalized and refined by Gorenstein and Zierler [4], Chien [6], Forney [7], Berlekamp [8, 9], Massey [10, 11], Burton [12], and others. Among all the decoding algorithms for BCH codes, Berlekamp's iterative algorithm, and Chien's search algorithm are the most efficient ones.

In this chapter we consider primarily a subclass of the binary BCH codes that is the most important subclass from the standpoint of both theory and implementation. Nonbinary BCH codes and Reed–Solomon codes will be discussed in Chapter 7. For a detailed description of the BCH codes, and their algebraic properties and decoding algorithms, the reader is referred to [9] and [13–17].

6.1 BINARY PRIMITIVE BCH CODES

For any positive integers $m (m \geq 3)$ and $t (t < 2^{m-1})$, there exists a binary BCH code with the following parameters:

$$\begin{array}{ll} \text{Block length:} & n = 2^m - 1, \\ \text{Number of parity-check digits:} & n - k \leq mt, \\ \text{Minimum distance:} & d_{\min} \geq 2t + 1. \end{array}$$

Clearly, this code is capable of correcting any combination of t or fewer errors in a block of $n = 2^m - 1$ digits. We call this code a t -error-correcting BCH code. The generator polynomial of this code is specified in terms of its roots from the Galois field $GF(2^m)$. Let α be a primitive element in $GF(2^m)$. The generator polynomial $\mathbf{g}(X)$ of the t -error-correcting BCH code of length $2^m - 1$ is the *lowest-degree polynomial* over $GF(2)$ that has

$$\alpha, \alpha^2, \alpha^3, \dots, \alpha^{2t} \quad (6.1)$$

as its roots [i.e., $\mathbf{g}(\alpha^i) = 0$ for $1 \leq i \leq 2t$]. It follows from Theorem 2.11 that $\mathbf{g}(X)$ has $\alpha, \alpha^2, \dots, \alpha^{2t}$ and their conjugates as all its roots. Let $\phi_i(X)$ be the

minimal polynomial of α^i . Then, $\mathbf{g}(X)$ must be the *least common multiple* (LCM) of $\phi_1(X), \phi_2(X), \dots, \phi_{2t}(X)$, that is,

$$\mathbf{g}(X) = \text{LCM}\{\phi_1(X), \phi_2(X), \dots, \phi_{2t}(X)\}. \quad (6.2)$$

If i is an even integer, it can be expressed as a product of the following form:

$$i = i'2^l,$$

where i' is an odd number, and $l \geq 1$. Then, $\alpha^i = (\alpha^{i'})^{2^l}$ is a conjugate of $\alpha^{i'}$, and therefore α^i and $\alpha^{i'}$ have the same minimal polynomial; that is,

$$\phi_i(X) = \phi_{i'}(X).$$

Hence, every even power of α in the sequence of (6.1) has the same minimal polynomial as some preceding odd power of α in the sequence. As a result, the generator polynomial $\mathbf{g}(X)$ of the binary t -error-correcting BCH code of length $2^m - 1$ given by (6.2) can be reduced to

$$\mathbf{g}(X) = \text{LCM}\{\phi_1(X), \phi_3(X), \dots, \phi_{2t-1}(X)\}. \quad (6.3)$$

Because the degree of each minimal polynomial is m or less, the degree of $\mathbf{g}(X)$ is at most mt ; that is, the number of parity-check digits, $n - k$, of the code is at most equal to mt . There is no simple formula for enumerating $n - k$, but if t is small, $n - k$ is exactly equal to mt [9, 18]. The parameters for all binary BCH codes of length $2^m - 1$ with $m \leq 10$ are given in Table 6.1. The BCH codes just defined are usually called *primitive* (or *narrow-sense*) BCH codes.

TABLE 6.1: BCH codes generated by primitive elements of order less than 2^{10} .

n	k	t	n	k	t	n	k	t
7	4	1	127	50	13	255	71	29
15	11	1		43	14		63	30
	7	2		36	14		55	31
	5	3		29	21		47	42
31	26	1		22	23		45	43
	21	2		15	27		37	45
	16	3		8	31		29	47
	11	5	255	247	1		21	55
	6	7		239	2		13	59
63	57	1		231	3		9	63
	51	2		223	4	511	502	1
	45	3		215	5		493	2
	39	4		207	6		484	3
	36	5		199	7		475	4
	30	6		191	8		466	5
	24	7		187	9		457	6

(continued overleaf)

TABLE 6.1: (continued)

<i>n</i>	<i>k</i>	<i>t</i>	<i>n</i>	<i>k</i>	<i>t</i>	<i>n</i>	<i>k</i>	<i>t</i>
127	18	10		179	10		448	7
	16	11		171	11		439	8
	10	13		163	12		430	9
	7	15		155	13		421	10
	120	1		147	14		412	11
	113	2		139	18		403	12
	106	3		131	19		394	13
	99	4		123	21		385	14
	92	5		115	22		376	15
	85	6		107	23		367	16
511	78	7		99	24		358	18
	71	9		91	25		349	19
	64	10		87	26		340	20
	57	11		79	27		331	21
	322	22	511	166	47	511	10	121
	313	23		157	51	1023	1013	1
	304	25		148	53		1003	2
	295	26		139	54		993	3
	286	27		130	55		983	4
	277	28		121	58		973	5
1023	268	29		112	59		963	6
	259	30		103	61		953	7
	250	31		94	62		943	8
	241	36		85	63		933	9
	238	37		76	85		923	10
	229	38		67	87		913	11
	220	39		58	91		903	12
	211	41		49	93		893	13
	202	42		40	95		883	14
	193	43		31	109		873	15
738	184	45		28	111		863	16
	175	46		19	119		858	17
	848	18	1023	553	52	1023	268	103
	838	19		543	53		258	106
	828	20		533	54		249	107
	818	21		523	55		238	109
	808	22		513	57		228	110
	798	23		503	58		218	111
	788	24		493	59		208	115
	778	25		483	60		203	117
748	768	26		473	61		193	118
	758	27		463	62		183	119
	748	28		453	63		173	122
	738	29		443	73		163	123

TABLE 6.1: (*continued*)

n	k	t	n	k	t	n	k	t
728	30		433	74		153	125	
718	31		423	75		143	126	
708	34		413	77		133	127	
698	35		403	78		123	170	
688	36		393	79		121	171	
678	37		383	82		111	173	
668	38		378	83		101	175	
658	39		368	85		91	181	
648	41		358	86		86	183	
638	42		348	87		76	187	
628	43		338	89		66	189	
618	44		328	90		56	191	
608	45		318	91		46	219	
598	46		308	93		36	223	
588	47		298	94		26	239	
578	49		288	95		16	147	
573	50		278	102		11	255	
563	51							

From (6.3), we see that the single-error-correcting BCH code of length $2^m - 1$ is generated by

$$\mathbf{g}(X) = \phi_1(X).$$

Because α is a primitive element of $GF(2^m)$, $\phi_1(X)$ is a primitive polynomial of degree m . Therefore, the single-error-correcting BCH code of length $2^m - 1$ is a Hamming code.

EXAMPLE 6.1

Let α be a primitive element of the Galois field $GF(2^4)$ given by Table 2.8 such that $1 + \alpha + \alpha^4 = 0$. From Table 2.9 we find that the minimal polynomials of α , α^3 , and α^5 are

$$\phi_1(X) = 1 + X + X^4,$$

$$\phi_3(X) = 1 + X + X^2 + X^3 + X^4,$$

and

$$\phi_5(X) = 1 + X + X^2,$$

respectively. It follows from (6.3) that the double-error-correcting BCH code of length $n = 2^4 - 1 = 15$ is generated by

$$\mathbf{g}(X) = \text{LCM}\{\phi_1(X), \phi_3(X)\}.$$

Because $\phi_1(X)$ and $\phi_3(X)$ are two distinct irreducible polynomials,

$$\begin{aligned}\mathbf{g}(X) &= \phi_1(X)\phi_3(X) \\ &= (1 + X + X^4)(1 + X + X^2 + X^3 + X^4) \\ &= 1 + X^4 + X^6 + X^7 + X^8.\end{aligned}$$

Thus, the code is a (15, 7) cyclic code with $d_{min} \geq 5$. Since the generator polynomial is a code polynomial of weight 5, the minimum distance of this code is exactly 5.

The triple-error-correcting BCH code of length 15 is generated by

$$\begin{aligned}\mathbf{g}(X) &= \text{LCM}\{\phi_1(X), \phi_3(X), \phi_5(X)\} \\ &= (1 + X + X^4)(1 + X + X^2 + X^3 + X^4)(1 + X + X^2) \\ &= 1 + X + X^2 + X^4 + X^5 + X^8 + X^{10}.\end{aligned}$$

This triple-error-correcting BCH code is a (15, 5) cyclic code with $d_{min} \geq 7$. Because the weight of the generator polynomial is 7, the minimum distance of this code is exactly 7.

Using the primitive polynomial $\mathbf{p}(X) = 1 + X + X^6$, we may construct the Galois field $GF(2^6)$, as shown in Table 6.2. The minimal polynomials of the elements

TABLE 6.2: Galois field $GF(2^6)$ with $\mathbf{p}(\alpha) = 1 + \alpha + \alpha^6 = 0$.

0	0		(0 0 0 0 0 0)
1	1		(1 0 0 0 0 0)
α	α		(0 1 0 0 0 0)
α^2		α^2	(0 0 1 0 0 0)
α^3			(0 0 0 1 0 0)
α^4			(0 0 0 0 1 0)
α^5			(0 0 0 0 0 1)
α^6	1	$+$ α	(1 1 0 0 0 0)
α^7		α $+$ α^2	(0 1 1 0 0 0)
α^8		α^2 $+$ α^3	(0 0 1 1 0 0)
α^9			(0 0 0 1 1 0)
α^{10}			(0 0 0 0 1 1)
α^{11}	1	$+$ α	(1 1 0 0 0 1)
α^{12}	1	$+$ α^2	(1 0 1 0 0 0)
α^{13}	α	α^3	(0 1 0 1 0 0)
α^{14}		α^2 $+$ α^4	(0 0 1 0 1 0)
α^{15}		α^3 $+$ α^5	(0 0 0 1 0 1)
α^{16}	1	$+$ α	(1 1 0 0 1 0)
α^{17}	α $+$ α^2		(0 1 1 0 0 1)
α^{18}	1	$+$ α $+$ α^2 $+$ α^3	(1 1 1 1 0 0)
α^{19}	α $+$ α^2 $+$ α^3 $+$ α^4		(0 1 1 1 1 0)
α^{20}	α^2 $+$ α^3 $+$ α^4 $+$ α^5		(0 0 1 1 1 1)

TABLE 6.2: (continued)

α^{21}	1	+	α		+	α^3	+	α^4	+	α^5	(1 1 0 1 1 1)
α^{22}	1			+	α^2			+	α^4	+	α^5
α^{23}	1					+	α^3			+	α^5
α^{24}	1							+	α^4		
α^{25}			α							+	α^5
α^{26}	1	+	α	+	α^2						(1 1 1 0 0 0)
α^{27}			α	+	α^2	+	α^3				(0 1 1 1 0 0)
α^{28}					α^2	+	α^3	+	α^4		(0 0 1 1 1 0)
α^{29}						α^3	+	α^4	+	α^5	(0 0 0 1 1 1)
α^{30}	1	+	α								(1 1 0 0 1 1)
α^{31}	1			+	α^2					+	α^5
α^{32}	1					+	α^3				(1 0 0 1 0 0)
α^{33}			α					α^4			(0 1 0 0 1 0)
α^{34}					α^2				+	α^5	(0 0 1 0 0 1)
α^{35}	1	+	α			+	α^3				(1 1 0 1 0 0)
α^{36}			α	+	α^2			+	α^4		(0 1 1 0 1 0)
α^{37}					α^2		α^3			+	α^5
α^{38}	1	+	α			+	α^3	+	α^4		(1 1 0 1 1 0)
α^{39}			α	+	α^2			+	α^4	+	α^5
α^{40}	1	+	α	+	α^2	+	α^3			+	α^5
α^{41}	1			+	α^2	+	α^3	+	α^4		(1 0 1 1 1 0)
α^{42}			α			+	α^3	+	α^4	+	α^5
α^{43}	1	+	α	+	α^2			+	α^4	+	α^5
α^{44}	1			+	α^2	+	α^3			+	α^5
α^{45}	1					+	α^3	+	α^4		(1 0 0 1 1 0)
α^{46}			α					+	α^4	+	α^5
α^{47}	1	+	α	+	α^2				+	α^5	(1 1 1 0 0 1)
α^{48}	1			+	α^2	+	α^3				(1 0 1 1 0 0)
α^{49}			α			+	α^3	+	α^4		(0 1 0 1 1 0)
α^{50}					α^2			+	α^4		(0 0 1 0 1 1)
α^{51}	1	+	α			+	α^3			+	α^5
α^{52}	1			+	α^2			+	α^4		(1 0 1 0 1 0)
α^{53}			α			+	α^3			+	α^5
α^{54}	1	+	α	+	α^2			+	α^4		(1 1 1 0 1 0)
α^{55}			α	+	α^2	+	α^3			+	α^5
α^{56}	1	+	α	+	α^2	+	α^3	+	α^4		(1 1 1 1 1 0)
α^{57}			α	+	α^2	+	α^3	+	α^4	+	α^5
α^{58}	1	+	α	+	α^2	+	α^3	+	α^4	+	α^5
α^{59}	1			+	α^2	+	α^3	+	α^4	+	α^5
α^{60}	1					+	α^3	+	α^4	+	α^5
α^{61}	1							+	α^4	+	α^5
α^{62}	1								+	α^5	(1 0 0 0 0 1)

$$\alpha^{63} = 1$$

TABLE 6.3: Minimal polynomials of the elements in $GF(2^6)$.

Elements	Minimal polynomials
$\alpha, \alpha^2, \alpha^4, \alpha^{16}, \alpha^{32}$	$1 + X + X^6$
$\alpha^3, \alpha^6, \alpha^{12}\alpha^{24}, \alpha^{48}\alpha^{33}$	$1 + X + X^2 + X^4 + X^6$
$\alpha^5, \alpha^{10}, \alpha^{20}, \alpha^{40}, \alpha^{17}, \alpha^{34}$	$1 + X + X^2 + X^5 + X^6$
$\alpha^7, \alpha^{14}, \alpha^{28}, \alpha^{56}, \alpha^{49}, \alpha^{35}$	$1 + X^3 + X^6$
$\alpha^9, \alpha^{18}, \alpha^{36}$	$1 + X^2 + X^3$
$\alpha^{11}, \alpha^{22}, \alpha^{44}, \alpha^{25}, \alpha^{50}, \alpha^{37}$	$1 + X^2 + X^3 + X^5 + X^6$
$\alpha^{13}, \alpha^{26}, \alpha^{52}, \alpha^{41}, \alpha^{19}, \alpha^{38}$	$1 + X + X^3 + X^4 + X^6$
$\alpha^{15}, \alpha^{30}, \alpha^{60}, \alpha^{57}, \alpha^{51}, \alpha^{39}$	$1 + X^2 + X^4 + X^5 + X^6$
α^{21}, α^{42}	$1 + X + X^2$
$\alpha^{23}, \alpha^{46}, \alpha^{29}, \alpha^{58}, \alpha^{53}, \alpha^{43}$	$1 + X + X^4 + X^5 + X^6$
$\alpha^{27}, \alpha^{54}, \alpha^{45}$	$1 + X + X^3$
$\alpha^{31}, \alpha^{62}, \alpha^{61}, \alpha^{59}, \alpha^{55}, \alpha^{47}$	$1 + X^5 + X^6$

TABLE 6.4: Generator polynomials of all the BCH codes of length 63.

n	k	t	$\mathbf{g}(X)$
63	57	1	$\mathbf{g}_1(X) = 1 + X + X^6$
	51	2	$\mathbf{g}_2(X) = (1 + X + X^6)(1 + X + X^2 + X^4 + X^6)$
	45	3	$\mathbf{g}_3(X) = (1 + X + X^2 + X^5 + X^6)\mathbf{g}_2(X)$
	39	4	$\mathbf{g}_4(X) = (1 + X^3 + X^6)\mathbf{g}_3(X)$
	36	5	$\mathbf{g}_5(X) = (1 + X^2 + X^3)\mathbf{g}_4(X)$
	30	6	$\mathbf{g}_6(X) = (1 + X^2 + X^3 + X^5 + X^6)\mathbf{g}_5(X)$
	24	7	$\mathbf{g}_7(X) = (1 + X + X^3 + X^4 + X^6)\mathbf{g}_6(X)$
	18	10	$\mathbf{g}_{10}(X) = (1 + X^2 + X^4 + X^5 + X^6)\mathbf{g}_7(X)$
	16	11	$\mathbf{g}_{11}(X) = (1 + X + X^2)\mathbf{g}_{10}(X)$
	10	13	$\mathbf{g}_{13}(X) = (1 + X + X^4 + X^5 + X^6)\mathbf{g}_{11}(X)$
	7	15	$\mathbf{g}_{15}(X) = (1 + X + X^3)\mathbf{g}_{13}(X)$

in $GF(2^6)$ are listed in Table 6.3. Using (6.3), we find the generator polynomials of all the BCH codes of length 63, as shown in Table 6.4. The generator polynomials of all binary primitive BCH codes of length $2^m - 1$ with $m \leq 10$ are given in Appendix C.

It follows from the definition of a t -error-correcting BCH code of length $n = 2^m - 1$ that each code polynomial has $\alpha, \alpha^2, \dots, \alpha^{2t}$ and their conjugates as roots. Now, let $\mathbf{v}(X) = v_0 + v_1X + \dots + v_{n-1}X^{n-1}$ be a polynomial with coefficients from $GF(2)$. If $\mathbf{v}(X)$ has $\alpha, \alpha^2, \dots, \alpha^{2t}$ as roots, it follows from Theorem 2.14 that $\mathbf{v}(X)$ is divisible by the minimal polynomials $\phi_1(X), \phi_2(X), \dots, \phi_{2t}(X)$ of $\alpha, \alpha^2, \dots, \alpha^{2t}$. Obviously, $\mathbf{v}(X)$ is divisible by their least common multiple (the generator polynomial),

$$\mathbf{g}(X) = \text{LCM} \{\phi_1(X), \phi_2(X), \dots, \phi_{2t}(X)\}.$$

Hence, $\mathbf{v}(X)$ is a code polynomial. Consequently, we may define a t -error-correcting BCH code of length $n = 2^m - 1$ in the following manner: a binary n -tuple $\mathbf{v} = (v_0, v_1, v_2, \dots, v_{n-1})$ is a codeword if and only if the polynomial $\mathbf{v}(X) = v_0 + v_1X + \dots + v_{n-1}X^{n-1}$ has $\alpha, \alpha^2, \dots, \alpha^{2t}$ as roots. This definition is useful in proving the minimum distance of the code.

Let $\mathbf{v}(X) = v_0 + v_1X + \dots + v_{n-1}X^{n-1}$ be a code polynomial in a t -error-correcting BCH code of length $n = 2^m - 1$. Because α^i is a root of $\mathbf{v}(X)$ for $1 \leq i \leq 2t$, then

$$\mathbf{v}(\alpha^i) = v_0 + v_1\alpha^i + v_2\alpha^{2i} + \dots + v_{n-1}\alpha^{(n-1)i} = 0. \quad (6.4)$$

This equality can be written as a matrix product as follows:

$$(v_0, v_1, \dots, v_{n-1}) \cdot \begin{bmatrix} 1 \\ \alpha^i \\ \alpha^{2i} \\ \vdots \\ \alpha^{(n-1)i} \end{bmatrix} = 0 \quad (6.5)$$

for $1 \leq i \leq 2t$. The condition given by (6.5) simply says that the inner product of $(v_0, v_1, \dots, v_{n-1})$ and $(1, \alpha^i, \alpha^{2i}, \dots, \alpha^{(n-1)i})$ is equal to zero. Now, we form the following matrix:

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \dots & \alpha^{n-1} \\ 1 & (\alpha^2) & (\alpha^2)^2 & (\alpha^2)^3 & \dots & (\alpha^2)^{n-1} \\ 1 & (\alpha^3) & (\alpha^3)^2 & (\alpha^3)^3 & \dots & (\alpha^3)^{n-1} \\ \vdots & & & & & \vdots \\ 1 & (\alpha^{2t}) & (\alpha^{2t})^2 & (\alpha^{2t})^3 & \dots & (\alpha^{2t})^{n-1} \end{bmatrix}. \quad (6.6)$$

It follows from (6.5) that if $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ is a codeword in the t -error-correcting BCH code, then

$$\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}. \quad (6.7)$$

On the other hand, if an n -tuple $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ satisfies the condition of (6.7), it follows from (6.5) and (6.4) that, for $1 \leq i \leq 2t$, α^i is a root of the polynomial $\mathbf{v}(X)$. Therefore, \mathbf{v} must be a codeword in the t -error-correcting BCH code. Hence, the code is the null space of the matrix \mathbf{H} , and \mathbf{H} is a parity-check matrix of the code. If for some i and j , α^j is a conjugate of α^i , then $\mathbf{v}(\alpha^j) = 0$ if and only if $\mathbf{v}(\alpha^i) = 0$ (see Theorem 2.11); that is, if the inner product of $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ and the i th row of \mathbf{H} is zero, the inner product of \mathbf{v} and the j th row of \mathbf{H} is also zero. For this reason, the j th row of \mathbf{H} can be omitted. As a result, the \mathbf{H} matrix given by (6.6) can be reduced to the following form:

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \dots & \alpha^{n-1} \\ 1 & (\alpha^3) & (\alpha^3)^2 & (\alpha^3)^3 & \dots & (\alpha^3)^{n-1} \\ 1 & (\alpha^5) & (\alpha^5)^2 & (\alpha^5)^3 & \dots & (\alpha^5)^{n-1} \\ \vdots & & & & & \vdots \\ 1 & (\alpha^{2t-1}) & (\alpha^{2t-1})^2 & (\alpha^{2t-1})^3 & \dots & (\alpha^{2t-1})^{n-1} \end{bmatrix}. \quad (6.8)$$

Note that the entries of \mathbf{H} are elements in $GF(2^m)$. We can represent each element in $GF(2^m)$ by an m -tuple over $GF(2)$. If we replace each entry of \mathbf{H} with its corresponding m -tuple over $GF(2)$ arranged in column form, we obtain a binary parity-check matrix for the code.

EXAMPLE 6.2

Consider the double-error-correcting BCH code of length $n = 2^4 - 1 = 15$. From Example 6.1 we know that this is a $(15, 7)$ code. Let α be a primitive element in $GF(2^4)$. Then, the parity-check matrix of this code is

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & \alpha^{15} & \alpha^{18} & \alpha^{21} & \alpha^{24} & \alpha^{27} & \alpha^{30} & \alpha^{33} & \alpha^{36} & \alpha^{39} & \alpha^{42} \end{bmatrix}$$

[by (6.8)]. Using Table 2.8 and the fact that $\alpha^{15} = 1$, and representing each entry of \mathbf{H} with its corresponding 4-tuple, we obtain the following binary parity-check matrix for the code:

$$\mathbf{H} = \left[\begin{array}{cccccccccccccccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{array} \right].$$

Now, we are ready to prove that the t -error-correcting BCH code just defined indeed has a minimum distance of at least $2t + 1$. To prove this, we need to show that no $2t$ or fewer columns of \mathbf{H} given by (6.6) sum to zero (Corollary 3.2.1). Suppose that there exists a nonzero codeword $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ with weight $\delta \leq 2t$. Let $v_{j_1}, v_{j_2}, \dots, v_{j_\delta}$ be the nonzero components of \mathbf{v} (i.e., $v_{j_1} = v_{j_2} = \dots = v_{j_\delta} = 1$). Using (6.6) and (6.7), we have

$$\mathbf{0} = \mathbf{v} \cdot \mathbf{H}^T$$

$$= (v_{j_1}, v_{j_2}, \dots, v_{j_\delta}) \cdot \begin{bmatrix} \alpha^{j_1} & (\alpha^2)^{j_1} & \cdots & (\alpha^{2t})^{j_1} \\ \alpha^{j_2} & (\alpha^2)^{j_2} & \cdots & (\alpha^{2t})^{j_2} \\ \alpha^{j_3} & (\alpha^2)^{j_3} & \cdots & (\alpha^{2t})^{j_3} \\ \vdots & \vdots & & \vdots \\ \alpha^{j_\delta} & (\alpha^2)^{j_\delta} & \cdots & (\alpha^{2t})^{j_\delta} \end{bmatrix}$$

$$= (1, 1, \dots, 1) \cdot \begin{bmatrix} \alpha^{j_1} & (\alpha^{j_1})^2 & \dots & (\alpha^{j_1})^{2t} \\ \alpha^{j_2} & (\alpha^{j_2})^2 & \dots & (\alpha^{j_2})^{2t} \\ \alpha^{j_3} & (\alpha^{j_3})^2 & \dots & (\alpha^{j_3})^{2t} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \alpha^{j_\delta} & (\alpha^{j_\delta})^2 & \dots & (\alpha^{j_\delta})^{2t} \end{bmatrix}.$$

The preceding equality implies the following equality:

$$(1, 1, \dots, 1) \cdot \begin{bmatrix} \alpha^{j_1} & (\alpha^{j_1})^2 & \dots & (\alpha^{j_1})^\delta \\ \alpha^{j_2} & (\alpha^{j_2})^2 & \dots & (\alpha^{j_2})^\delta \\ \alpha^{j_3} & (\alpha^{j_3})^2 & \dots & (\alpha^{j_3})^\delta \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \alpha^{j_\delta} & (\alpha^{j_\delta})^2 & \dots & (\alpha^{j_\delta})^\delta \end{bmatrix} = \mathbf{0}, \quad (6.9)$$

where the second matrix on the left is a $\delta \times \delta$ square matrix. To satisfy the equality of (6.9), the *determinant* of the $\delta \times \delta$ matrix must be zero; that is,

$$\begin{vmatrix} \alpha^{j_1} & (\alpha^{j_1})^2 & \dots & (\alpha^{j_1})^\delta \\ \alpha^{j_2} & (\alpha^{j_2})^2 & \dots & (\alpha^{j_2})^\delta \\ \alpha^{j_3} & (\alpha^{j_3})^2 & \dots & (\alpha^{j_3})^\delta \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \alpha^{j_\delta} & (\alpha^{j_\delta})^2 & \dots & (\alpha^{j_\delta})^\delta \end{vmatrix} = 0.$$

Taking out the common factor from each row of the foregoing determinant, we obtain

$$\alpha^{(j_1+j_2+\dots+j_\delta)} \cdot \begin{vmatrix} 1 & \alpha^{j_1} & \dots & \alpha^{(\delta-1)j_1} \\ 1 & \alpha^{j_2} & \dots & \alpha^{(\delta-1)j_2} \\ 1 & \alpha^{j_3} & \dots & \alpha^{(\delta-1)j_3} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ 1 & \alpha^{j_\delta} & \dots & \alpha^{(\delta-1)j_\delta} \end{vmatrix} = 0. \quad (6.10)$$

The determinant in the preceding equality is a *Vandermonde determinant* that is *nonzero*. Therefore, the product on the left-hand side of (6.10) cannot be zero. This is a contradiction, and hence our assumption that there exists a nonzero codeword \mathbf{v} of weight $\delta \leq 2t$ is *invalid*. This implies that the minimum weight of the t -error-correcting BCH code defined previously is at least $2t + 1$. Consequently, the minimum distance of the code is at least $2t + 1$.

The parameter $2t + 1$ is usually called the *designed distance* of the t -error-correcting BCH code. The true minimum distance of a BCH code may or may not be equal to its designed distance. In many cases the true minimum distance of a

BCH code is equal to its designed distance; however, there are also cases in which the true minimum distance is greater than the designed distance.

Binary BCH codes with length $n \neq 2^m - 1$ can be constructed in the same manner as for the case $n = 2^m - 1$. Let β be an element of order n in the field $GF(2^m)$. We know that n is a factor of $2^m - 1$. Let $\mathbf{g}(X)$ be the binary polynomial of minimum degree that has

$$\beta, \beta^2, \dots, \beta^{2t}$$

as roots. Let $\Psi_1(X), \Psi_2(X), \dots, \Psi_{2t}(X)$ be the minimal polynomials of $\beta, \beta^2, \dots, \beta^{2t}$, respectively. Then,

$$\mathbf{g}(X) = \text{LCM} \{\Psi_1(X), \Psi_2(X), \dots, \Psi_{2t}(X)\}.$$

Because $\beta^n = 1, \beta, \beta^2, \dots, \beta^{2t}$ are roots of $X^n + 1$. We see that $\mathbf{g}(X)$ is a factor of $X^n + 1$. The cyclic code generated by $\mathbf{g}(X)$ is a t -error-correcting BCH code of length n . In a manner similar to that used for the case $n = 2^m - 1$, we can prove that the number of parity-check digits of this code is at most mt , and the minimum distance of the code is at least $2t + 1$. If β is not a primitive element of $GF(2^m)$, $n \neq 2^m - 1$, and the code is called a *nonprimitive* BCH code.

EXAMPLE 6.3

Consider the Galois field $GF(2^6)$ given in Table 6.2. The element $\beta = \alpha^3$ has order $n = 21$. Let $t = 2$. Let $\mathbf{g}(X)$ be the binary polynomial of minimum degree that has

$$\beta, \beta^2, \beta^3, \beta^4$$

as roots. The elements β, β^2 , and β^4 have the same minimal polynomial, which is

$$\Psi_1(X) = 1 + X + X^2 + X^4 + X^6.$$

The minimal polynomial of β^3 is

$$\Psi_3(X) = 1 + X^2 + X^3.$$

Therefore,

$$\begin{aligned}\mathbf{g}(X) &= \Psi_1(X)\Psi_3(X) \\ &= 1 + X + X^4 + X^5 + X^7 + X^8 + X^9.\end{aligned}$$

We can check easily that $\mathbf{g}(X)$ divides $X^{21} + 1$. The $(21, 12)$ code generated by $\mathbf{g}(X)$ is a double-error-correcting nonprimitive BCH code.

Now, we give a general definition of binary BCH codes. Let β be an element of $GF(2^m)$. Let l_0 be any nonnegative integer. Then, a binary BCH code with designed distance d_0 is generated by the binary polynomial $\mathbf{g}(X)$ of minimum degree that has as roots the following consecutive powers of β :

$$\beta^{l_0}, \beta^{l_0+1}, \dots, \beta^{l_0+d_0-2}$$

For $0 \leq i < d_0 - 1$, let $\Psi_i(X)$ and n_i be the minimum polynomial and order of β^{l_0+i} , respectively. Then,

$$\mathbf{g}(X) = \text{LCM}\{\Psi_0(X), \Psi_1(X), \dots, \Psi_{d_0-2}(X)\}$$

and the length of the code is

$$n = \text{LCM}\{n_0, n_1, \dots, n_{d_0-2}\}.$$

The BCH code just defined has a minimum distance of at least d_0 and has no more than $m(d_0 - 1)$ parity-check digits (the proof of these is left as an exercise). Of course, the code is capable of correcting $\lfloor (d_0 - 1)/2 \rfloor$ or fewer errors. If we let $l_0 = 1$, $d_0 = 2t + 1$, and β be a primitive element of $GF(2^m)$, the code becomes a t -error-correcting primitive BCH code of length $2^m - 1$. If $l_0 = 1$, $d_0 = 2t + 1$, and β is not a primitive element of $GF(2^m)$, the code is a nonprimitive t -error-correcting BCH code of length n that is the order of β . We note that in the definition of a BCH code with designed distance d_0 , we require that the generator polynomial $\mathbf{g}(X)$ has $d_0 - 1$ consecutive powers of a field element β as roots. This requirement guarantees that the code has a minimum distance of at least d_0 . This lower bound on the minimum distance is called the *BCH bound*. Any cyclic code whose generator polynomial has $d - 1$ consecutive powers of a field element as roots has a minimum distance of at least d .

In the rest of this chapter, we consider only the primitive BCH codes.

6.2 DECODING OF BCH CODES

Suppose that a codeword $\mathbf{v}(X) = v_0 + v_1X + v_2X^2 + \dots + v_{n-1}X^{n-1}$ is transmitted, and the transmission errors result in the following received vector:

$$\mathbf{r}(X) = r_0 + r_1X + r_2X^2 + \dots + r_{n-1}X^{n-1}.$$

Let $\mathbf{e}(X)$ be the error pattern. Then,

$$\mathbf{r}(X) = \mathbf{v}(X) + \mathbf{e}(X). \quad (6.11)$$

As usual, the first step of decoding a code is to compute the syndrome from the received vector $\mathbf{r}(X)$. For decoding a t -error-correcting primitive BCH code, the syndrome is a $2t$ -tuple:

$$\mathbf{S} = (S_1, S_2, \dots, S_{2t}) = \mathbf{r} \cdot \mathbf{H}^T, \quad (6.12)$$

where \mathbf{H} is given by (6.6). From (6.6) and (6.12) we find that the i th component of the syndrome is

$$\begin{aligned} S_i &= \mathbf{r}(\alpha^i) \\ &= r_0 + r_1\alpha^i + r_2\alpha^{2i} + \dots + r_{n-1}\alpha^{(n-1)i} \end{aligned} \quad (6.13)$$

for $1 \leq i \leq 2t$. Note that the syndrome components are elements in the field $GF(2^m)$. We can compute from $\mathbf{r}(X)$ as follows. Dividing $\mathbf{r}(X)$ by the minimal polynomial $\phi_i(X)$ of α^i , we obtain

$$\mathbf{r}(X) = \mathbf{a}_i(X)\phi_i(X) + \mathbf{b}_i(X),$$

where $\mathbf{b}_i(X)$ is the remainder with degree less than that of $\phi_i(X)$. Because $\phi_i(\alpha^i) = 0$, we have

$$S_i = \mathbf{r}(\alpha^i) = \mathbf{b}_i(\alpha^i). \quad (6.14)$$

Thus, the syndrome component S_i can be obtained by evaluating $\mathbf{b}_i(X)$ with $X = \alpha^i$.

EXAMPLE 6.4

Consider the double-error-correcting (15, 7) BCH code given in Example 6.1. Suppose that the vector

$$\mathbf{r} = (1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0)$$

is received. The corresponding polynomial is

$$\mathbf{r}(X) = 1 + X^8.$$

The syndrome consists of four components:

$$\mathbf{S} = (S_1, S_2, S_3, S_4).$$

The minimal polynomials for α , α^2 , and α^4 are identical, and

$$\phi_1(X) = \phi_2(X) = \phi_4(X) = 1 + X + X^4.$$

The minimal polynomial of α^3 is

$$\phi_3(X) = 1 + X + X^2 + X^3 + X^4.$$

Dividing $\mathbf{r}(X) = 1 + X^8$ by $\phi_1(X) = 1 + X + X^4$, we obtain the remainder

$$\mathbf{b}_1(X) = X^2.$$

Dividing $\mathbf{r}(X) = 1 + X^8$ by $\phi_3(X) = 1 + X + X^2 + X^3 + X^4$, we have the remainder

$$\mathbf{b}_3(X) = 1 + X^3.$$

Using $GF(2^4)$ given by Table 2.8 and substituting α , α^2 , and α^4 into $\mathbf{b}_1(X)$, we obtain

$$S_1 = \alpha^2, \quad S_2 = \alpha^4, \quad S_4 = \alpha^8.$$

Substituting α^3 into $\mathbf{b}_3(X)$, we obtain

$$S_3 = 1 + \alpha^9 = 1 + \alpha + \alpha^3 = \alpha^7.$$

Thus,

$$S = (\alpha^2, \alpha^4, \alpha^7, \alpha^8).$$

Because $\alpha^1, \alpha^2, \dots, \alpha^{2t}$ are roots of each code polynomial, $\mathbf{v}(\alpha^i) = 0$ for $1 \leq i \leq 2t$. The following relationship between the syndrome and the error pattern follows from (6.11) and (6.13):

$$S_i = \mathbf{e}(\alpha^i) \quad (6.15)$$

for $1 \leq i \leq 2t$. From (6.15) we see that the syndrome \mathbf{S} depends on the error pattern \mathbf{e} only. Suppose that the error pattern $\mathbf{e}(X)$ has v errors at locations $X^{j_1}, X^{j_2}, \dots, X^{j_v}$; that is,

$$\mathbf{e}(X) = X^{j_1} + X^{j_2} + \dots + X^{j_v}, \quad (6.16)$$

where $0 \leq j_1 < j_2 < \dots < j_v < n$. From (6.15) and (6.16) we obtain the following set of equations:

$$\begin{aligned} S_1 &= \alpha^{j_1} + \alpha^{j_2} + \dots + \alpha^{j_v} \\ S_2 &= (\alpha^{j_1})^2 + (\alpha^{j_2})^2 + \dots + (\alpha^{j_v})^2 \\ S_3 &= (\alpha^{j_1})^3 + (\alpha^{j_2})^3 + \dots + (\alpha^{j_v})^3 \\ &\vdots \\ S_{2t} &= (\alpha^{j_1})^{2t} + (\alpha^{j_2})^{2t} + \dots + (\alpha^{j_v})^{2t}, \end{aligned} \quad (6.17)$$

where $\alpha^{j_1}, \alpha^{j_2}, \dots, \alpha^{j_v}$ are unknown. Any method for solving these equations is a decoding algorithm for the BCH codes. Once we have found $\alpha^{j_1}, \alpha^{j_2}, \dots, \alpha^{j_v}$, the powers j_1, j_2, \dots, j_v tell us the error locations in $\mathbf{e}(X)$, as in (6.16). In general, the equations of (6.17) have many possible solutions (2^k of them). Each solution yields a different error pattern. If the number of errors in the actual error pattern $\mathbf{e}(X)$ is t or fewer (i.e., $v \leq t$), the solution that yields an error pattern with the smallest number of errors is the right solution; that is, the error pattern corresponding to this solution is the most probable error pattern $\mathbf{e}(X)$ caused by the channel noise. For large t , solving the equations of (6.17) directly is difficult and ineffective. In the following, we describe an effective procedure for determining α^{j_l} for $l = 1, 2, \dots, v$ from the syndrome components S_i 's.

For convenience, let

$$\beta_l = \alpha^{j_l} \quad (6.18)$$

for $1 \leq l \leq v$. We call these elements the *error location numbers*, since they tell us the locations of the errors. Now, we can express the equations of (6.17) in the following form:

$$\begin{aligned} S_1 &= \beta_1 + \beta_2 + \dots + \beta_v \\ S_2 &= \beta_1^2 + \beta_2^2 + \dots + \beta_v^2 \\ &\vdots \\ S_{2t} &= \beta_1^{2t} + \beta_2^{2t} + \dots + \beta_v^{2t}. \end{aligned} \quad (6.19)$$

These $2t$ equations are symmetric functions in $\beta_1, \beta_2, \dots, \beta_v$, which are known as *power-sum symmetric functions*. Now, we define the following polynomial:

$$\begin{aligned}\sigma(X) &\stackrel{\Delta}{=} (1 + \beta_1 X)(1 + \beta_2 X) \cdots (1 + \beta_v X) \\ &= \sigma_0 + \sigma_1 X + \sigma_2 X^2 + \cdots + \sigma_v X^v.\end{aligned}\tag{6.20}$$

The roots of $\sigma(X)$ are $\beta_1^{-1}, \beta_2^{-1}, \dots, \beta_v^{-1}$, which are the inverses of the error-location numbers. For this reason, $\sigma(X)$ is called the *error-location polynomial*. Note that $\sigma(X)$ is an unknown polynomial whose coefficients must be determined. The coefficients of $\sigma(X)$ and the error-location numbers are related by the following equations:

$$\begin{aligned}\sigma_0 &= 1 \\ \sigma_1 &= \beta_1 + \beta_2 + \cdots + \beta_v \\ \sigma_2 &= \beta_1\beta_2 + \beta_2\beta_3 + \cdots + \beta_{v-1}\beta_v \\ &\vdots \\ \sigma_v &= \beta_1\beta_2 \cdots \beta_v.\end{aligned}\tag{6.21}$$

The σ_i 's are known as *elementary symmetric functions* of β_i 's. From (6.19) and (6.21), we see that the σ_i 's are related to the syndrome components S_j 's. In fact, they are related to the syndrome components by the following *Newton's identities*:

$$\begin{aligned}S_1 + \sigma_1 &= 0 \\ S_2 + \sigma_1 S_1 + 2\sigma_2 &= 0 \\ S_3 + \sigma_1 S_2 + \sigma_2 S_1 + 3\sigma_3 &= 0 \\ &\vdots \\ S_v + \sigma_1 S_{v-1} + \cdots + \sigma_{v-1} S_1 + v\sigma_v &= 0 \\ S_{v+1} + \sigma_1 S_v + \cdots + \sigma_{v-1} S_2 + \sigma_v S_1 &= 0 \\ &\vdots\end{aligned}\tag{6.22}$$

For the binary case, since $1 + 1 = 2 = 0$, we have

$$i\sigma_i = \begin{cases} \sigma_i & \text{for odd } i, \\ 0 & \text{for even } i. \end{cases}$$

If it is possible to determine the elementary symmetric functions $\sigma_1, \sigma_2, \dots, \sigma_v$ from the equations of (6.22), the error-location numbers $\beta_1, \beta_2, \dots, \beta_v$ can be found by determining the roots of the error-location polynomial $\sigma(X)$. Again, the equations of (6.22) may have many solutions; however, we want to find the solution that yields a $\sigma(X)$ of minimal degree. This $\sigma(X)$ will produce an error pattern with a minimum number of errors. If $v \leq t$, this $\sigma(X)$ will give the actual error pattern $e(X)$. Next, we describe a procedure to determine the polynomial $\sigma(X)$ of minimum degree that satisfies the first $2t$ equations of (6.22) (since we know only S_1 through S_{2t}).

At this point, we outline the error-correcting procedure for BCH codes. The procedure consists of three major steps:

1. Compute the syndrome $\mathbf{S} = (S_1, S_2, \dots, S_{2t})$ from the received polynomial $\mathbf{r}(X)$.
2. Determine the error-location polynomial $\sigma(X)$ from the syndrome components S_1, S_2, \dots, S_{2t} .
3. Determine the error-location numbers $\beta_1, \beta_2, \dots, \beta_v$ by finding the roots of $\sigma(X)$, and correct the errors in $\mathbf{r}(X)$.

The first decoding algorithm that carries out these three steps was devised by Peterson [3]. Steps 1 and 3 are quite simple; step 2 is the most complicated part of decoding a BCH code.

6.3 ITERATIVE ALGORITHM FOR FINDING THE ERROR-LOCATION POLYNOMIAL $\sigma(X)$

Here we present Berlekamp's iterative algorithm for finding the error-location polynomial. We describe only the algorithm, without giving any proof. The reader who is interested in details of this algorithm is referred to Berlekamp [9], Peterson and Weldon [13], and MacWilliams and Sloane [14].

The first step of iteration is to find a minimum-degree polynomial $\sigma^{(1)}(X)$ whose coefficients satisfy the first Newton's identity of (6.22). The next step is to test whether the coefficients of $\sigma^{(1)}(X)$ also satisfy the second Newton's identity of (6.22). If the coefficients of $\sigma^{(1)}(X)$ do satisfy the second Newton's identity of (6.22), we set

$$\sigma^{(2)}(X) = \sigma^{(1)}(X).$$

If the coefficients of $\sigma^{(1)}(X)$ do not satisfy the second Newton's identity of (6.22), we add a correction term to $\sigma^{(1)}(X)$ to form $\sigma^{(2)}(X)$ such that $\sigma^{(2)}(X)$ has minimum degree and its coefficients satisfy the first two Newton's identities of (6.22). Therefore, at the end of the second step of iteration, we obtain a minimum-degree polynomial $\sigma^{(2)}(X)$ whose coefficients satisfy the first two Newton's identities of (6.22). The third step of iteration is to find a minimum-degree polynomial $\sigma^{(3)}(X)$ from $\sigma^{(2)}(X)$ such that the coefficients of $\sigma^{(3)}(X)$ satisfy the first three Newton's identities of (6.22). Again, we test whether the coefficients of $\sigma^{(2)}(X)$ satisfy the third Newton's identity of (6.22). If they do, we set $\sigma^{(3)}(X) = \sigma^{(2)}(X)$. If they do not, we add a correction term to $\sigma^{(2)}(X)$ to form $\sigma^{(3)}(X)$. Iteration continues until we obtain $\sigma^{(2t)}(X)$. Then, $\sigma^{(2t)}(X)$ is taken to be the error-location polynomial $\sigma(X)$, that is,

$$\sigma(X) = \sigma^{(2t)}(X).$$

This $\sigma(X)$ will yield an error pattern $\mathbf{e}(X)$ of minimum weight that satisfies the equations of (6.17). If the number of errors in the received polynomial $\mathbf{r}(X)$ is t or less, then $\sigma(X)$ produces the true error pattern.

Let

$$\sigma^{(\mu)}(X) = 1 + \sigma_1^{(\mu)} X + \sigma_2^{(\mu)} X^2 + \dots + \sigma_{l_\mu}^{(\mu)} X^{l_\mu} \quad (6.23)$$

be the minimum-degree polynomial determined at the μ th step of iteration whose coefficients satisfy the first μ Newton's identities of (6.22). To determine $\sigma^{(\mu+1)}(X)$,

we compute the following quantity:

$$d_\mu = S_{\mu+1} + \sigma_1^{(\mu)} S_\mu + \sigma_2^{(\mu)} S_{\mu-1} + \cdots + \sigma_{l_\mu}^{(\mu)} S_{\mu+1-l_\mu}. \quad (6.24)$$

This quantity d_μ is called the μ th *discrepancy*. If $d_\mu = 0$, the coefficients of $\sigma^{(\mu)}(X)$ satisfy the $(\mu + 1)$ th Newton's identity. In this event, we set

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X).$$

If $d_\mu \neq 0$, the coefficients of $\sigma^{(\mu)}(X)$ do not satisfy the $(\mu + 1)$ th Newton's identity, and we must add a correction term to $\sigma^{(\mu)}(X)$ to obtain $\sigma^{(\mu+1)}(X)$. To make this correction, we go back to the steps *prior to* the μ th step and determine a polynomial $\sigma^{(\rho)}(X)$ such that the ρ th discrepancy $d_\rho \neq 0$, and $\rho - l_\rho$ [l_ρ is the degree of $\sigma^{(\rho)}(X)$] has the largest value. Then,

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X) + d_\mu d_\rho^{-1} X^{(\mu-\rho)} \sigma^{(\rho)}(X), \quad (6.25)$$

which is the minimum-degree polynomial whose coefficients satisfy the first $\mu + 1$ Newton's identities. The proof of this is quite complicated and is omitted from this introductory book.

To carry out the iteration of finding $\sigma(X)$, we begin with Table 6.5 and proceed to fill out the table, where l_μ is the degree of $\sigma^{(\mu)}(X)$. Assuming that we have filled out all rows up to and including the μ th row, we fill out the $(\mu + 1)$ th row as follows:

1. If $d_\mu = 0$, then $\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X)$, and $l_{\mu+1} = l_\mu$.
2. If $d_\mu \neq 0$, we find another row ρ prior to the μ th row such that $d_\rho \neq 0$ and the number $\rho - l_\rho$ in the last column of the Table has the largest value. Then, $\sigma^{(\mu+1)}(X)$ is given by (6.25), and

$$l_{\mu+1} = \max(l_\mu, l_\rho + \mu - \rho). \quad (6.26)$$

In either case,

$$d_{\mu+1} = S_{\mu+2} + \sigma_1^{(\mu+1)} S_{\mu+1} + \cdots + \sigma_{l_{\mu+1}}^{(\mu+1)} S_{\mu+2-l_{\mu+1}}, \quad (6.27)$$

where the $\sigma_i^{(\mu+1)}$'s are the coefficients of $\sigma^{(\mu+1)}(X)$. The polynomial $\sigma^{(2t)}(X)$ in the last row should be the required $\sigma(X)$. If its degree is greater than t , there are more

TABLE 6.5: Berlekamp's iterative procedure for finding the error-location polynomial of a BCH code.

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	S_1	0	0
1				
2				
\vdots				
$2t$				

than t errors in the received polynomial $\mathbf{r}(X)$, and generally it is not possible to locate them.

EXAMPLE 6.5

Consider the (15, 5) triple-error-correcting BCH code given in Example 6.1. Assume that the codeword of all zeros,

$$\mathbf{v} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),$$

is transmitted, and the vector

$$\mathbf{r} = (0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0)$$

is received. Then, $\mathbf{r}(X) = X^3 + X^5 + X^{12}$. The minimal polynomials for α , α^2 , and α^4 are identical, and

$$\phi_1(X) = \phi_2(X) = \phi_4(X) = 1 + X + X^4.$$

The elements α^3 and α^6 have the same minimal polynomial,

$$\phi_3(X) = \phi_6(X) = 1 + X + X^2 + X^3 + X^4.$$

The minimal polynomial for α^5 is

$$\phi_5(X) = 1 + X + X^2.$$

Dividing $\mathbf{r}(X)$ by $\phi_1(X)$, $\phi_3(X)$, and $\phi_5(X)$, respectively, we obtain the following remainders:

$$\begin{aligned}\mathbf{b}_1(X) &= 1, \\ \mathbf{b}_3(X) &= 1 + X^2 + X^3, \\ \mathbf{b}_5(X) &= X^2.\end{aligned}\tag{6.28}$$

Using Table 2.8 and substituting α , α^2 , and α^4 into $\mathbf{b}_1(X)$, we obtain the following syndrome components:

$$S_1 = S_2 = S_4 = 1.$$

Substituting α^3 and α^6 into $\mathbf{b}_3(X)$, we obtain

$$\begin{aligned}S_3 &= 1 + \alpha^6 + \alpha^9 = \alpha^{10}, \\ S_6 &= 1 + \alpha^{12} + \alpha^{18} = \alpha^5.\end{aligned}\tag{6.29}$$

Substituting α^5 into $\mathbf{b}_5(X)$, we have

$$S_5 = \alpha^{10}.$$

Using the iterative procedure described previously, we obtain Table 6.6. Thus, the error-location polynomial is

$$\sigma(X) = \sigma^{(6)}(X) = 1 + X + \alpha^5 X^3.$$

TABLE 6.6: Steps for finding the error-location polynomial of the (15,5) BCH code given in Example 6.5.

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	1	0	0
1	$1 + X$	0	1	0 (take $\rho = -1$)
2	$1 + X$	α^5	1	1
3	$1 + X + \alpha^5 X^2$	0	2	1 (take $\rho = 0$)
4	$1 + X + \alpha^5 X^2$	α^{10}	2	2
5	$1 + X + \alpha^5 X^3$	0	3	2 (take $\rho = 2$)
6	$1 + X + \alpha^5 X^3$	—	—	—

We can easily check that α^3 , α^{10} , and α^{12} are the roots of $\sigma(X)$. Their inverses are α^{12} , α^5 , and α^3 , which are the error-location numbers. Therefore, the error pattern is

$$\mathbf{e}(X) = X^3 + X^5 + X^{12}.$$

Adding $\mathbf{e}(X)$ to the received polynomial $\mathbf{r}(X)$, we obtain the all-zero vector.

If the number of errors in the received polynomial $\mathbf{r}(X)$ is less than the designed error-correcting capability t of the code, it is not necessary to carry out the $2t$ steps of iteration to find the error-location polynomial $\sigma(X)$. Let $\sigma^{(\mu)}(X)$ and d_μ be the solution and discrepancy obtained at the μ th step of iteration. Let l_μ be the degree of $\sigma^{(\mu)}(X)$. Chen [19] has shown that if d_μ and the discrepancies at the next $t - l_\mu - 1$ steps are all zero, $\sigma^{(\mu)}(X)$ is the error-location polynomial. Therefore, if the number of errors in the received polynomial $\mathbf{r}(X)$ is v ($v \leq t$), only $t + v$ steps of iteration are needed to determine the error-location polynomial $\sigma(X)$. If v is small (this is often the case), the reduction in the number of iteration steps results in an increase in decoding speed.

The described iterative algorithm for finding $\sigma(X)$ applies not only to binary BCH codes but also to nonbinary BCH codes.

6.4 SIMPLIFIED ITERATIVE ALGORITHM FOR FINDING THE ERROR-LOCATION POLYNOMIAL $\sigma(X)$

The described iterative algorithm for finding $\sigma(X)$ applies to both binary and nonbinary BCH codes, including Reed–Solomon codes; however, for binary BCH codes, this algorithm can be simplified to t -steps for computing $\sigma(X)$.

Recall that for a polynomial $f(X)$ over $GF(2)$,

$$f^2(X) = f(X^2),$$

[see (2.10)]. Because the received polynomial $\mathbf{r}(X)$ is a polynomial over $GF(2)$, we have

$$\mathbf{r}^2(X) = \mathbf{r}(X^2).$$

Substituting α^i for X in the preceding equality, we obtain

$$\mathbf{r}^2(\alpha^i) = \mathbf{r}(\alpha^{2i}). \quad (6.30)$$

Because $S_i = \mathbf{r}(\alpha^i)$, and $S_{2i} = \mathbf{r}(\alpha^{2i})$ [see (6.13)], we obtain the following relationship between S_{2i} and S_i :

$$S_{2i} = S_i^2. \quad (6.31)$$

Suppose the first Newton's identity of (6.22) holds. Then,

$$S_1 + \sigma_1 = 0.$$

This result says that

$$\sigma_1 = S_1. \quad (6.32)$$

It follows from (6.31) and (6.32) that

$$\begin{aligned} S_2 + \sigma_1 S_1 + 2\sigma_2 &= S_1^2 + S_1 \cdot S_1 + 0 \\ &= 0. \end{aligned} \quad (6.33)$$

The foregoing equality is simply the Newton's second equality. This result says that if the first Newton's identity holds, then the second Newton's identity also holds. Now, suppose the first and the third Newton's identities of (6.22) hold; that is,

$$S_1 + \sigma_1 = 0, \quad (6.34)$$

$$S_3 + \sigma_1 S_2 + \sigma_2 S_1 + 3\sigma_3 = 0. \quad (6.35)$$

The equality of (6.34) implies that the second Newton's identity holds:

$$S_2 + \sigma_1 S_1 + 2\sigma_2 = 0. \quad (6.36)$$

Then,

$$(S_2 + \sigma_1 S_1 + 2\sigma_2)^2 = 0,$$

$$S_2^2 + \sigma_1^2 S_1^2 + 4\sigma_1 \sigma_2 S_1 + 4\sigma_2^2 = 0. \quad (6.37)$$

It follows from (6.31) that (6.37) becomes

$$S_4 + \sigma_1^2 S_2 = 0. \quad (6.38)$$

Multiplying both sides of (6.35) by σ_1 , we obtain

$$\sigma_1 S_3 + \sigma_1^2 S_2 + \sigma_1 \sigma_2 S_1 + 3\sigma_1 \sigma_3 = 0. \quad (6.39)$$

Adding (6.38) and (6.39) and using the equalities of (6.31) and (6.32), we find that the fourth Newton's identity holds:

$$S_4 + \sigma_1 S_3 + \sigma_2 S_2 + \sigma_3 S_1 + 4\sigma_4 = 0.$$

This result says that if the first and the third Newton's identities hold, the second and the fourth Newton's identities also hold.

With some effort, it is possible to prove that if the first, third, \dots , $(2t - 1)$ th Newton's identities hold, then the second, fourth, \dots , $2t$ th Newton's identities also hold. This implies that with the iterative algorithm for finding the error-location polynomial $\sigma(X)$, the solution $\sigma^{(2\mu-1)}(X)$ at the $(2\mu - 1)$ th step of iteration is also the solution $\sigma^{(2\mu)}(X)$ at the 2μ th step of iteration; that is,

$$\sigma^{(2\mu)}(X) = \sigma^{(2\mu-1)}(X). \quad (6.40)$$

This suggests that the $(2\mu - 1)$ th and the 2μ th steps of iteration can be combined. As a result, the foregoing iterative algorithm for finding $\sigma(X)$ can be reduced to t steps. Only the even steps are needed.

The simplified algorithm can be carried out by filling out a table with only t rows, as illustrated in Table 6.7.

Assuming that we have filled out all rows up to and including the μ th row, we fill out the $(\mu + 1)$ th row as follows:

1. If $d_\mu = 0$, then $\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X)$.
2. If $d_\mu \neq 0$, we find another row preceding the μ th row, say the ρ th, such that the number $2\rho - l_\rho$ in the last column is as large as possible and $d_\rho \neq 0$. Then,

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X) + d_\mu d_\rho^{-1} X^{2(\mu-\rho)} \sigma^{(\rho)}(X). \quad (6.41)$$

In either case, $l_{\mu+1}$ is exactly the degree of $\sigma^{(\mu+1)}(X)$, and the discrepancy at the $(\mu + 1)$ th step is

$$d_{\mu+1} = S_{2\mu+3} + \sigma_1^{(\mu+1)} S_{2\mu+2} + \sigma_2^{(\mu+1)} S_{2\mu+1} + \dots + \sigma_{l_{\mu+1}}^{(\mu+1)} S_{2\mu+3-l_{\mu+1}}. \quad (6.42)$$

The polynomial $\sigma^{(\mu+1)}(X)$ in the last row should be the required $\sigma(X)$. If its degree is greater than t , there were more than t errors, and generally it is not possible to locate them.

The computation required in this simplified algorithm is half of the computation required in the general algorithm; however, we must remember that the simplified algorithm applies only to binary BCH codes. Again, if the number of errors in the

TABLE 6.7: A simplified Berlekamp iterative procedure for finding the error-location polynomial of a binary BCH code.

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$2\mu - l_\mu$
$-\frac{1}{2}$	1	1	0	-1
0	1	S_1	0	0
1				
2				
\vdots				
t				

TABLE 6.8: Steps for finding the error-location polynomial of the (15,5) binary BCH code given in Example 6.6.

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$2\mu - l_\mu$
$-\frac{1}{2}$	1	1	0	-1
0	1	$S_1 = 1$	0	0
1	$1 + S_1 X = 1 + X$	$S_3 + S_2 S_1 = \alpha^5$	1	1 (take $\rho = -\frac{1}{2}$)
2	$1 + X + \alpha^5 X^2$	α^{10}	2	2 (take $\rho = 0$)
3	$1 + X + \alpha^5 X^3$	—	3	3 (take $\rho = 2$)

received polynomial $\mathbf{r}(X)$ is less than t , it is not necessary to carry out the t steps of iteration to determine $\sigma(X)$ for a t -error-correcting binary BCH code. Based on Chen's result [19], if for some μ , d_μ and the discrepancies at the next $\lceil(t - l_\mu - 1)/2\rceil$ steps of iteration are zero, $\sigma^{(\mu)}(X)$ is the error-location polynomial. If the number of errors in the received polynomial is v ($v \leq t$), only $\lceil(t + v)/2\rceil$ steps of iteration are needed to determine the error-location polynomial $\sigma(X)$.

EXAMPLE 6.6

The simplified table for finding $\sigma(X)$ for the code considered in Example 6.5 is given in Table 6.8. Thus, $\sigma(X) = \sigma^{(3)}(X) = 1 + X + \alpha^5 X^3$, which is identical to the solution found in Example 6.5.

6.5 FINDING THE ERROR-LOCATION NUMBERS AND ERROR CORRECTION

The last step in decoding a BCH code is to find the error-location numbers that are the reciprocals of the roots of $\sigma(X)$. The roots of $\sigma(X)$ can be found simply by substituting $1, \alpha, \alpha^2, \dots, \alpha^{n-1}$ ($n = 2^m - 1$) into $\sigma(X)$. Since $\alpha^n = 1, \alpha^{-l} = \alpha^{n-l}$. Therefore, if α^l is a root of $\sigma(X)$, α^{n-l} is an error-location number, and the received digit r_{n-l} is an erroneous digit. Consider Example 6.6, where the error-location polynomial was found to be

$$\sigma(X) = 1 + X + \alpha^5 X^3.$$

By substituting $1, \alpha, \alpha^2, \dots, \alpha^{14}$ into $\sigma(X)$, we find that α^3, α^{10} , and α^{12} are roots of $\sigma(X)$. Therefore, the error-location numbers are α^{12}, α^5 , and α^3 . The error pattern is

$$\mathbf{e}(X) = X^3 + X^5 + X^{12},$$

which is exactly the assumed error pattern. The decoding of the code is completed by adding (modulo-2) $\mathbf{e}(X)$ to the received vector $\mathbf{r}(X)$.

The described substitution method for finding the roots of the error-location polynomial was first used by Peterson in his algorithm for decoding BCH codes [3]. Later, Chien [6] formulated a procedure for carrying out the substitution and error correction. Chien's procedure for searching for error-location numbers is described next. The received vector

$$\mathbf{r}(X) = r_0 + r_1 X + r_2 X^2 + \dots + r_{n-1} X^{n-1}$$

is decoded bit by bit. The high-order bits are decoded first. To decode r_{n-1} , the decoder tests whether α^{n-1} is an error-location number; this is equivalent to testing whether its inverse, α , is a root of $\sigma(X)$. If α is a root, then

$$1 + \sigma_1\alpha + \sigma_2\alpha^2 + \cdots + \sigma_v\alpha^v = 0.$$

Therefore, to decode r_{n-1} , the decoder forms $\sigma_1\alpha, \sigma_2\alpha^2, \dots, \sigma_v\alpha^v$. If the sum $1 + \sigma_1\alpha + \sigma_2\alpha^2 + \cdots + \sigma_v\alpha^v = 0$, then α^{n-1} is an error-location number, and r_{n-1} is an erroneous digit; otherwise, r_{n-1} is a correct digit. To decode r_{n-l} , the decoder forms $\sigma_1\alpha^l, \sigma_2\alpha^{2l}, \dots, \sigma_v\alpha^{vl}$ and tests the sum

$$1 + \sigma_1\alpha^l + \sigma_2\alpha^{2l} + \cdots + \sigma_v\alpha^{vl}.$$

If this sum is 0, then α^l is a root of $\sigma(X)$, and r_{n-l} is an erroneous digit; otherwise, r_{n-l} is a correct digit.

The described testing procedure for error locations can be implemented in a straightforward manner by a circuit such as that shown in Figure 6.1 [6]. The t -registers are initially stored with $\sigma_1, \sigma_2, \dots, \sigma_t$, calculated in step 2 of the decoding ($\sigma_{v+1} = \sigma_{v+2} = \cdots = \sigma_t = 0$ for $v < t$). Immediately before r_{n-1} is read out of the buffer, the t multipliers \otimes are pulsed once. The multiplications are performed, and $\sigma_1\alpha, \sigma_2\alpha^2, \dots, \sigma_v\alpha^v$ are stored in the σ -registers. The output of the logic circuit A is 1 if and only if the sum $1 + \sigma_1\alpha + \sigma_2\alpha^2 + \cdots + \sigma_v\alpha^v = 0$; otherwise, the output of A is 0. The digit r_{n-1} is read out of the buffer and corrected by the output of A . Once r_{n-1} is decoded, the t multipliers are pulsed again. Now, $\sigma_1\alpha^2, \sigma_2\alpha^4, \dots, \sigma_v\alpha^{2v}$ are stored in the σ -registers. The sum

$$1 + \sigma_1\alpha^2 + \sigma_2\alpha^4 + \cdots + \sigma_v\alpha^{2v}$$

is tested for 0. The digit r_{n-2} is read out of the buffer and corrected in the same manner as r_{n-1} was corrected. This process continues until the whole received vector is read out of the buffer.

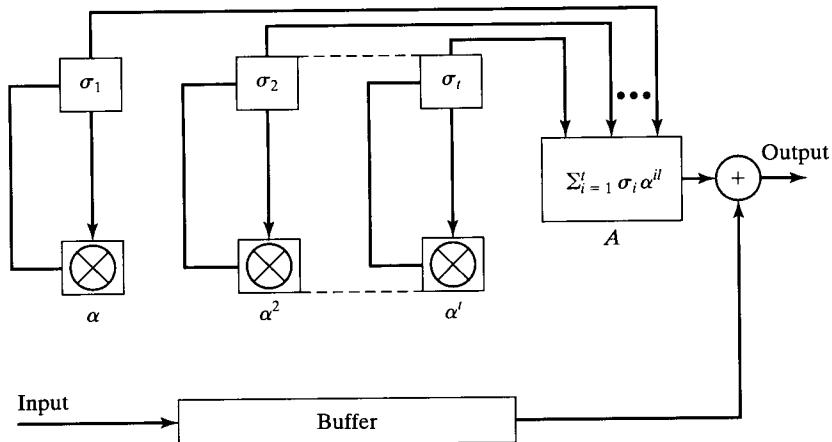


FIGURE 6.1: Cyclic error location search unit.

The described decoding algorithm also applies to nonprimitive BCH codes. The $2t$ syndrome components are given by

$$S_i = \mathbf{r}(\beta^i)$$

for $1 \leq i \leq 2t$.

6.6 CORRECTION OF ERRORS AND ERASURES

If the channel is the binary symmetric erasure channel as shown in Figure 1.6(b), the received vector may contain both errors and erasures. It was shown in Section 3.4, that a code with minimum distance d_{\min} is capable of correcting all combinations of v errors and e erasures provided that

$$2v + e + 1 \leq d_{\min}. \quad (6.43)$$

Erasure and error correction with binary BCH codes are quite simple. Suppose a BCH code is designed to correct t errors, and the received polynomial $\mathbf{r}(X)$ contains v (unknown) random errors and e (known) erasures. The decoding can be accomplished in two steps. First, the erased positions are replaced with 0's and the resulting vector is decoded using the standard BCH decoding algorithm. Next, the e erased positions are replaced with 1's, and the resulting vector is decoded in the same manner. The decodings result in two codewords. The codeword with the smallest number of errors corrected outside the e erased positions is chosen as the decoded codeword. If the inequality of (6.43) holds, this decoding algorithm always results in correct decoding. To see this, we write (6.43) in terms of the error-correcting capability t of the code as

$$v + e/2 \leq t. \quad (6.44)$$

Assume that when e erasures are replaced with 0's, $e^* \leq e/2$ errors are introduced in those e erased positions. As a result, the resulting vector contains a total of $v + e^* \leq t$ errors that are guaranteed to be correctable if the inequality of (6.44) (or (6.43)) holds; however, if $e^* > e/2$ then only $e - e^* < e/2$ errors are introduced when the e erasures are replaced with 1's. In this case the resultant vector contains $v + (e - e^*) < t$ errors. Such errors are also correctable. Therefore, with the described decoding algorithm, at least one of the two decoded codewords is correct.

6.7 IMPLEMENTATION OF GALOIS FIELD ARITHMETIC

Decoding of BCH codes requires computations using Galois field arithmetic. Galois field arithmetic can be implemented more easily than ordinary arithmetic because there are no carries. In this Section we discuss circuits that perform addition and multiplication over a Galois field. For simplicity, we consider the arithmetic over the Galois field $GF(2^4)$ given by Table 2.8.

To add two field elements, we simply add their vector representations. The resultant vector is then the vector representation of the sum of the two field elements. For example, we want to add α^7 and α^{13} of $GF(2^4)$. From Table 2.8 we find that their vector representations are $(1 \ 1 \ 0 \ 1)$ and $(1 \ 0 \ 1 \ 1)$, respectively. Their vector sum is $(1 \ 1 \ 0 \ 1) + (1 \ 0 \ 1 \ 1) = (0 \ 1 \ 1 \ 0)$, which is the vector representation of α^5 .

ADD

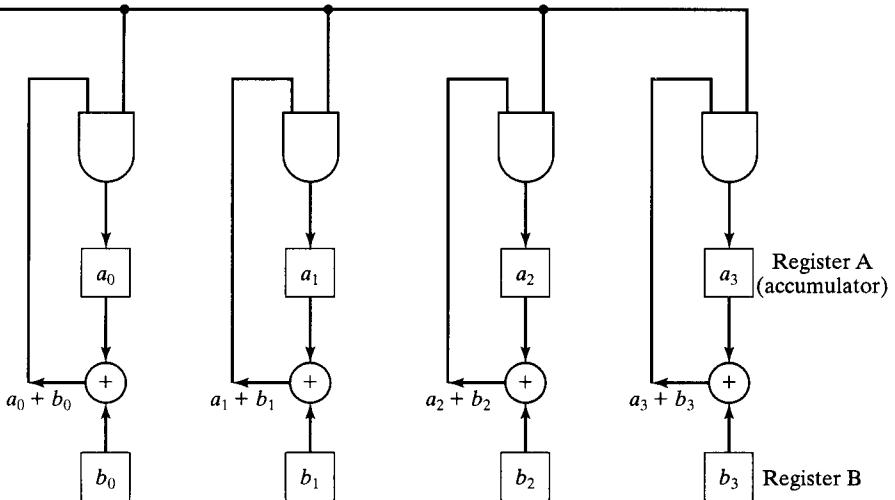


FIGURE 6.2: Galois field adder.

Two field elements can be added with the circuit shown in Figure 6.2. First, the vector representations of the two elements to be added are loaded into registers A and B. Their vector sum then appears at the inputs of register A. When register A is pulsed (or clocked), the sum is loaded into register A (register A serves as an accumulator).

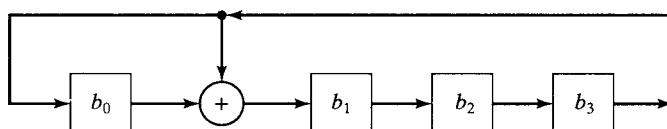
For multiplication, we first consider multiplying a field element by a fixed element from the same field. Suppose that we want to multiply a field element β in $GF(2^4)$ by the primitive element α whose minimal polynomial is $\phi(X) = 1 + X + X^4$. We can express element β as a polynomial in α as follows:

$$\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3.$$

Multiplying both sides of this equality by α and using the fact that $\alpha^4 = 1 + \alpha$, we obtain the following equality:

$$\alpha\beta = b_3 + (b_0 + b_3)\alpha + b_1\alpha^2 + b_2\alpha^3.$$

This multiplication can be carried out by the feedback shift register shown in Figure 6.3. First, the vector representation (b_0, b_1, b_2, b_3) of β is loaded into the register, then the register is pulsed. The new contents in the register form the

FIGURE 6.3: Circuit for multiplying an arbitrary element in $GF(2^4)$ by α .

vector representation of $\alpha\beta$. For example, let $\beta = \alpha^7 = 1 + \alpha + \alpha^3$. The vector representation of β is $(1\ 1\ 0\ 1)$. We load this vector into the register of the circuit shown in Figure 6.3. After the register is pulsed, the new contents in the register will be $(1\ 0\ 1\ 0)$, which represents α^8 , the product of α^7 and α . The circuit shown in Figure 6.3 can be used to generate (or count) all the nonzero elements of $GF(2^4)$. First, we load $(1\ 0\ 0\ 0)$ (vector representation of $\alpha^0 = 1$) into the register. Successive shifts of the register will generate vector representations of successive powers of α , in exactly the same order as they appear in Table 2.8. At the end of the fifteenth shift, the register will contain $(1\ 0\ 0\ 0)$ again.

As another example, suppose that we want to devise a circuit to multiply an arbitrary element β of $GF(2^4)$ by the element α^3 . Again, we express β in polynomial form:

$$\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3.$$

Multiplying both sides of the preceding equation by α^3 , we have

$$\begin{aligned}\alpha^3\beta &= b_0\alpha^3 + b_1\alpha^4 + b_2\alpha^5 + b_3\alpha^6 \\ &= b_0\alpha^3 + b_1(1 + \alpha) + b_2(\alpha + \alpha^2) + b_3(\alpha^2 + \alpha^3) \\ &= b_1 + (b_1 + b_2)\alpha + (b_2 + b_3)\alpha^2 + (b_0 + b_3)\alpha^3.\end{aligned}$$

Based on the preceding expression, we obtain the circuit shown in Figure 6.4, which is capable of multiplying any element β in $GF(2^4)$ by α^3 . To multiply, we first load the vector representation (b_0, b_1, b_2, b_3) of β into the register, then we pulse the register. The new contents in the register will be the vector representation of $\alpha^3\beta$. Next, we consider multiplying two arbitrary field elements. Again, we use $GF(2^4)$ for illustration. Let β and γ be two elements in $GF(2^4)$. We express these two elements in polynomial form:

$$\begin{aligned}\beta &= b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3, \\ \gamma &= c_0 + c_1\alpha + c_2\alpha^2 + c_3\alpha^3.\end{aligned}$$

Then, we can express the product $\beta\gamma$ in the following form:

$$\beta\gamma = (((c_3\beta)\alpha + c_2\beta)\alpha + c_1\beta)\alpha + c_0\beta \quad (6.45)$$

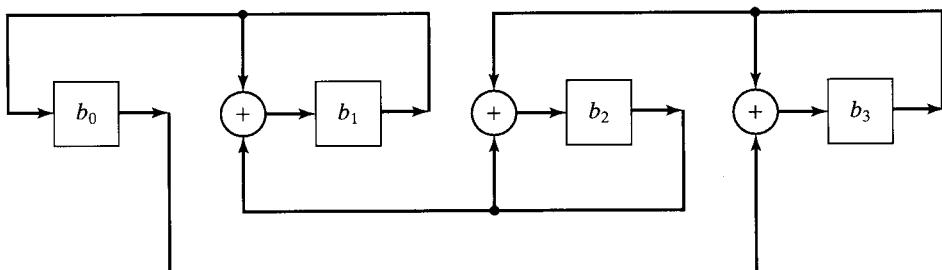


FIGURE 6.4: Circuit for multiplying an arbitrary element in $GF(2^4)$ by α^3 .

This product can be evaluated with the following steps:

1. Multiply $c_3\beta$ by α and add the product to $c_2\beta$.
2. Multiply $(c_3\beta)\alpha + c_2\beta$ by α and add the product to $c_1\beta$.
3. Multiply $((c_3\beta)\alpha + c_2\beta)\alpha + c_1\beta$ by α and add the product to $c_0\beta$.

Multiplication by α can be carried out by the circuit shown in Figure 6.3. This circuit can be modified to carry out the computation given by (6.45). The resultant circuit is shown in Figure 6.5. In operation of this circuit, the feedback shift register A is initially empty, and (b_0, b_1, b_2, b_3) and (c_0, c_1, c_2, c_3) , the vector representations of β and γ , are loaded into registers B and C, respectively. Then, registers A and C are shifted four times. At the end of the first shift, register A contains $(c_3b_0, c_3b_1, c_3b_2, c_3b_3)$, the vector representation of $c_3\beta$. At the end of the second shift, register A contains the vector representation of $(c_3\beta)\alpha + c_2\beta$. At the end of the third shift, register A contains the vector representation of $((c_3\beta)\alpha + c_2\beta)\alpha + c_1\beta$. At the end of the fourth shift, register A contains the product $\beta\gamma$ in vector form. If we express the product $\beta\gamma$ in the form

$$\beta\gamma = (((c_0\beta) + c_1\beta\alpha) + c_2\beta\alpha^2) + c_3\beta\alpha^3,$$

we obtain a different multiplication circuit, as shown in Figure 6.6. To perform the multiplication, β and γ are loaded into registers B and C, respectively, and register A is initially empty. Then, registers A, B, and C are shifted four times. At the end of

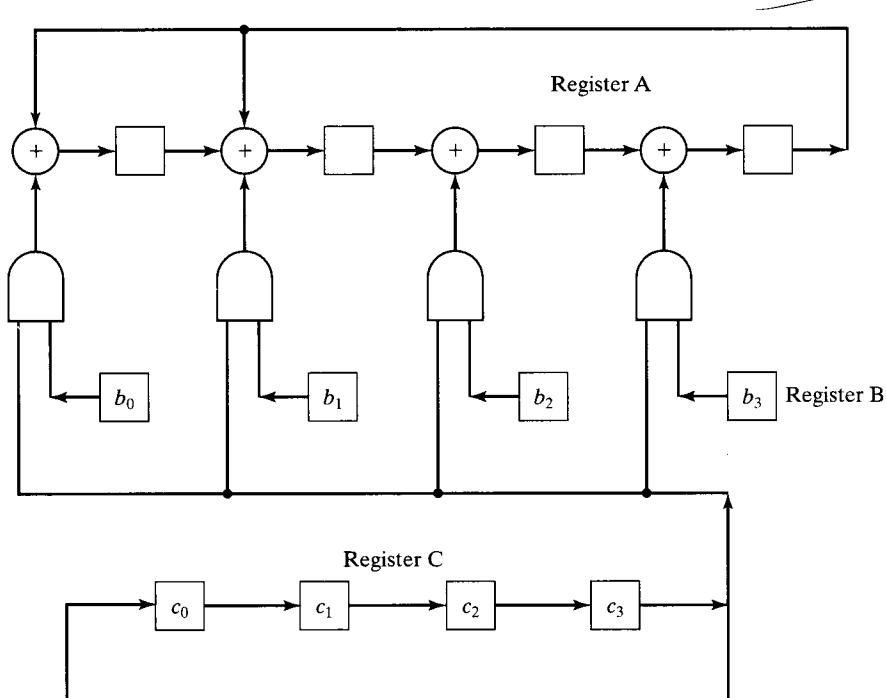


FIGURE 6.5: Circuit for multiplying two elements of $GF(2^4)$.

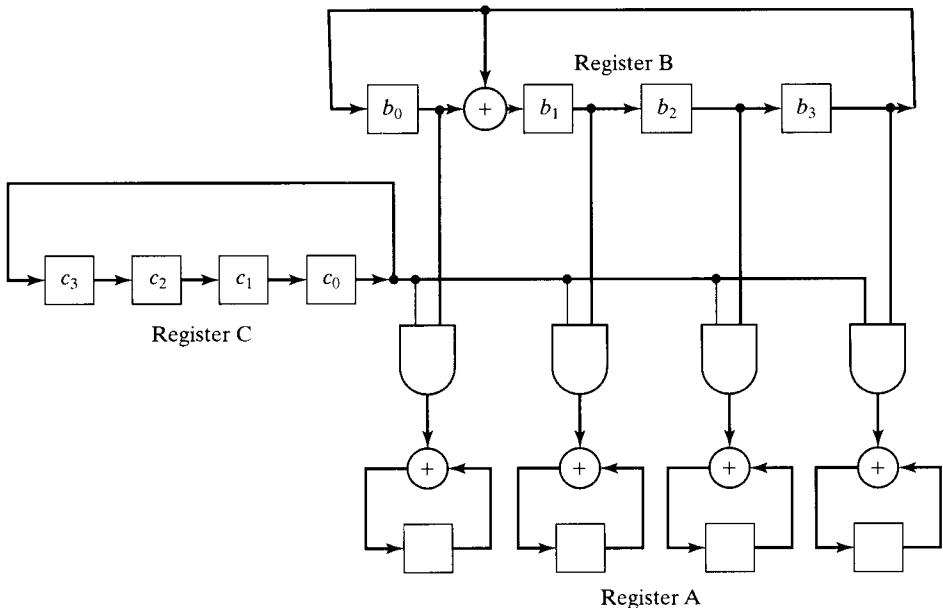


FIGURE 6.6: Another circuit for multiplying two elements of $GF(2^4)$.

the fourth shift, register A holds the product $\beta\gamma$. Both multiplication circuits shown in Figures 6.5 and 6.6 are of the same complexity and require the same amount of computation time.

Two elements from $GF(2^m)$ can be multiplied with a combinational logic circuit with $2m$ inputs and m outputs. The advantage of this implementation is its speed; however, for large m , it becomes prohibitively complex and costly. Multiplication can also be programmed in a general-purpose computer; it requires roughly $5m$ instruction executions.

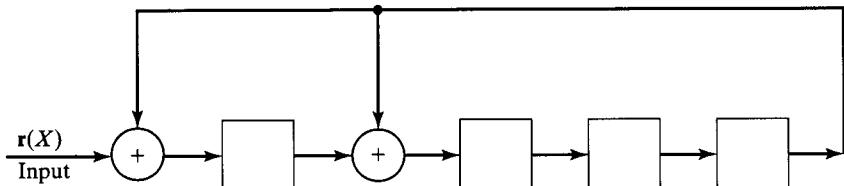
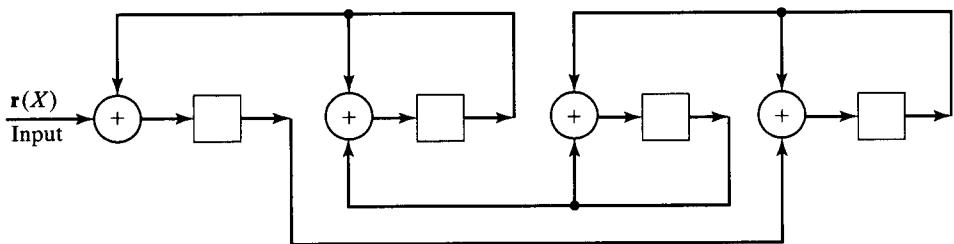
Let $\mathbf{r}(X)$ be a polynomial over $GF(2)$. We consider now how to compute $\mathbf{r}(\alpha^i)$. This type of computation is required in the first step of decoding of a BCH code. It can be done with a circuit for multiplying a field element by α^i in $GF(2^m)$. Again, we use computation over $GF(2^4)$ for illustration. Suppose that we want to compute

$$\mathbf{r}(\alpha) = r_0 + r_1\alpha + r_2\alpha^2 + \cdots + r_{14}\alpha^{14}, \quad (6.46)$$

where α is a primitive element in $GF(2^4)$ given by Table 2.8. We can express the right-hand side of (6.46) in the form

$$\mathbf{r}(\alpha) = (\cdots (((r_{14})\alpha + r_{13})\alpha + r_{12})\alpha + \cdots)\alpha + r_0.$$

Then, we can compute $\mathbf{r}(\alpha)$ by adding an input to the circuit for multiplying by α shown in Figure 6.3. The resultant circuit for computing $\mathbf{r}(\alpha)$ is shown in Figure 6.7. In operation of this circuit, the register is initially empty. The vector $(r_0, r_1, \dots, r_{14})$ is shifted into the circuit one digit at a time. After the first shift, the register contains $(r_{14}, 0, 0, 0)$. At the end of the second shift, the register contains the vector representation of $r_{14}\alpha + r_{13}$. At the completion of the third shift, the

FIGURE 6.7: Circuit for computing $r(\alpha)$.FIGURE 6.8: Circuit for computing $r(\alpha^3)$.

register contains the vector representation of $(r_{14}\alpha + r_{13})\alpha + r_{12}$. When the last digit r_0 is shifted into the circuit, the register contains $\mathbf{r}(\alpha)$ in vector form.

Similarly, we can compute $\mathbf{r}(\alpha^3)$ by adding an input to the circuit for multiplying by α^3 of Figure 6.4. The resultant circuit for computing $\mathbf{r}(\alpha^3)$ is shown in Figure 6.8.

There is another way of computing $\mathbf{r}(\alpha^i)$. Let $\phi_i(X)$ be the minimal polynomial of α^i . Let $\mathbf{b}(X)$ be the remainder resulting from dividing $\mathbf{r}(X)$ by $\phi_i(X)$. Then,

$$\mathbf{r}(\alpha^i) = \mathbf{b}(\alpha^i).$$

Thus, computing $\mathbf{r}(\alpha^i)$ is equivalent to computing $\mathbf{b}(\alpha^i)$. A circuit can be devised to compute $\mathbf{b}(\alpha^i)$. For illustration, we again consider computation over $GF(2^4)$. Suppose that we want to compute $\mathbf{r}(\alpha^3)$. The minimal polynomial of α^3 is $\phi_3(X) = 1 + X + X^2 + X^3 + X^4$. The remainder resulting from dividing $\mathbf{r}(X)$ by $\phi_3(X)$ has the form

$$\mathbf{b}(X) = b_0 + b_1X + b_2X^2 + b_3X^3.$$

Then,

$$\begin{aligned}
 \mathbf{r}(\alpha^3) &= \mathbf{b}(\alpha^3) \\
 &= b_0 + b_1\alpha^3 + b_2\alpha^6 + b_3\alpha^9 \\
 &= b_0 + b_1\alpha^3 + b_2(\alpha^2 + \alpha^3) + b_3(\alpha + \alpha^3) \\
 &= b_0 + b_3\alpha + b_2\alpha^2 + (b_1 + b_2 + b_3)\alpha^3.
 \end{aligned} \tag{6.47}$$

From the preceding expression we see that $\mathbf{r}(\alpha^3)$ can be computed by using a circuit that divides $\mathbf{r}(X)$ by $\phi_3(X) = 1 + X + X^2 + X^3 + X^4$ and then combining

the coefficients of the remainder $\mathbf{b}(X)$ as given by (6.47). Such a circuit is shown in Figure 6.9, where the feedback connection of the shift register is based on $\phi_3(X) = 1 + X + X^2 + X^3 + X^4$. Because α^6 is a conjugate of α^3 , it has the same minimal polynomial as α^3 , and therefore $\mathbf{r}(\alpha^6)$ can be computed from the same remainder $\mathbf{b}(X)$ resulting from dividing $\mathbf{r}(X)$ by $\phi_3(X)$. To form $\mathbf{r}(\alpha^6)$, we combine

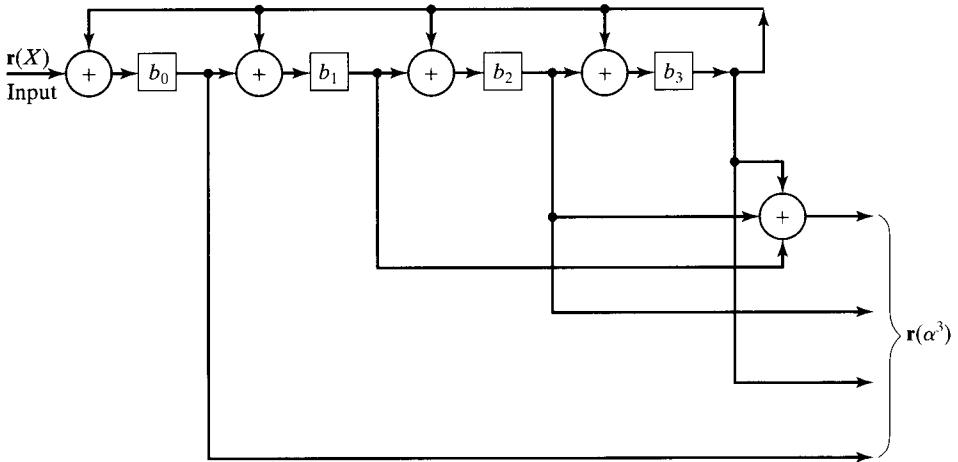


FIGURE 6.9: Another circuit for computing $\mathbf{r}(\alpha^3)$ in $GF(2^4)$.

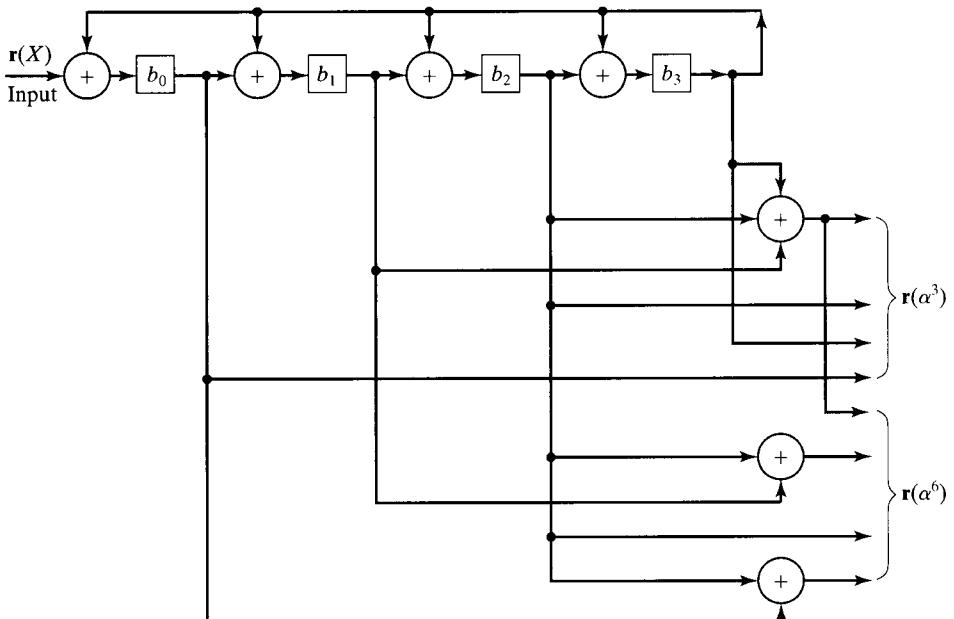


FIGURE 6.10: Circuit for computing $\mathbf{r}(\alpha^3)$ and $\mathbf{r}(\alpha^6)$ in $GF(2^4)$.

the coefficients of $\mathbf{b}(X)$ in the following manner:

$$\begin{aligned}\mathbf{r}(\alpha^6) &= \mathbf{b}(\alpha^6) \\ &= b_0 + b_1\alpha^6 + b_2\alpha^{12} + b_3\alpha^{18} \\ &= b_0 + b_1(\alpha^2 + \alpha^3) + b_2(1 + \alpha + \alpha^2 + \alpha^3) + b_3\alpha^3 \\ &= (b_0 + b_2) + b_2\alpha + (b_1 + b_2)\alpha^2 + (b_1 + b_2 + b_3)\alpha^3.\end{aligned}$$

The combined circuit for computing $\mathbf{r}(\alpha^3)$ and $\mathbf{r}(\alpha^6)$ is shown in Figure 6.10.

The arithmetic operation of division over $GF(2^m)$ can be performed by first forming the multiplicative inverse of the divisor β and then multiplying this inverse β^{-1} by the dividend, thus forming the quotient. The multiplicative inverse of β can be found by using the fact that $\beta^{2^m-1} = 1$. Thus,

$$\beta^{-1} = \beta^{2^m-2}.$$

6.8 IMPLEMENTATION OF ERROR CORRECTION

Each step in the decoding of a BCH code can be implemented either by digital hardware or by software. Each implementation has certain advantages. We consider these implementations next.

6.8.1 Syndrome Computations

The first step in decoding a t -error-correction BCH code is to compute the $2t$ syndrome components S_1, S_2, \dots, S_{2t} . These syndrome components may be obtained by substituting the field elements $\alpha, \alpha^2, \dots, \alpha^{2t}$ into the received polynomial $\mathbf{r}(X)$. For software implementation, α^i into $\mathbf{r}(X)$ is best substituted as follows:

$$\begin{aligned}S_i &= \mathbf{r}(\alpha^i) = r_{n-1}(\alpha^i)^{n-1} + r_{n-2}(\alpha^i)^{n-2} + \dots + r_1\alpha^i + r_0 \\ &= (\dots((r_{n-1}\alpha^i + r_{n-2})\alpha^i + r_{n-3})\alpha^i + \dots + r_1)\alpha^i + r_0.\end{aligned}$$

This computation takes $n - 1$ additions and $n - 1$ multiplications. For binary BCH codes, we have shown that $S_{2i} = S_i^2$. With this equality, the $2t$ syndrome components can be computed with $(n - 1)t$ additions and nt multiplications.

For hardware implementation, the syndrome components may be computed with feedback shift registers as described in Section 6.7. We may use either the type of circuits shown in Figures 6.7 and 6.8 or the type of circuit shown in Figure 6.10. The second type of circuit is simpler. From the expression of (6.3), we see that the generator polynomial is a product of at most t minimal polynomials. Therefore, at most t feedback shift registers, each consisting of at most m stages, are needed to form the $2t$ syndrome components. The computation is performed as the received polynomial $\mathbf{r}(X)$ enters the decoder. As soon as the entire $\mathbf{r}(X)$ has entered the decoder, the $2t$ syndrome components are formed. It takes n clock cycles to complete the computation. A syndrome computation circuit for the double-error-correcting (15, 7) BCH code is shown in Figure 6.11, where two feedback shift registers, each with four stages, are employed.

The advantage of hardware implementation of syndrome computation is speed; however, software implementation is less expensive.

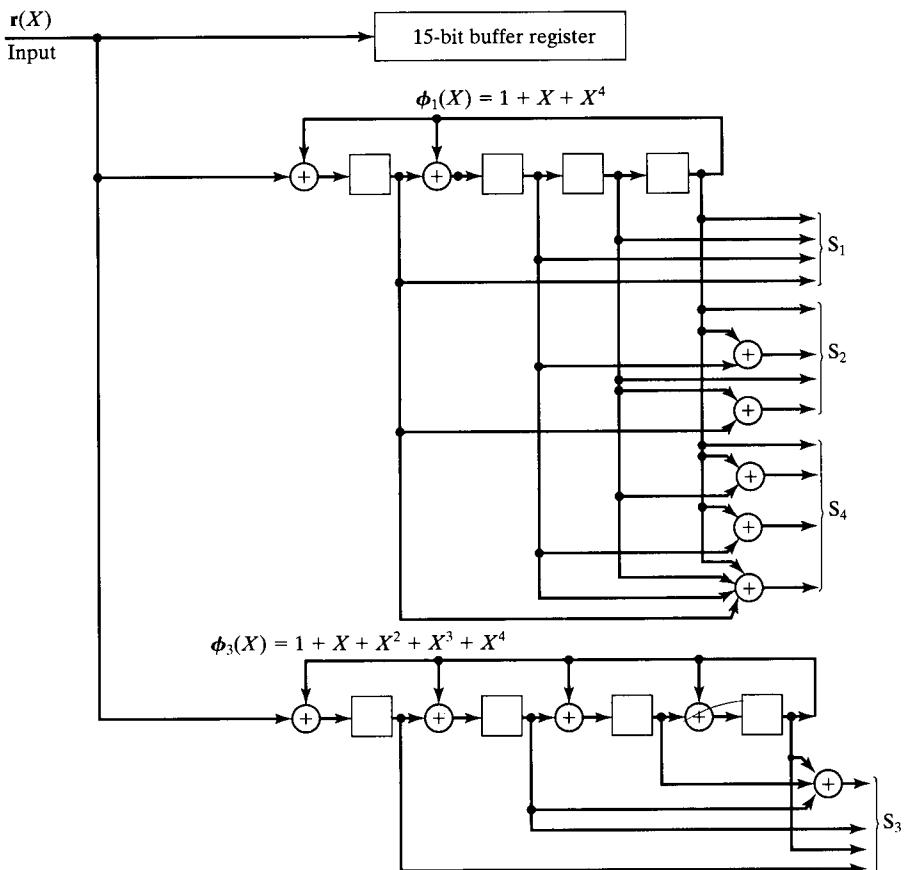


FIGURE 6.11: Syndrome computation circuit for the double-error-correcting (15, 7) BCH code.

6.8.2 Finding the Error-Location Polynomial $\sigma(X)$

For this step the software computation requires somewhat fewer than t additions and t multiplications to compute each $\sigma^{(\mu)}(X)$ and each d_μ , and since there are t of each, the total is roughly $2t^2$ additions and $2t^2$ multiplications. A pure hardware implementation requires the same total, and the speed depends on how much is done in parallel. The type of circuit shown in Figure 6.2 may be used for addition, and the type of circuits shown in Figures 6.5 and 6.6 may be used for multiplications. A very fast hardware implementation of finding $\sigma(X)$ would probably be very expensive, whereas a simple hardware implementation would probably be organized much like a general-purpose computer, except with a wired rather than a stored program.

6.8.3 Computation of Error-Location Numbers and Error Correction

In the worst case, this step requires substituting n field elements into an error-location polynomial $\sigma(X)$ of degree t to determine its roots. In software this requires nt multiplications and nt additions. This step can also be performed in hardware using Chien's

searching circuit, shown in Figure 6.11. Chien's searching circuit requires t multipliers for multiplying by $\alpha, \alpha^2, \dots, \alpha^t$, respectively. These multipliers may be the type of circuits shown in Figures 6.3 and 6.4. Initially, $\sigma_1, \sigma_2, \dots, \sigma_t$ found in step 2 are loaded into the registers of the t multipliers. Then, these multipliers are shifted n times. At the end of the l th shift, the t registers contain $\sigma_1\alpha^l, \sigma_2\alpha^{2l}, \dots, \sigma_t\alpha^{tl}$. Then, the sum

$$1 + \sigma_1\alpha^l + \sigma_2\alpha^{2l} + \dots + \sigma_t\alpha^{tl}$$

is tested. If the sum is zero, α^{n-l} is an error-location number; otherwise, α^{n-l} is not an error-location number. This sum can be formed by using t m -input modulo-2 adders. An m -input OR gate is used to test whether the sum is zero. It takes n clock cycles to complete this step. If we want to correct only the message digits, only k clock cycles are needed. A Chien's searching circuit for the double-error-correcting (15, 6) BCH code is shown in Figure 6.12.

For large t and m , the cost for building t wired multipliers for multiplying $\alpha, \alpha^2, \dots, \alpha^t$ in one clock cycle becomes substantial. For more economical but

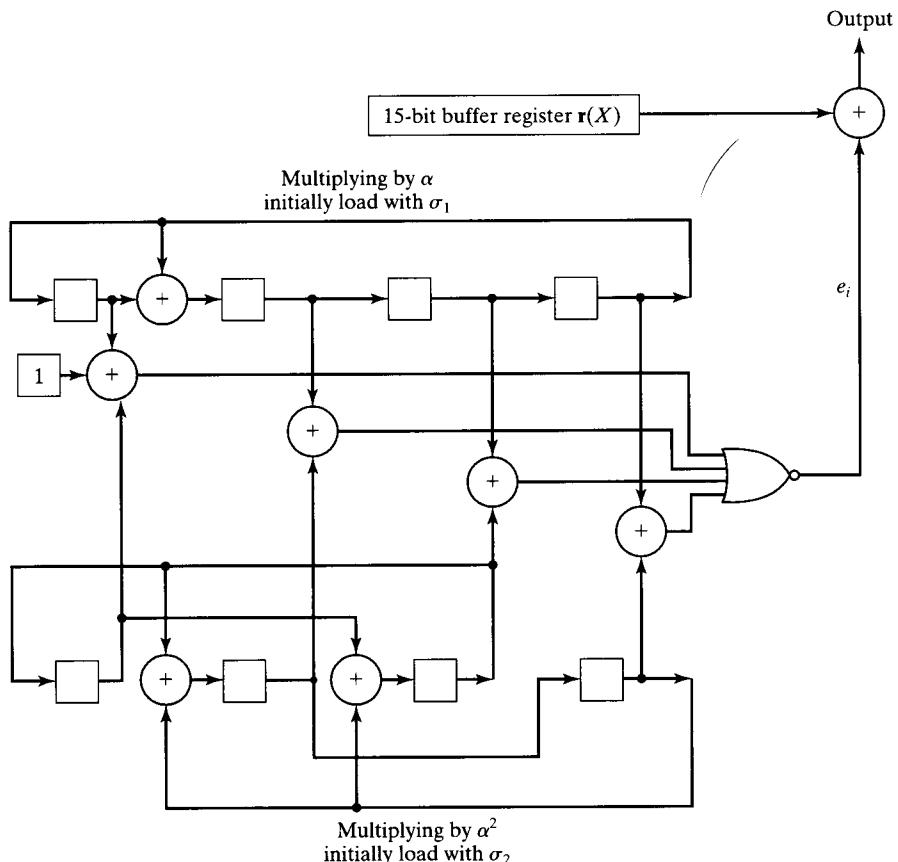


FIGURE 6.12: Chien's searching circuit for the double-error-correcting (15, 7) BCH code.

slower multipliers, we may use the type of circuit shown in Figure 6.5 (or shown in Figure 6.6). Initially, σ_i is loaded into register B, and α^i is stored in register C. After m clock cycles, the product $\sigma_i \alpha^i$ is in register A. To form $\sigma_i \alpha^{2i}$, $\sigma_i \alpha^i$ is loaded into register B. After another m clock cycles, $\sigma_i \alpha^{2i}$ will be in register A. Using this type of multiplier, nm clock cycles are needed to complete the third step of decoding a binary BCH code.

Steps 1 and 3 involve roughly the same amount of computation. Because n is generally much larger than t , $4nt$ is much larger than $4t^2$, and steps 1 and 3 involve most of the computation. Thus, hardware implementation of these steps is essential if high decoding speed is needed. With hardware implementation, step 1 can be done as the received polynomial $r(X)$ is read in, and step 3 can be accomplished as $r(X)$ is read out. In this case the computation time required in steps 1 and 3 is essentially negligible.

6.9 WEIGHT DISTRIBUTION AND ERROR DETECTION OF BINARY BCH CODES

The weight distributions of double-error-correcting, triple-error-correcting, and some low-rate primitive BCH codes have been completely determined; however, the weight distributions for the other BCH codes are still unknown. The weight distribution of a double-error-correcting or a triple-error-correcting primitive BCH code can be determined by first computing the weight distribution of its dual code and then applying the MacWilliams identity of (3.32). The weight distribution of the dual of a double-error-correcting primitive BCH code of length $2^m - 1$ is given in

TABLE 6.9: Weight distribution of the dual of a double-error-correcting primitive binary BCH code of length $2^m - 1$.

Odd $m \geq 3$	
Weight, i	Number of vectors with weight i , B_i
0	1
$2^{m-1} - 2^{(m+1)/2-1}$	$[2^{m-2} + 2^{(m-1)/2-1}](2^m - 1)$
2^{m-1}	$(2^m - 2^{m-1} + 1)(2^m - 1)$
$2^{m-1} + 2^{(m+1)/2-1}$	$[2^{m-2} - 2^{(m-1)/2-1}](2^m - 1)$

TABLE 6.10: Weight distribution of the dual of a double-error-correcting primitive binary BCH code of length $2^m - 1$.

Even $m \geq 4$	
Weight, i	Number of vectors with weight i , B_i
0	1
$2^{m-1} - 2^{(m+2)/2-1}$	$2^{(m-2)/2-1}[2^{(m-2)/2} + 1](2^m - 1)/3$
$2^{m-1} - 2^{m/2-1}$	$2^{(m+2)/2-1}(2^{m/2} + 1)(2^m - 1)/3$
2^{m-1}	$(2^{m-2} + 1)(2^m - 1)$
$2^{m-1} + 2^{m/2-1}$	$2^{(m+2)/2-1}(2^{m/2} - 1)(2^m - 1)/3$
$2^{m-1} + 2^{(m+2)/2-1}$	$2^{(m-2)/2-1}[2^{(m-2)/2} - 1](2^m - 1)/3$

TABLE 6.11: Weight distribution of the dual of a triple-error-correcting primitive binary BCH code of length $2^m - 1$.

Odd $m \geq 5$	
Weight, i	Number of Vectors with weight i , B_i
0	1
$2^{m-1} - 2^{(m+1)/2}$	$2^{(m-5)/2}[2^{(m-3)/2} + 1](2^{m-1} - 1)(2^m - 1)/3$
$2^{m-1} - 2^{(m-1)/2}$	$2^{(m-3)/2}[2^{(m-1)/2} + 1](5 \cdot 2^{m-1} + 4)(2^m - 1)/3$
2^{m-1}	$(9 \cdot 2^{2m-4} + 3 \cdot 2^{m-3} + 1)(2^m - 1)$
$2^{m-1} + 2^{(m-1)/2}$	$2^{(m-3)/2}[2^{(m-1)/2} - 1](5 \cdot 2^{m-1} + 4)(2^m - 1)/3$
$2^{m-1} + 2^{(m+1)/2}$	$2^{(m-5)/2}[2^{(m-3)/2} - 1](2^{m-1} - 1)(2^m - 1)/3$

TABLE 6.12: Weight distribution of the dual of a triple-error-correcting primitive binary BCH code of length $2^m - 1$.

Even $m \geq 6$	
Weight, i	Number of vectors with weight i , B_i
0	1
$2^{m-1} - 2^{(m+4)/2-1}$	$[2^{m-1} + 2^{(m+4)/2-1}](2^m - 4)(2^m - 1)/960$
$2^{m-1} - 2^{(m+2)/2-1}$	$7[2^{m-1} + 2^{(m+2)/2-1}]2^m(2^m - 1)/48$
$2^{m-1} - 2^{m/2-1}$	$2(2^{m-1} + 2^{m/2-1})(3 \cdot 2^m + 8)(2^m - 1)/15$
2^{m-1}	$(29 \cdot 2^{2m} - 4 \cdot 2^m + 64)(2^m - 1)/64$
$2^{m-1} + 2^{m/2-1}$	$2(2^{m-1} - 2^{m/2-1})(3 \cdot 2^m + 8)(2^m - 1)/15$
$2^{m-1} + 2^{(m+2)/2-1}$	$7[2^{m-1} - 2^{(m+2)/2-1}]2^m(2^m - 1)/48$
$2^{m-1} + 2^{(m+4)/2-1}$	$[2^{m-1} - 2^{(m+4)/2-1}](2^m - 4)(2^m - 1)/960$

Tables 6.9 and 6.10. The weight distribution of the dual of a triple-error-correcting primitive BCH code is given in Tables 6.11 and 6.12. Results presented in Tables 6.9 to 6.11 were mainly derived by Kasami [22]. For more on the weight distribution of primitive binary BCH codes, the reader is referred to [9] and [22].

If a double-error-correcting or a triple-error-correcting primitive BCH code is used for error detection on a BSC with transition probability p , its probability of an undetected error can be computed from (3.36) and one of the weight distribution tables, Tables 6.9 to 6.12. It has been proved [23] that the probability of an undetected error, $P_u(E)$, for a double-error-correcting primitive BCH code of length $2^m - 1$ is upper bounded by 2^{-2m} for $p \leq \frac{1}{2}$, where $2m$ is the number of parity-check digits in the code. The probability of an undetected error for a triple-error-correcting primitive BCH code of length $2^m - 1$ with odd m satisfies the upper bound 2^{-3m} , where $3m$ is the number of parity-check digits in the code [24].

It would be interesting to know how a general t -error-correcting primitive BCH code performs when it is used for error detection on a BSC with transition probability p . It has been proved [25] that for a t -error-correcting primitive BCH of

length $2^m - 1$, if the number of parity-check digits is equal to mt , and m is greater than a certain constant $m_0(t)$, the number of codewords of weight i satisfies the following equalities:

$$A_i = \begin{cases} 0 & \text{for } 0 < i \leq 2t \\ (1 + \lambda_0 \cdot n^{1/10}) \binom{n}{i} 2^{-(n-k)} & \text{for } i > 2t, \end{cases} \quad (6.48)$$

where $n = 2^m - 1$, and λ_0 is upper bounded by a constant. From (3.19) and (6.48), we obtain the following expression for the probability of an undetected error:

$$P_u(E) = (1 + \lambda_0 \cdot n^{-1/10}) 2^{-(n-k)} \sum_{i=2t+1}^n \binom{n}{i} p^i (1-p)^{n-i}. \quad (6.49)$$

Let $\varepsilon = (2t + 1)/n$. Then the summation of (6.49) can be upper bounded as follows [13]:

$$\sum_{i=n\varepsilon}^n \binom{n}{i} p^i (1-p)^{n-i} \leq 2^{-nE(\varepsilon, p)} \quad (6.50)$$

provided that $p < \varepsilon$, where

$$\begin{aligned} E(\varepsilon, p) &= H(p) + (\varepsilon - p)H'(p) - H(\varepsilon) \\ H(x) &= -x \log_2 x - (1-x) \log_2(1-x), \end{aligned}$$

and

$$H'(x) = \log_2 \frac{1-x}{x}.$$

$E(\varepsilon, p)$ is positive for $\varepsilon > p$. Combining (6.49) and (6.50), we obtain the following upper bound on $P_u(E)$ for $\varepsilon > p$:

$$P_u(E) \leq (1 + \lambda_0 \cdot n^{-1/10}) 2^{-nE(\varepsilon, p)} 2^{-(n-k)}$$

For $p < \varepsilon$ and sufficient large n , $P_u(E)$ can be made very small. For $p \geq \varepsilon$, we can also derive a bound on $P_u(E)$. It is clear from (6.49) that

$$P_u(E) \leq (1 + \lambda_0 \cdot n^{-1/10}) 2^{-(n-k)} \sum_{i=0}^n \binom{n}{i} p^i (1-p)^{n-i}.$$

Because

$$\sum_{i=0}^n \binom{n}{i} p^i (1-p)^i = 1,$$

we obtain the following upper bound on $P_u(E)$:

$$P_u(E) \leq (1 + \lambda_0 \cdot n^{-1/10}) 2^{-(n-k)}. \quad (6.51)$$

We see that for $p \geq \varepsilon$, P_u still decreases exponentially with the number of parity-check digits, $n - k$. If we use a sufficiently large number of parity-check digits, the probability of an undetected error $P_u(E)$ will become very small. Now, we may

summarize the preceding results above as follows: For a t -error-correcting primitive BCH code of length $n = 2^m - 1$ with number of parity-check digits $n - k = mt$ and $m \geq m_0(t)$, its probability of an undetected error on a BSC with transition probability p satisfies the following bounds:

$$P_u(E) \leq \begin{cases} (1 + \lambda_0 \cdot n^{-1/10}) 2^{-n[1-R+E(\varepsilon,p)]} & \text{for } p < \varepsilon \\ (1 + \lambda_0 \cdot n^{-1/10}) 2^{-n(1-R)} & \text{for } p \geq \varepsilon \end{cases} \quad (6.52)$$

where $\varepsilon = (2t + 1)/n$, $R = k/n$, and λ_0 is a constant.

The foregoing analysis indicates that primitive BCH codes are very effective for error detection on a BSC.

6.10 REMARKS

BCH codes form a subclass of a very special class of linear codes known as *Goppa codes* [21, 22]. It has been proved that the class of Goppa codes contains good codes. Goppa codes are in general noncyclic (except the BCH codes), and they can be decoded much like BCH codes. The decoding also consists of four steps: (1) compute the syndromes; (2) determine the error-location polynomial $\sigma(X)$; (3) find the error-location numbers; and (4) evaluate the error values (this step is not needed for binary Goppa codes). Berlekamp's iterative algorithm for finding the error-location polynomial for a BCH code can be modified for finding the error-location polynomial for Goppa codes [26]. Discussion of Goppa codes is beyond the scope of this introductory book. Moreover, implementation of BCH codes is simpler than that of Goppa codes, and no Goppa codes better than BCH codes have been found. For details on Goppa codes, the reader is referred to [26]–[30].

Our presentation of BCH codes and their implementation is given in the time domain. BCH codes also can be defined and implemented in the frequency domain using Fourier transforms over Galois fields. Decoding BCH codes in the frequency domain sometimes offers computational or implementation advantages. This topic will be discussed in Chapter 7.

PROBLEMS

- 6.1** Consider the Galois field $GF(2^4)$ given by Table 2.8. The element $\beta = \alpha^7$ is also a primitive element. Let $\mathbf{g}_0(X)$ be the lowest-degree polynomial over $GF(2)$ that has

$$\beta, \beta^2, \beta^3, \beta^4$$

as its roots. This polynomial also generates a double-error-correcting primitive BCH code of length 15.

- a. Determine $\mathbf{g}_0(X)$.
 - b. Find the parity-check matrix for this code.
 - c. Show that $\mathbf{g}_0(X)$ is the reciprocal polynomial of the polynomial $\mathbf{g}(X)$ that generates the (15, 7) double-error-correcting BCH code given in Example 6.1.
- 6.2** Determine the generator polynomials of all the primitive BCH codes of length 31. Use the Galois field $GF(2^5)$ generated by $\mathbf{p}(X) = 1 + X^2 + X^5$.

- 6.3** Suppose that the double-error-correcting BCH code of length 31 constructed in Problem 6.2 is used for error correction on a BSC. Decode the received polynomials $\mathbf{r}_1(X) = X^7 + X^{30}$ and $\mathbf{r}_2(X) = 1 + X^{17} + X^{28}$.
- 6.4** Consider a t -error-correcting primitive binary BCH code of length $n = 2^m - 1$. If $2t + 1$ is a factor of n , prove that the minimum distance of the code is exactly $2t + 1$. (*Hint:* Let $n = l(2t + 1)$. Show that $(X^n + 1)/(X^l + 1)$ is a code polynomial of weight $2t + 1$.)
- 6.5** Is there a binary t -error-correcting BCH code of length $2^m + 1$ for $m \geq 3$ and $t < 2^{m-1}$? If there is such a code, determine its generator polynomial.
- 6.6** Consider the field $GF(2^4)$ generated by $\mathbf{p}(X) = 1 + X + X^4$ (see Table 2.8). Let α be a primitive element in $GF(2^4)$ such that $\mathbf{p}(\alpha) = 0$. Devise a circuit that is capable of multiplying any element in $GF(2^4)$ by α^7 .
- 6.7** Devise a circuit that is capable of multiplying any two elements in $GF(2^5)$. Use $\mathbf{p}(X) = 1 + X^2 + X^5$ to generate $GF(2^5)$.
- 6.8** Devise a syndrome computation circuit for the binary double-error-correcting (31, 21) BCH code.
- 6.9** Devise a Chien's searching circuit for the binary double-error-correcting (31, 21) BCH code.
- 6.10** Consider the Galois field $GF(2^6)$ given by Table 6.2. Let $\beta = \alpha^3$, $l_0 = 2$, and $d = 5$. Determine the generator polynomial of the BCH code that has

$$\beta^2, \beta^3, \beta^4, \beta^5$$

as its roots (the general form presented at the end of Section 6.1). What is the length of this code?

- 6.11** Let $l_0 = -t$ and $d = 2t + 2$. Then we obtain a BCH code of designed distance $2t + 2$ whose generator polynomial has

$$\beta^{-t}, \dots, \beta^{-1}, \beta^0, \beta^1, \dots, \beta^t$$

and their conjugates as all its roots.

- a. Show that this code is a reversible cyclic code.
- b. Show that if t is odd, the minimum distance of this code is at least $2t + 4$. (*Hint:* Show that $\beta^{-(t+1)}$ and β^{t+1} are also roots of the generator polynomial.)

BIBLIOGRAPHY

1. A. Hocquenghem, "Codes corecteurs d'erreurs," *Chiffres*, 2: 147–56, 1959.
2. R. C. Bose and D. K. Ray-Chaudhuri, "On a Class of Error Correcting Binary Group Codes," *Inform. Control*, 3: 68–79, March 1960.
3. W. W. Peterson, "Encoding and Error-Correction Procedures for the Bose–Chaudhuri Codes," *IRE Trans. Inform. Theory*, IT-6: 459–70, September 1960.
4. D. Gorenstein and N. Zierler, "A Class of Cyclic Linear Error-Correcting Codes in p^m Symbols," *J. Soc. Ind. Appl. Math.*, 9: 107–214, June 1961.
5. I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *J. Soc. Ind. Appl. Math.*, 8: 300–304, June 1960.

6. R. T. Chien, "Cyclic Decoding Procedure for the Bose-Chaudhuri-Hocquenghem Codes," *IEEE Trans. Inform. Theory*, IT-10: 357–63, October 1964.
7. G. D. Forney, "On Decoding BCH Codes," *IEEE Trans. Inform. Theory*, IT-11: 549–57, October 1965.
8. E. R. Berlekamp, "On Decoding Binary Bose-Chaudhuri-Hocquenghem Codes," *IEEE Trans. Inform. Theory*, IT-11: 577–80, October 1965.
9. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
10. J. L. Massey, "Step-by-Step Decoding of the Bose-Chaudhuri-Hocquenghem Codes," *IEEE Trans. Inform. Theory*, IT-11: 580–85, October 1965.
11. J. L. Massey, "Shift-Register Synthesis and BCH Decoding," *IEEE Trans. Inform. Theory*, IT-15: 122–27, January 1969.
12. H. O. Burton, "Inversionless Decoding of Binary BCH Codes," *IEEE Trans. Inform. Theory*, IT-17: 464–66, July 1971.
13. W. W. Peterson and E. J. Weldon, *Error-Correcting Codes*, 2d ed., MIT Press, Cambridge, 1970.
14. F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North Holland, Amsterdam, 1977.
15. G. C. Clark, Jr., and J. B. Cain, *Error-Correction Coding for Digital Communications*, Plenum Press, New York, 1981.
16. R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley Reading Mass., 1983.
17. S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, Englewood Cliffs, N.J., 1995.
18. H. B. Mann, "On the Number of Information Symbols in Bose-Chaudhuri Codes," *Inform. Control*, 5: 153–62, June 1962.
19. C. L. Chen, "High-Speed Decoding of BCH Codes," *IEEE Trans. Inform. Theory*, IT-27(2): 254–56, March 1981.
20. T. Kasami, S. Lin, and W. W. Peterson, "Some Results on Weight Distributions of BCH Codes," *IEEE Trans. Inform. Theory*, IT-12(2): 274, April 1966.
21. T. Kasami, S. Lin, and W. W. Peterson, "Some Results on Cyclic Codes Which are Invariant under the Affine Group," *Scientific Report AFCRL-66-622*, Air Force Cambridge Research Labs, Bedford, Mass., 1966.
22. T. Kasami, "Weight Distributions of Bose-Chaudhuri-Hocquenghem Codes," *Proc. Conf. Combinatorial Mathematics and Its Applications*, R. C. Bose and T. A. Dowling, eds., University of North Carolina Press, Chapel Hill, 1968.

23. S. K. Leung-Yan-Cheong, E. R. Barnes, and D. U. Friedman, "On Some Properties of the Undetected Error Probability of Linear Codes," *IEEE Trans. Inform. Theory*, IT-25(1): 110–12, January 1979.
24. G. T. Ong and C. Leung, "On the Undetected Error Probability of Triple-Error-Correcting BCH Codes," *IEEE Trans. Inform. Theory*, IT-37: 673–78, 1991.
25. V. M. Sidel'nikov, "Weight Spectrum of Binary Bose–Chaudhuri–Hocquenghem Code." *Probl. Inform. Transm.*, 7(1): 11–17, 1971.
26. V. D. Goppa, "A New Class of Linear Codes," *Probl. Peredachi Inform.*, 6(3): 24–30, September 1970.
27. V. D. Goppa, "Rational Representation of Codes and (L, g) Codes," *Probl. Peredachi Inform.*, 7(3): 41–49, September 1971.
28. N. J. Patterson, "The Algebraic Decoding of Goppa Codes," *IEEE Trans. Inform. Theory*, IT-21: 203–7, March 1975.
29. E. R. Berlekamp, "Goppa Codes," *IEEE Trans. Inform. Theory*, IT-19(5): 590–92, September 1973.
30. Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A Method for Solving Key Equation for Decoding Goppa Codes," *Inform. Control*, 27: 87–99, January 1975.
31. R. E. Blahut, "Transform Techniques for Error Control Codes," *IBM J. Res. Dev.*, 23(3): 299–315 May 1979.

CHAPTER 7

Nonbinary BCH Codes, Reed–Solomon Codes, and Decoding Algorithms

This chapter presents nonbinary BCH codes with symbols from $GF(q)$ and their decoding algorithms. The most important and popular subclass of nonbinary BCH codes is the class of Reed–Solomon codes. Even though Reed–Solomon codes form a subclass of BCH codes, they were constructed independently using a totally different approach by Reed and Solomon in 1960 [1], the same year as BCH codes were discovered. The relationship between Reed–Solomon codes and BCH codes was proved by Gorenstein and Zierler in 1961 [2]. The minimum distance of a Reed–Solomon code is equal to the number of its parity-check symbols plus one. Reed–Solomon codes are very effective in correcting random symbol errors and random burst errors, and they are widely used for error control in communication and data storage systems, ranging from deep-space telecommunications to compact discs. Concatenation of these codes as outer codes with simple binary codes as inner codes provides high data reliability with reduced decoding complexity.

Decoding of a nonbinary BCH code or a Reed–Solomon code requires determination of both the locations and the values of the symbol errors. The first error-correction procedure for nonbinary BCH and Reed–Solomon codes was found by Gorenstein and Zierler [2], and it was later improved by Chien [3] and Forney [4]. But Berlekamp’s iterative decoding algorithm [5] presented in the previous chapter was the first efficient decoding algorithm for both binary and nonbinary BCH codes. In 1975 Sugiyama, Kasahara, Hirasawa, and Namekawa showed that the Euclidean algorithm for finding the greatest common divisor of two polynomials can be used for decoding BCH and Reed–Solomon codes [6]. This Euclidean decoding algorithm is simple in concept and easy to implement. BCH and Reed–Solomon codes can also be decoded in the frequency domain. The first such frequency-domain decoding algorithm was introduced by Gore [7], and it was later much improved by Blahut [8]. All the preceding decoding algorithms can be modified for correcting both symbol errors and erasures.

Reed–Solomon codes have been proved to be good error-detecting codes [9], and their weight distribution has been completely determined [10]–[12].

Good treatment of nonbinary BCH and Reed–Solomon codes and their decoding algorithms can be found in [5] and [13–20].

7.1 q -ARY LINEAR BLOCK CODES

Consider a Galois field $GF(q)$ with q elements. It is possible to construct codes with symbols from $GF(q)$. These codes are called q -ary codes. The concepts and

properties developed for the binary codes in the previous chapters apply to q -ary codes with few modifications.

Consider the vector space of all the q^n n -tuples over $GF(q)$:

$$(v_0, v_1, \dots, v_{n-1})$$

with $v_i \in GF(q)$ for $0 \leq i < n$. The vector addition is defined as follows:

$$(u_0, u_1, \dots, u_{n-1}) + (v_0, v_1, \dots, v_{n-1}) \stackrel{\Delta}{=} (u_0 + v_0, u_1 + v_1, \dots, u_{n-1} + v_{n-1}),$$

where the addition $u_i + v_i$ is carried out in $GF(q)$. The multiplication of a scalar in $GF(q)$ and an n -tuple $(v_0, v_1, \dots, v_{n-1})$ over $GF(q)$ is given as follows:

$$a \cdot (v_0, v_1, \dots, v_{n-1}) \stackrel{\Delta}{=} (a \cdot v_0, a \cdot v_1, \dots, a \cdot v_{n-1})$$

where the product $a \cdot v_i$ is carried out in $GF(q)$. The inner product of two n -tuples, $(u_0, u_1, \dots, u_{n-1})$ and $(v_0, v_1, \dots, v_{n-1})$, is defined as follows:

$$(u_0, u_1, \dots, u_{n-1}) \cdot (v_0, v_1, \dots, v_{n-1}) \stackrel{\Delta}{=} \sum_{i=0}^{n-1} u_i \cdot v_i$$

where addition and multiplication are carried out in $GF(q)$.

DEFINITION 7.1 An (n, k) linear block code with symbols from $GF(q)$ is simply a k -dimensional subspace of the vector space of all the n -tuples over $GF(q)$.

A q -ary linear block code has all the structures and properties developed for binary linear block codes. A q -ary linear block code is specified either by a generator matrix or by a parity-check matrix over $GF(q)$. Encoding and decoding of q -ary linear codes are the same as for binary linear codes, except that operations and computations are performed over $GF(q)$.

A q -ary (n, k) cyclic code is generated by a polynomial of degree $n - k$ over $GF(q)$,

$$\mathbf{g}(X) = g_0 + g_1 X + \dots + g_{n-k-1} X^{n-k-1} + X^{n-k}$$

where $g_0 \neq 0$ and $g_i \in GF(q)$. The generator polynomial $\mathbf{g}(X)$ is a factor of $X^n - 1$. A polynomial $\mathbf{v}(X)$ of degree $n - 1$ or less over $GF(q)$ is a code polynomial if and only if $\mathbf{v}(X)$ is divisible by the generator polynomial $\mathbf{g}(X)$.

In this chapter we present two important classes of cyclic codes over $GF(q)$ whose constructions are based on an extension field of $GF(q)$. Construction of an extension field of $GF(q)$ is similar to the construction of an extension field of $GF(2)$.

A polynomial $f(X)$ with coefficients from $GF(q)$ is called *monic* if the coefficient of the highest power of X is 1. A polynomial $p(X)$ of degree m over $GF(q)$ is said to be *irreducible* if it is not divisible by any polynomial over $GF(q)$ of degree less than m but greater than zero. An irreducible polynomial $p(X)$ of degree m over $GF(q)$ is called *primitive* if the smallest positive integer n for which $p(X)$ divides $X^n - 1$ is $n = q^m - 1$.

For any positive integer m , a Galois field $GF(q^m)$ with q^m elements can be constructed from the ground field $GF(q)$. The construction is exactly the same as the

construction of $GF(2^m)$ from $GF(2)$. The construction of $GF(q^m)$ is based on a monic primitive polynomial $p(X)$ of degree m over $GF(q)$. Let α be a root of $p(X)$. Then,

$$0, 1, \alpha, \alpha^2, \dots, \alpha^{q^m-2}$$

form all the elements of $GF(q^m)$, and $\alpha^{q^m-1} = 1$. The element α is called a *primitive element*. Every element β in $GF(q^m)$ can be expressed as a polynomial in α ,

$$\beta = a_0 + a_1\alpha + a_2\alpha^2 + \dots + a_{m-1}\alpha^{q^m-1}$$

where $a_i \in GF(q)$ for $0 \leq i < m$. Then, $(a_0, a_1, \dots, a_{m-1})$ is a vector representation of β . Therefore, every element in $GF(q^m)$ has three forms: power (0 is represented by α^∞), polynomial, and vector forms.

The elements in $GF(q^m)$ form all the roots of $X^{q^m} - X$. Let β be an element in $GF(q^m)$. The minimal polynomial of β is the monic polynomial $\phi(X)$ of the smallest degree over $GF(q)$ that has β as a root; that is, $\phi(\beta) = 0$. Just as in the binary case, $\phi(X)$ is irreducible. Let e be the smallest nonnegative integer for which $\beta^{q^e} = \beta$. The integer e is called the *exponent* of β and $e \leq m$. The elements $\beta, \beta^q, \beta^{q^2}, \dots, \beta^{q^{e-1}}$ are conjugates. Then,

$$\phi(X) = \prod_{i=0}^{e-1} (X - \beta^{q^i}),$$

and $\phi(X)$ divides $X^{q^m} - X$.

7.2 PRIMITIVE BCH CODES OVER $GF(q)$

The binary BCH codes defined in Section 6.1 can be generalized to nonbinary codes in a straightforward manner. Let α be a primitive element in $GF(q^m)$. The generator polynomial $\mathbf{g}(X)$ of a t -error-correcting primitive q -ary BCH code is the polynomial of the smallest degree over $GF(q)$ that has $\alpha, \alpha^2, \dots, \alpha^{2t}$ as roots. For $1 \leq i \leq 2t$, let $\phi_i(X)$ be the minimal polynomial of α^i . Then,

$$\mathbf{g}(X) = \text{LCM}\{\phi_1(X), \phi_2(X), \dots, \phi_{2t}(X)\}. \quad (7.1)$$

Because each $\phi_i(X)$ divides $X^{q^m-1} - 1$, $\mathbf{g}(X)$ divides $X^{q^m-1} - 1$. Since α is a primitive element in $GF(q^m)$, $\phi_1(X)$ is a primitive polynomial of degree m . Hence, the smallest positive integer n for which $\phi_1(X)$ divides $X^n - 1$ is $n = q^m - 1$. This result implies that $q^m - 1$ is the smallest positive integer for which $\mathbf{g}(X)$ divides $X^{q^m-1} - 1$. Because the degree of each $\phi_i(X)$ is m or less, the degree of $\mathbf{g}(X)$ is $2mt$ or less. Similar to the way we proved the BCH bound for the minimum distance of a binary BCH code, we can prove that the minimum distance of the q -ary BCH code generated by $\mathbf{g}(X)$ of (7.1) is lower bounded by $2t + 1$.

Summarizing the foregoing results, we see that the q -ary BCH code generated by the polynomial $\mathbf{g}(X)$ of (7.1) is a cyclic code with the following parameters:

Block length: $n = q^m - 1$,

Number of parity-check symbols: $n - k \leq 2mt$,

Minimum distance: $d_{min} \geq 2t + 1$.

This code is capable of correcting t or fewer random symbol errors over a span of $q^m - 1$ symbol positions. For $q = 2$, we obtain the binary primitive BCH codes. Similar to the binary case, the matrix over $GF(q^m)$,

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & \cdots & (\alpha^2)^{n-1} \\ 1 & \alpha^3 & (\alpha^3)^2 & \cdots & (\alpha^3)^{n-1} \\ \vdots & & & & \vdots \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & \cdots & (\alpha^{2t})^{n-1} \end{bmatrix},$$

is a parity-check matrix of the t -error-correcting primitive q -ary BCH code generated by the polynomial $\mathbf{g}(X)$ of (7.1).

7.3 REED-SOLOMON CODES

The special subclass of q -ary BCH codes for which $m = 1$ is the most important subclass of q -ary BCH codes. The codes of this subclass are called the Reed–Solomon (RS) codes in honor of their discoverers, Irving S. Reed and Gustave Solomon [1]. RS codes have been widely used for error control in both digital communication and storage systems.

Let α be a primitive element in $GF(q)$. The generator polynomial $\mathbf{g}(X)$ of a t -error-correcting RS code with symbols from $GF(q)$ has $\alpha, \alpha^2, \dots, \alpha^{2t}$ as all its roots. Because α^i is an element of $GF(q)$, its minimal polynomial $\phi_i(X)$ is simply $X - \alpha^i$. Then, it follows from (7.1) that

$$\begin{aligned} \mathbf{g}(X) &= (X - \alpha)(X - \alpha^2) \cdots (X - \alpha^{2t}) \\ &= g_0 + g_1 X + g_2 X^2 + \cdots + g_{2t-1} X^{2t-1} + X^{2t} \end{aligned} \tag{7.2}$$

with $g_i \in GF(q)$ for $0 \leq i < 2t$. Since $\alpha, \alpha^2, \dots, \alpha^{2t}$ are roots of $X^{q-1} - 1$, $\mathbf{g}(X)$ divides $X^{q-1} - 1$. Therefore, $\mathbf{g}(X)$ generates a q -ary cyclic code of length $n = q - 1$ with exactly $2t$ parity-check symbols. It follows from the BCH bound that the minimum distance of the code is at least $2t + 1$; however, the generator polynomial $\mathbf{g}(X)$ is a code polynomial and has $2t + 1$ terms. None of the coefficients in $\mathbf{g}(X)$ can be zero, otherwise the resulting codeword would have weight less than $2t + 1$, which would contradict the BCH bound on the minimum distance. Therefore, $\mathbf{g}(X)$ corresponds to a codeword of weight exactly $2t + 1$. This implies that the minimum distance of the RS code generated by the polynomial $\mathbf{g}(X)$ of (7.2) is exactly $2t + 1$, and the code is capable of correcting t or fewer symbol errors. In summary, a t -error-correcting RS code with symbols from $GF(q)$ has the following parameters:

Block length: $n = q - 1$,

Number of parity-check symbols: $n - k = 2t$,

Dimension: $k = q - 1 - 2t$,

Minimum distance: $d_{min} = 2t + 1$.

Thus, we see that RS codes have two important features: (1) the length of the code is one less than the size of the code alphabet, and (2) the minimum distance is one greater than the number of parity-check symbols. Codes with minimum distance one greater than the number of parity-check symbols are called *maximum distance separable* (MDS) codes. RS codes form the most important class of MDS codes.

EXAMPLE 7.1

Let α be a primitive element in $GF(2^6)$ constructed based on the primitive polynomial $p(X) = 1 + X + X^6$ (see Table 6.2). Consider the triple-error-correcting RS code with symbols from $GF(2^6)$. The generator polynomial $\mathbf{g}(X)$ of this code has $\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5$, and α^6 as all its roots; hence,

$$\begin{aligned}\mathbf{g}(X) &= (X + \alpha)(X + \alpha^2)(X + \alpha^3)(X + \alpha^4)(X + \alpha^5)(X + \alpha^6) \\ &= \alpha^{21} + \alpha^{10}X + \alpha^{55}X^2 + \alpha^{43}X^3 + \alpha^{48}X^4 + \alpha^{59}X^5 + X^6.\end{aligned}$$

The code is a $(63, 57)$ triple-error-correcting RS code over $GF(2^6)$.

Encoding of a RS code is similar to that of the binary case. Let

$$\mathbf{a}(X) = a_0 + a_1X + a_2X^2 + \cdots + a_{k-1}X^{k-1}$$

be the message to be encoded, where $k = n - 2t$. In systematic form, the $2t$ parity-check symbols are the coefficients of the remainder $\mathbf{b}(X) = b_0 + b_1X + \cdots + b_{2t-1}X^{2t-1}$ resulting from dividing the message polynomial $X^{2t}\mathbf{a}(X)$ by the generator polynomial. In hardware implementation, this is accomplished by using a division circuit as shown in Figure 7.1. As soon as the message $\mathbf{a}(X)$ has entered the channel and the circuit, the parity-check symbols appear in the register.

The weight distribution of Reed–Solomon codes has been completely determined [10]–[12]. For a t -error-correcting RS code of length $q - 1$ with symbols from $GF(q)$, the number of codewords of weight i is given by

$$A_i = \binom{q-1}{i} q^{-2t} \left\{ (q-1)^i + \sum_{j=0}^{2t} (-1)^{i+j} \binom{i}{j} (q^{2t} - q^i) \right\}, \quad (7.3)$$

for $2t + 1 \leq i \leq q - 1$.

Suppose a q -ary RS code is used for error detection on a discrete memoryless channel with q inputs and q outputs. Let $(1 - \varepsilon)$ be the probability that a transmitted symbol is received correctly and $\varepsilon/(q-1)$ be the probability that a transmitted symbol is changed into each of the $q - 1$ other symbols. Using the weight distribution given by (7.3), it can be shown that the probability of undetected error for a RS code is [9]

$$\begin{aligned}P_u(E) &= q^{-2t} \left\{ 1 + \sum_{i=0}^{2t-1} \binom{q-1}{i} (q^{2t} - q^i) \left(\frac{\varepsilon}{q-1}\right)^i \right. \\ &\quad \times \left. \left(1 - \frac{q\varepsilon}{q-1}\right)^{q-1-i} - q^{2t}(1-\varepsilon)^{q-1} \right\}.\end{aligned} \quad (7.4)$$

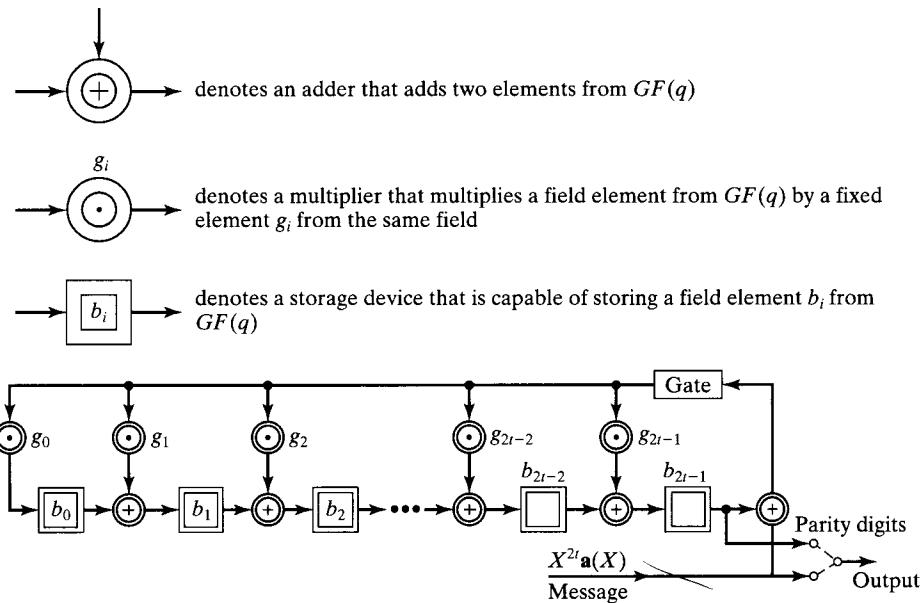


FIGURE 7.1: Encoding circuit for a q -ary RS code with generator polynomial $\mathbf{g}(X) = g_0 + g_1X + g_2X^2 + \dots + g_{2t-1}X^{2t-1} + X^{2t}$.

It also has been shown in [9] that $P_u(E) < q^{-2t}$ and decreases monotonically as ε decreases from $(q - 1)/q$ to 0. Hence, RS codes are good for error detection.

Let λ be a nonnegative integer less than t . Suppose a t -error-correcting q -ary RS code is used to correct all error patterns with λ or fewer symbol errors. Let $P_u(E, \lambda)$ denote the probability of undetected error after correction. It has also been proved in [9] that

$$\begin{aligned} P_u(E, \lambda) = & \sum_{h=0}^{\lambda} \binom{q-1}{h} \left[q^{-2t} (q-1)^h - \varepsilon^h (1-\varepsilon)^{q-1-h} \right. \\ & \left. + \sum_{l=0}^{\min\{2t-1, q-1-h\}} \binom{q-1-h}{l} \left(\frac{\varepsilon}{q-1} \right)^l \left(1 - \frac{q\varepsilon}{q-1} \right)^{q-1-h-l} R_{h,l}(\varepsilon) \right], \end{aligned} \quad (7.5)$$

where

$$R_{h,l}(\varepsilon) = \sum_{j=0}^{\min\{2t-1-l, h\}} (-1)^{h-j} \binom{h}{j} \left(1 - q^{-2t+l+j} \right) \left(1 - \frac{\varepsilon}{q-1} \right)^j \left(1 - \frac{q\varepsilon}{q-1} \right)^{h-j} \quad (7.6)$$

for $0 \leq l < 2t$. It is easy to check that $P_u(E)$ given by (7.4) can be obtained from $P_u(E, \lambda)$ of (7.5) by setting $\lambda = 0$. In [9] it is shown that the probability $P_u(E, \lambda)$ of

undetected error after decoding decreases monotonically from

$$(q^{-2t} - q^{-n}) \sum_{h=0}^{\lambda} \binom{q-1}{h} (q-1)^h \quad (7.7)$$

as ε decreases from $(q-1)/q$ to 0. Therefore, we have the following upper bound on $P_u(E, \lambda)$ from (7.7):

$$P_u(E, \lambda) < q^{-2t} \sum_{h=0}^{\lambda} \binom{q-1}{h} (q-1)^h \quad (7.8)$$

for $0 \leq \varepsilon \leq (q-1)/q$.

The preceding results on error probabilities indicate that RS codes are effective for pure error detection or simultaneous error correction and detection.

Consider the set of codewords in a $(q-1, q-1-2t)$ q -ary RS code whose l leading information symbols are identical to zero, with $0 \leq l < q-1-2t$. Deleting these l zero-information symbols from each codeword in the set, we obtain a shortened $(q-1-l, q-1-2t-l)$ RS code. The minimum distance of this shortened RS code is still $2t+1$. Hence, the shortened code is also a MDS code. Encoding and decoding of a shortened RS code are the same as those for the original RS code of natural length (see Section 5.10).

Two information symbols can be added to a RS code of length $q-1$ without reducing its minimum distance. The extended RS code has length $q+1$ and the same number of parity-check symbols as the original code. For a t -error-correcting RS code of length $q-1$, the parity-check matrix takes the form

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{q-2} \\ 1 & \alpha^2 & (\alpha^2)^2 & \cdots & (\alpha^2)^{q-2} \\ \vdots & & & & \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & \cdots & (\alpha^{2t})^{q-2} \end{bmatrix}.$$

Then, the parity-check matrix of the extended RS code is

$$H_1 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ \vdots & \vdots & H \\ 0 & 0 \\ 1 & 0 \end{bmatrix}. \quad (7.9)$$

This code is called a *doubly extended RS code* and is also a MDS code. The preceding result was first obtained by Kasami, Lin, and Peterson [11, 21] and later independently by Wolf [22]. If we delete the first column of H_1 , we obtain a singly extended RS code of length q , which is again a MDS code.

In all practical applications of RS codes for error control, q is set to 2^m , and code symbols are from the Galois field $GF(2^m)$.

7.4 DECODING OF NONBINARY BCH AND RS CODES: THE BERLEKAMP ALGORITHM

In this and the next three sections, we present various algorithms for decoding nonbinary BCH and RS codes.

Let

$$\mathbf{v}(X) = v_0 + v_1 X + \cdots + v_{n-1} X^{n-1}$$

be the transmitted code polynomial, and let

$$\mathbf{r}(X) = r_0 + r_1 X + \cdots + r_{n-1} X^{n-1}$$

be the corresponding received polynomial. Then, the error pattern added by the channel is

$$\begin{aligned}\mathbf{e}(X) &= \mathbf{r}(X) - \mathbf{v}(X) \\ &= e_0 + e_1 X + \cdots + e_{n-1} X^{n-1},\end{aligned}$$

where $e_i = r_i - v_i$ is a symbol in $GF(q)$. Suppose the error pattern $\mathbf{e}(X)$ contains v errors (nonzero components) at locations $X^{j_1}, X^{j_2}, \dots, X^{j_v}$ with $0 \leq j_1 < j_2 < \dots < j_v \leq n-1$. Then,

$$\mathbf{e}(X) = e_{j_1} X^{j_1} + e_{j_2} X^{j_2} + \cdots + e_{j_v} X^{j_v} \quad (7.10)$$

where $e_{j_1}, e_{j_2}, \dots, e_{j_v}$ are error values. Hence, to determine $\mathbf{e}(X)$, we need to know the error locations X^{j_i} 's and the error values e_{j_i} 's (i.e., we need to know the v pairs (X^{j_i}, e_{j_i}) 's).

As with binary BCH codes, the syndrome is a $2t$ -tuple over $GF(q^m)$:

$$(S_1, S_2, \dots, S_{2t})$$

with $S_i = \mathbf{r}(\alpha^i)$ for $1 \leq i \leq 2t$. Because $\mathbf{r}(X) = \mathbf{v}(X) + \mathbf{e}(X)$, we have

$$S_i = \mathbf{v}(\alpha^i) + \mathbf{e}(\alpha^i) = \mathbf{e}(\alpha^i). \quad (7.11)$$

From (7.10) and (7.11), we obtain the following set of equations that relate the error locations and error values to the syndrome of the received polynomial $\mathbf{r}(X)$:

$$\begin{aligned}S_1 &= e_{j_1} \alpha^{j_1} + e_{j_2} \alpha^{j_2} + \cdots + e_{j_v} \alpha^{j_v} \\ S_2 &= e_{j_1} \alpha^{2j_1} + e_{j_2} \alpha^{2j_2} + \cdots + e_{j_v} \alpha^{2j_v} \\ &\vdots \\ S_{2t} &= e_{j_1} \alpha^{2tj_1} + e_{j_2} \alpha^{2tj_2} + \cdots + e_{j_v} \alpha^{2tj_v}.\end{aligned} \quad (7.12)$$

For $1 \leq i \leq v$, let

$$\beta_i \stackrel{\Delta}{=} \alpha^{j_i} \quad \text{and} \quad \delta_i \stackrel{\Delta}{=} e_{j_i}$$

which are simply the error-location numbers and error values. With the preceding definitions of β_i and δ_i , we can simplify the equations of (7.12) as follows:

$$\begin{aligned} S_1 &= \delta_1\beta_1 + \delta_2\beta_2 + \cdots + \delta_v\beta_v \\ S_2 &= \delta_1\beta_1^2 + \delta_2\beta_2^2 + \cdots + \delta_v\beta_v^2 \\ &\vdots \\ S_{2t} &= \delta_1\beta_1^{2t} + \delta_2\beta_2^{2t} + \cdots + \delta_v\beta_v^{2t}. \end{aligned} \quad (7.13)$$

For decoding a q -ary BCH code or a RS code, the same three steps used for decoding a binary BCH code are required; in addition, a fourth step involving computation of error values is required. Therefore, the decoding consists of the following four steps:

1. Compute the syndrome (S_1, S_2, \dots, S_{2t}).
2. Determine the error-location polynomial $\sigma(X)$.
3. Determine the error-value evaluator.
4. Evaluate error-location numbers and error values and perform error correction.

As with binary BCH codes, the error-location polynomial $\sigma(X)$ is defined as

$$\begin{aligned} \sigma(X) &= (1 - \beta_1 X)(1 - \beta_2 X) \cdots (1 - \beta_v X) \\ &= \sigma_0 + \sigma_1 X + \sigma_2 X^2 + \cdots + \sigma_v X^v, \end{aligned} \quad (7.14)$$

where $\sigma_0 = 1$. The error-location numbers are the reciprocals of the roots of $\sigma(X)$. From (7.13) and (7.14), it is possible to obtain the following set of equations that relates the coefficients σ_i 's of $\sigma(X)$ and the syndrome components S_i 's:

$$\begin{aligned} S_{v+1} + \sigma_1 S_v + \sigma_2 S_{v-1} + \cdots + \sigma_v S_1 &= 0 \\ S_{v+2} + \sigma_1 S_{v+1} + \sigma_2 S_v + \cdots + \sigma_v S_2 &= 0 \\ &\vdots \\ S_{2t} + \sigma_1 S_{2t-1} + \sigma_2 S_{2t-2} + \cdots + \sigma_v S_{2t-v} &= 0. \end{aligned} \quad (7.15)$$

(These equalities will be derived later.) These equations are known as the *generalized Newton's identities*. Our objective is to find the minimum-degree polynomial $\sigma(X)$ whose coefficients satisfy these generalized Newton's identities. Once we have found $\sigma(X)$, we can determine the error locations and error values.

We can compute the error-location polynomial $\sigma(X)$ iteratively in $2t$ steps with Berlekamp's algorithm presented in Section 6.2. At the μ th step, we determine a polynomial of minimum degree

$$\sigma^{(\mu)}(X) = \sigma_0^{(\mu)} + \sigma_1^{(\mu)} X + \cdots + \sigma_{l_\mu}^{(\mu)} X^{l_\mu}$$

whose coefficients satisfy the following $\mu - l_\mu$ identities:

$$\begin{aligned} S_{l_\mu+1} + \sigma_1^{(\mu)} S_{l_\mu} + \cdots + \sigma_{l_\mu}^{(\mu)} S_1 &= 0 \\ S_{l_\mu+2} + \sigma_1^{(\mu)} S_{l_\mu+1} + \cdots + \sigma_{l_\mu}^{(\mu)} S_2 &= 0 \\ &\vdots \\ S_\mu + \sigma_1^{(\mu)} S_{\mu-1} + \cdots + \sigma_{l_\mu}^{(\mu)} S_{\mu-l_\mu} &= 0. \end{aligned} \tag{7.16}$$

The next step is to find a new polynomial of minimum degree

$$\sigma^{(\mu+1)}(X) = \sigma_0^{(\mu+1)} + \sigma_1^{(\mu+1)} X + \cdots + \sigma_{l_{\mu+1}}^{(\mu+1)} X^{l_{\mu+1}}$$

whose coefficients satisfy the following $(\mu+1) - l_{\mu+1}$ identities:

$$\begin{aligned} S_{l_{\mu+1}+1} + \sigma_1^{(\mu+1)} S_{l_{\mu+1}} + \cdots + \sigma_{l_{\mu+1}}^{(\mu+1)} S_1 &= 0 \\ S_{l_{\mu+1}+2} + \sigma_1^{(\mu+1)} S_{l_{\mu+1}+1} + \cdots + \sigma_{l_{\mu+1}}^{(\mu+1)} S_2 &= 0 \\ &\vdots \\ S_{\mu+1} + \sigma_1^{(\mu+1)} S_\mu + \cdots + \sigma_{l_{\mu+1}}^{(\mu+1)} S_{\mu+1-l_{\mu+1}} &= 0. \end{aligned} \tag{7.17}$$

We continue the foregoing process until $2t$ steps have been completed. At the $2t$ th step, we have

$$\sigma(X) = \sigma^{(2t)}(X),$$

which is the true error-location polynomial provided that the number of errors in $\mathbf{e}(X)$ does not exceed the error-correcting capability t . In this case, the coefficients of $\sigma(X)$ satisfy the set of generalized Newton's identities given by (7.15).

Suppose we have just completed the μ th step and found the solution $\sigma^{(\mu)}(X)$. To find $\sigma^{(\mu+1)}(X)$, we first check whether the coefficients of $\sigma^{(\mu)}(X)$ satisfy the next generalized Newton's identity; that is,

$$S_{\mu+1} + \sigma_1^{(\mu)} S_\mu + \cdots + \sigma_{l_\mu}^{(\mu)} S_{\mu+1-l_\mu} \stackrel{?}{=} 0 \tag{7.18}$$

If yes, $\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X)$ is the minimum-degree polynomial whose coefficients satisfy the generalized Newton's identities of (7.17). If not, we add a correction term to $\sigma^{(\mu)}(X)$ so that its coefficients satisfy the set of generalized Newton's identities of (7.17). To test the equality of (7.18), we compute the discrepancy,

$$d_\mu = S_{\mu+1} + \sigma_1^{(\mu)} S_\mu + \cdots + \sigma_{l_\mu}^{(\mu)} S_{\mu+1-l_\mu}. \tag{7.19}$$

If $d_\mu = 0$, we set

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X).$$

If $d_\mu \neq 0$, we need to adjust $\sigma^{(\mu)}(X)$ to satisfy the equalities of (7.17). We make the correction as follows: we go back to the steps prior to the μ th step and determine a

TABLE 7.1: Berlekamp's iterative procedure for finding the error-location polynomial of a q -ary BCH code.

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	S_1	0	0
1	$1 - S_1 X$			
2				
3				
:				
$2t$				

polynomial $\sigma^{(\rho)}(X)$ such that $d_\rho \neq 0$ and $\rho - l_\rho$ has the largest value, where l_ρ is the degree of $\sigma^{(\rho)}(X)$. Then,

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X) - d_\mu d_\rho^{-1} X^{(\mu-\rho)} \sigma^{(\rho)}(X), \quad (7.20)$$

and $\sigma^{(\mu+1)}(X)$ is the solution at the $(\mu + 1)$ th step of the iteration process.

As with binary BCH codes, to find $\sigma(X)$, we fill out Table 7.1 (reproduction of Table 6.5).

We can determine the roots of $\sigma(X)$ in $GF(q^m)$ by substituting the elements of $GF(q^m)$ into $\sigma(X)$ cyclically. If $\sigma(\alpha^i) = 0$, then α^i is a root of $\sigma(X)$, and

$$\alpha^{-i} = \alpha^{q^m-1-i}$$

is an error-location number. We can do this systematically with Chien's search.

EXAMPLE 7.2

Consider a triple-error-correcting RS code with symbols from $GF(2^4)$. The generator polynomial of this code is

$$\begin{aligned} g(X) &= (X + \alpha)(X + \alpha^2)(X + \alpha^3)(X + \alpha^4)(X + \alpha^5)(X + \alpha^6) \\ &= \alpha^6 + \alpha^9 X + \alpha^6 X^2 + \alpha^4 X^3 + \alpha^{14} X^4 + \alpha^{10} X^5 + X^6. \end{aligned}$$

Let the all-zero vector be the transmitted codeword, and let $\mathbf{r} = (0\ 0\ 0\ \alpha^7\ 0\ 0\ \alpha^3\ 0\ 0\ 0\ 0\ \alpha^4\ 0\ 0)$ be the received vector. Thus, $\mathbf{r}(X) = \alpha^7 X^3 + \alpha^3 X^6 + \alpha^4 X^{12}$.

Step 1. We compute the syndrome components as follows (using Table 2.8):

$$S_1 = \mathbf{r}(\alpha) = \alpha^{10} + \alpha^9 + \alpha = \alpha^{12},$$

$$S_2 = \mathbf{r}(\alpha^2) = \alpha^{13} + 1 + \alpha^{13} = 1,$$

$$S_3 = \mathbf{r}(\alpha^3) = \alpha + \alpha^6 + \alpha^{10} = \alpha^{14},$$

$$S_4 = \mathbf{r}(\alpha^4) = \alpha^4 + \alpha^{12} + \alpha^7 = \alpha^{10},$$

$$S_5 = \mathbf{r}(\alpha^5) = \alpha^7 + \alpha^3 + \alpha^4 = 0,$$

$$S_6 = \mathbf{r}(\alpha^6) = \alpha^{10} + \alpha^9 + \alpha = \alpha^{12}.$$

TABLE 7.2: Steps for finding the error-location polynomial of the (15,9) RS code over $GF(2^4)$.

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	α^{12}	0	0
1	$1 + \alpha^{12}X$	α^7	1	0 (take $\rho = -1$)
2	$1 + \alpha^3X$	1	1	1 (take $\rho = 0$)
3	$1 + \alpha^3X + \alpha^3X^2$	α^7	2	1 (take $\rho = 0$)
4	$1 + \alpha^4X + \alpha^{12}X^2$	α^{10}	2	2 (take $\rho = 2$)
5	$1 + \alpha^7X + \alpha^4X^2 + \alpha^6X^3$	0	3	2 (take $\rho = 3$)
6	$1 + \alpha^7X + \alpha^4X^2 + \alpha^6X^3$	—	—	—

Step 2. To find the error-location polynomial $\sigma(X)$, we fill out Table 7.1 and obtain Table 7.2. Thus, $\sigma(X) = 1 + \alpha^7X + \alpha^4X^2 + \alpha^6X^3$.

Step 3. By substituting $1, \alpha, \alpha^2, \dots, \alpha^{14}$ into $\sigma(X)$, we find that α^3, α^9 , and α^{12} are roots of $\sigma(X)$. The reciprocals of these roots are α^{12}, α^6 , and α^3 , which are the error-location numbers of the error pattern $\mathbf{e}(X)$. Thus, errors occur at positions X^3, X^6 , and X^{12} .

Next, we need to determine the error values, by finding the error-value evaluator. We define the syndrome polynomial $\mathbf{S}(X)$ as follows:

$$\begin{aligned}\mathbf{S}(X) &\stackrel{\Delta}{=} S_1 + S_2X + \dots + S_{2t}X^{2t-1} + S_{2t+1}X^{2t} + \dots \\ &= \sum_{j=1}^{\infty} S_j X^{j-1}. \end{aligned}\tag{7.21}$$

Note that only the coefficients of the first $2t$ terms are known. For $1 \leq j < \infty$, we also define

$$S_j = \sum_{l=1}^v \delta_l \beta_l^j. \tag{7.22}$$

The first $2t$ such S_j 's are simply the $2t$ equalities of (7.13). Combining (7.21) and (7.22), we can put $\mathbf{S}(X)$ in the following form:

$$\begin{aligned}\mathbf{S}(X) &= \sum_{j=1}^{\infty} X^{j-1} \sum_{l=1}^v \delta_l \beta_l^j \\ &= \sum_{l=1}^v \delta_l \beta_l \sum_{j=1}^{\infty} (\beta_l X)^{j-1}. \end{aligned}\tag{7.23}$$

Note that

$$\frac{1}{(1 - \beta_l X)} = \sum_{j=1}^{\infty} (\beta_l X)^{j-1}. \tag{7.24}$$

It follows from (7.23) and (7.24) that

$$\mathbf{S}(X) = \sum_{l=1}^v \frac{\delta_l \beta_l}{1 - \beta_l X}. \quad (7.25)$$

Consider the product $\sigma(X)\mathbf{S}(X)$,

$$\begin{aligned} \sigma(X)\mathbf{S}(X) &= (1 + \sigma_1 X + \cdots + \sigma_v X^v) \cdot (S_1 + S_2 X + S_3 X^2 + \cdots) \\ &= S_1 + (S_2 + \sigma_1 S_1)X + (S_3 + \sigma_1 S_2 + \sigma_2 S_1)X^2 + \cdots + \\ &\quad (S_{2t} + \sigma_1 S_{2t-1} + \cdots + \sigma_v S_{2t-v})X^{2t-1} + \cdots \end{aligned} \quad (7.26)$$

Using (7.25), we can also put $\sigma(X)\mathbf{S}(X)$ in the following form:

$$\begin{aligned} \sigma(X)\mathbf{S}(X) &= \left\{ \prod_{i=1}^v (1 - \beta_i X) \right\} \cdot \left\{ \sum_{l=1}^v \frac{\delta_l \beta_l}{1 - \beta_l X} \right\} \\ &= \sum_{l=1}^v \frac{\delta_l \beta_l}{1 - \beta_l X} \cdot \prod_{i=1}^v (1 - \beta_i X) \\ &= \sum_{l=1}^v \delta_l \beta_l \prod_{i=1, i \neq l}^v (1 - \beta_i X). \end{aligned} \quad (7.27)$$

We define

$$\mathbf{Z}_0(X) \stackrel{\Delta}{=} \sum_{l=1}^v \delta_l \beta_l \prod_{i=1, i \neq l}^v (1 - \beta_i X). \quad (7.28)$$

Note that $\mathbf{Z}_0(X)$ is a polynomial of degree $v - 1$. From (7.26), (7.27), and (7.28), we see that $\mathbf{Z}_0(X)$ must be equal to the first v terms from X^0 to X^{v-1} in $\sigma(X)\mathbf{S}(X)$ of (7.26); that is,

$$\begin{aligned} \mathbf{Z}_0(X) &= S_1 + (S_2 + \sigma_1 S_1)X + (S_3 + \sigma_1 S_2 + \sigma_2 S_1)X^2 \\ &\quad + \cdots + (S_v + \sigma_1 S_{v-1} + \cdots + \sigma_{v-1} S_1)X^{v-1}. \end{aligned} \quad (7.29)$$

Because the degree of $\mathbf{Z}_0(X)$ is $v - 1$, the coefficients of powers from X^v to X^{2t-1} in the expansion of $\sigma(X)\mathbf{S}(X)$ ((7.26)) must be zero. Setting these coefficients to zero, we have exactly the same set of equations as (7.15).

Next, we show that the error values can be determined from $\mathbf{Z}_0(X)$ and $\sigma(X)$. Substituting β_k^{-1} in $\mathbf{Z}_0(X)$ (given by (7.28)), we have

$$\begin{aligned} \mathbf{Z}_0(\beta_k^{-1}) &= \sum_{l=1}^v \delta_l \beta_l \prod_{i=1, i \neq l}^v (1 - \beta_i \beta_k^{-1}) \\ &= \delta_k \beta_k \prod_{i=1, i \neq k}^v (1 - \beta_i \beta_k^{-1}). \end{aligned} \quad (7.30)$$

We take the derivative of $\sigma(X)$ in (7.14),

$$\begin{aligned}\sigma'(X) &= \frac{d}{dX} \prod_{i=1}^v (1 - \beta_i X) \\ &= - \sum_{l=1}^v \beta_l \prod_{i=1, i \neq l}^v (1 - \beta_i X).\end{aligned}\quad (7.31)$$

Then,

$$\sigma'(\beta_k^{-1}) = -\beta_k \prod_{i=1, i \neq k}^v (1 - \beta_i \beta_k^{-1}). \quad (7.32)$$

From (7.30) and (7.32), we find that the error value δ_k at location β_k is given by

$$\delta_k = \frac{-\mathbf{Z}_0(\beta_k^{-1})}{\sigma'(\beta_k^{-1})}. \quad (7.33)$$

This expression was derived by Forney [4]. The polynomial $\mathbf{Z}_0(X)$ is called the *error-value evaluator*.

A slightly different error-value evaluator is defined as

$$\begin{aligned}\mathbf{Z}(X) &\stackrel{\Delta}{=} \sigma(X) + X\mathbf{Z}_0(X) \\ &= 1 + (S_1 + \sigma_1)X + (S_2 + \sigma_1 S_1 + \sigma_2)X^2 \\ &\quad + \cdots + (S_v + \sigma_1 S_{v-1} + \cdots + \sigma_v)X^v.\end{aligned}\quad (7.34)$$

Then,

$$\delta_k = \frac{-\mathbf{Z}(\beta_k^{-1})}{\prod_{i=1, i \neq k}^v (1 - \beta_i \beta_k^{-1})}. \quad (7.35)$$

The expression for evaluating δ_k was derived by Berlekamp [5].

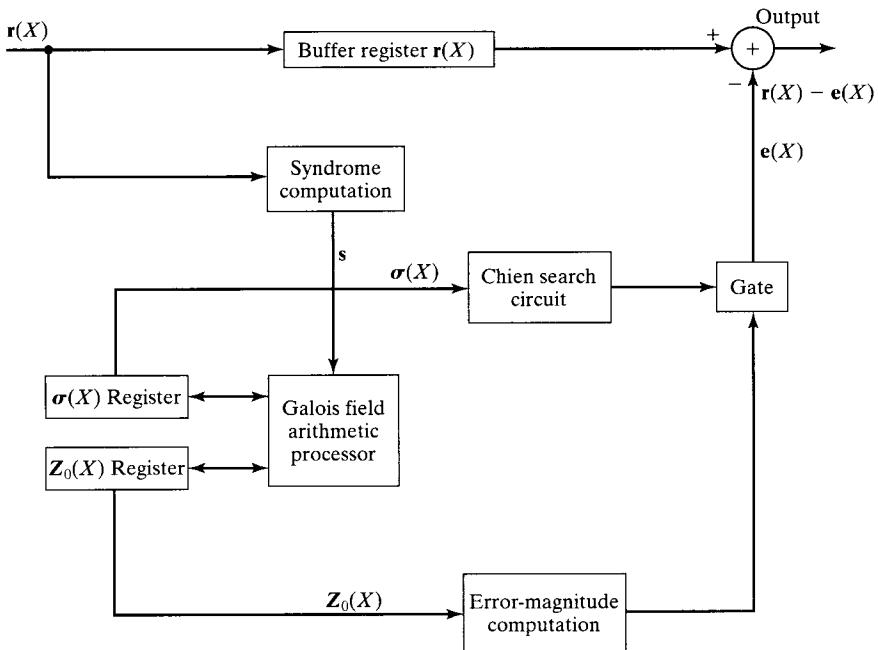
EXAMPLE 7.3

Consider the triple-error-correcting RS code of length 15 given in Example 7.2, where we assumed that the all-zero codeword was transmitted, and $\mathbf{r} = (000\alpha^7 00\alpha^3 00000\alpha^4 00)$ was received. Using the Berlekamp algorithm, we find that

$$\sigma(X) = 1 + \alpha^7 X + \alpha^4 X^2 + \alpha^6 X^3,$$

and error-location numbers are α^{12} , α^6 , and α^3 . The errors occur at X^3 , X^6 , and X^{12} . Now, we are ready to evaluate the error values at these positions. From (7.29) we find that

$$\begin{aligned}\mathbf{Z}_0(X) &= S_1 + (S_2 + \sigma_1 S_1)X + (S_3 + \sigma_1 S_2 + \sigma_2 S_1)X^2 \\ &= \alpha^{12} + (1 + \alpha^7 \alpha^{12})X + (\alpha^{14} + \alpha^7 + \alpha^4 \alpha^{12})X^2 \\ &= \alpha^{12} + (1 + \alpha^4)X + (\alpha^{14} + \alpha^7 + \alpha)X^2 \\ &= \alpha^{12} + \alpha X.\end{aligned}$$

FIGURE 7.2: A general organization of a q -ary BCH decoder.

(The computations are carried out in $GF(2^4)$ given by Table 2.8.) Using (7.33), we obtain the error values at locations X^3 , X^6 , and X^{12} :

$$\begin{aligned} e_3 &= \frac{-\mathbf{Z}_0(\alpha^{-3})}{\sigma'(\alpha^{-3})} = \frac{\alpha^{12} + \alpha\alpha^{-3}}{\alpha^3(1 + \alpha^6\alpha^{-3})(1 + \alpha^{12}\alpha^{-3})} = \frac{\alpha}{\alpha^9} = \alpha^7, \\ e_6 &= \frac{-\mathbf{Z}_0(\alpha^{-6})}{\sigma'(\alpha^{-6})} = \frac{\alpha^{12} + \alpha\alpha^{-6}}{\alpha^6(1 + \alpha^3\alpha^{-6})(1 + \alpha^{12}\alpha^{-6})} = \frac{\alpha^3}{1} = \alpha^3, \\ e_{12} &= \frac{-\mathbf{Z}_0(\alpha^{-12})}{\sigma'(\alpha^{-12})} = \frac{\alpha^{12} + \alpha\alpha^{-12}}{\alpha^{12}(1 + \alpha^3\alpha^{-12})(1 + \alpha^6\alpha^{-12})} = \frac{\alpha^6}{\alpha^2} = \alpha^4. \end{aligned}$$

Thus, the error pattern is

$$\mathbf{e}(X) = \alpha^7 X^3 + \alpha^3 X^6 + \alpha^4 X^{12},$$

which is exactly the difference between the received vector and the transmitted vector. The decoding is completed by taking $\mathbf{r}(X) - \mathbf{e}(X)$.

A general organization of a BCH decoder is shown in Figure 7.2.

7.5 DECODING WITH THE EUCLIDEAN ALGORITHM

In the expansion of $\sigma(X)\mathbf{S}(X)$ ((7.26)), only the coefficients of the first $2t$ terms (X^0 to X^{2t-1}) are known. Let

$$[\sigma(X)\mathbf{S}(X)]_{2t}$$

denote the first $2t$ terms of $\sigma(X)\mathbf{S}(X)$. Then,

$$\sigma(X)\mathbf{S}(X) = [\sigma(X)\mathbf{S}(X)]_{2t}$$

is divisible by X^{2t} . This simply says that if $\sigma(X)\mathbf{S}(X)$ is divided by X^{2t} , the remainder is $[\sigma(X)\mathbf{S}(X)]_{2t}$. Mathematically, this statement is expressed as follows:

$$\sigma(X)\mathbf{S}(X) \equiv [\sigma(X)\mathbf{S}(X)]_{2t} \pmod{X^{2t}}. \quad (7.36)$$

In fact,

$$\mathbf{Z}_0(X) = [\sigma(X)\mathbf{S}(X)]_{2t}. \quad (7.37)$$

Therefore, we have

$$\sigma(X)\mathbf{S}(X) \equiv \mathbf{Z}_0(X) \pmod{X^{2t}} \quad (7.38)$$

which is called the *key equation* in decoding of BCH codes [5].

Any method of solving the key equation to find $\sigma(X)$ and $\mathbf{Z}_0(X)$ is a decoding method for q -ary BCH codes. If the number of errors v during the transmission of a code polynomial $\mathbf{v}(X)$ is less than or equal to t , (i.e., $v \leq t$), then the key equation has a unique pair of solutions, $(\sigma(X), \mathbf{Z}_0(X))$, with

$$\deg \mathbf{Z}_0(X) < \deg \sigma(X) \leq t, \quad (7.39)$$

where $\deg f(X)$ denotes the degree of polynomial $f(X)$. We have already presented Berlekamp's algorithm for solving the key equation, which is a very effective method for practical implementation and has been widely used.

There is another method for solving the key equation that is much easier to understand. This method is based on the Euclidean algorithm for finding the greatest common divisor (GCD) of two polynomials.

Consider two polynomials, $a(X)$ and $b(X)$, over $GF(q)$. Assume that

$$\deg a(X) \geq \deg b(X).$$

Let $\text{GCD}[a(X), b(X)]$ denote the greatest common divisor of $a(X)$ and $b(X)$. Then, we can find $\text{GCD}[a(X), b(X)]$ by iteratively applying the division algorithm as follows:

$$\begin{aligned} a(X) &= q_1(X)b(X) + r_1(X) \\ b(X) &= q_2(X)r_1(X) + r_2(X) \\ r_1(X) &= q_3(X)r_2(X) + r_3(X) \\ &\vdots \\ r_{i-2}(X) &= q_i(X)r_{i-1}(X) + r_i(X) \\ &\vdots \\ r_{n-2}(X) &= q_n(X)r_{n-1}(X) + r_n(X) \\ r_{n-1}(X) &= q_{n+1}(X)r_n(X), \end{aligned} \quad (7.40)$$

where $q_i(X)$ and $r_i(X)$ are the quotient and the remainder, respectively, at the i th step of the iterative division. The iteration stops when the remainder is identical to zero. Then, the last nonzero remainder $r_n(X)$ is the GCD of $a(X)$ and $b(X)$ (may be different by a constant scalar c); that is,

$$r_n(X) = c \text{GCD}[a(X), b(X)],$$

where $c \in GF(q)$. Note that for $1 \leq i \leq n$,

$$\deg r_{i-1}(X) > \deg r_i(X).$$

From (7.40), it is possible to show that

$$\text{GCD}[a(X), b(X)] = f(X)a(X) + g(X)b(X), \quad (7.41)$$

where $f(X)$ and $g(X)$ are polynomials over $GF(q)$ [Euclid's algorithm].

In fact, we can express the remainder at each division step as follows:

$$\begin{aligned} r_1(X) &= f_1(X)a(X) + g_1(X)b(X) \\ r_2(X) &= f_2(X)a(X) + g_2(X)b(X) \\ &\vdots \\ r_i(X) &= f_i(X)a(X) + g_i(X)b(X) \\ &\vdots \\ r_n(X) &= f_n(X)a(X) + g_n(X)b(X). \end{aligned} \quad (7.42)$$

From (7.41) and (7.42), we have

$$\begin{aligned} f(X) &= c^{-1}f_n(X) \\ g(X) &= c^{-1}g_n(X). \end{aligned} \quad (7.43)$$

From (7.40) and (7.42), we obtain the following recursive equations for finding $r_i(X)$, $f_i(X)$, and $g_i(X)$:

$$\begin{aligned} r_i(X) &= r_{i-2}(X) - q_i(X)r_{i-1}(X) \\ f_i(X) &= f_{i-2}(X) - q_i(X)f_{i-1}(X) \\ g_i(X) &= g_{i-2}(X) - q_i(X)g_{i-1}(X) \end{aligned} \quad (7.44)$$

for $1 \leq i \leq n$. The initial conditions for the recursion are

$$\begin{aligned} r_{-1}(X) &= a(X), \\ r_0(X) &= b(X), \\ f_{-1}(X) &= g_0(X) = 1, \\ f_0(X) &= g_{-1}(X) = 0. \end{aligned} \quad (7.45)$$

TABLE 7.3: Steps for finding the GCD of $X^3 + 1$
 $X^2 + 1$ given in Example 7.4.

i	$r_i(X)$	$q_i(X)$	$f_i(X)$	$g_i(X)$
-1	$X^3 + 1$	—	1	0
0	$X^2 + 1$	—	0	1
1	$X + 1$	X	1	X
2	0	$X + 1$	$X + 1$	$X^2 + X + 1$

An important property of Euclid's algorithm is

$$\deg a(X) = \deg g_i(X) + \deg r_{i-1}(X). \quad (7.46)$$

We see that as i increases, the degree of $r_{i-1}(X)$ decreases, and the degree of $g_i(X)$ increases. This result will be used for solving the key equation.

EXAMPLE 7.4

Let $a(X) = X^3 + 1$ and $b(X) = X^2 + 1$ be two polynomials over $GF(2)$. Euclid's algorithm for finding the $\text{GCD}[X^3 + 1, X^2 + 1]$ is shown in Table 7.3. We see that last nonzero remainder is

$$r_1(X) = X + 1,$$

which is the GCD of $X^3 + 1$ and $X^2 + 1$.

7.5.1 Solving the Key Equation [6]

We can express the key equation in the following form:

$$\sigma(X)\mathbf{S}(X) = \mathbf{Q}(X)X^{2t} + \mathbf{Z}_0(X). \quad (7.47)$$

Rearranging (7.47), we have

$$\mathbf{Z}_0(X) = -\mathbf{Q}(X)X^{2t} + \sigma(X)\mathbf{S}(X). \quad (7.48)$$

Setting $a(X) = X^{2t}$ and $b(X) = \mathbf{S}(X)$, we see that (7.48) is exactly in the form given by (7.41). This suggests that $\sigma(X)$ and $\mathbf{Z}_0(X)$ can be found by the Euclidean iterative division algorithm for the two polynomials:

$$\begin{aligned} a(X) &= X^{2t}, \\ b(X) &= \mathbf{S}(X), \end{aligned} \quad (7.49)$$

where

$$\mathbf{S}(X) = S_1 + S_2X + S_3X^2 + \cdots + S_{2t}X^{2t-1}.$$

Let

$$\begin{aligned} \mathbf{Z}_0^{(i)}(X) &= r_i(X) \\ \sigma^{(i)}(X) &= g_i(X) \\ \gamma^{(i)}(X) &= f_i(X). \end{aligned} \quad (7.50)$$

Then, it follows from (7.49) and (7.50) that we can put (7.42), (7.44), and (7.45) in the following forms:

$$\mathbf{Z}_0^{(i)}(X) = \gamma^{(i)}(X)X^{2t} + \sigma^{(i)}(X)\mathbf{S}(X), \quad (7.51)$$

and

$$\begin{aligned}\mathbf{Z}_0^{(i)}(X) &= \mathbf{Z}_0^{(i-2)}(X) - q_i(X)\mathbf{Z}_0^{(i-1)}(X), \\ \sigma^{(i)}(X) &= \sigma^{(i-2)}(X) - q_i(X)\sigma^{(i-1)}(X), \\ \gamma^{(i)}(X) &= \gamma^{(i-2)}(X) - q_i(X)\gamma^{(i-1)}(X),\end{aligned}\quad (7.52)$$

with

$$\begin{aligned}\mathbf{Z}_0^{(-1)}(X) &= X^{2t}, \\ \mathbf{Z}_0^{(0)}(X) &= \mathbf{S}(X), \\ \gamma^{(-1)}(X) &= \sigma^{(0)}(X) = 1, \\ \gamma^{(0)}(X) &= \sigma^{(-1)}(X) = 0.\end{aligned}$$

To find $\sigma(X)$ and $\mathbf{Z}_0(X)$, we carry out the iteration process given by (7.52) as follows: at the i th step,

1. We divide $\mathbf{Z}_0^{(i-2)}(X)$ by $\mathbf{Z}_0^{(i-1)}(X)$ to obtain the quotient $q_i(X)$ and the remainder $\mathbf{Z}_0^{(i)}(X)$.
2. We find $\sigma^{(i)}(X)$ from

$$\sigma^{(i)}(X) = \sigma^{(i-2)}(X) - q_i(X)\sigma^{(i-1)}(X).$$

Iteration stops when we reach a step ρ for which

$$\deg \mathbf{Z}_0^{(\rho)}(X) < \deg \sigma^{(\rho)}(X) \leq t. \quad (7.53)$$

Then,

$$\begin{aligned}\mathbf{Z}_0(X) &= \mathbf{Z}_0^{(\rho)}(X), \\ \sigma(X) &= \sigma^{(\rho)}(X).\end{aligned}$$

If the number of errors is t or less, there always exists a step ρ for which the condition given by (7.53) holds. It is easy to see that

$$\rho \leq 2t.$$

Note that $\sigma(X) = \sigma^{(\rho)}(X)$ found by the foregoing algorithm may be different from $\sigma(X)$ defined by (7.14) by a constant scalar in $GF(q^m)$; however, it gives the same roots.

The iteration process for finding $\sigma(X)$ and $\mathbf{Z}_0(X)$ can be carried out by setting up and filling Table 7.4.

TABLE 7.4: Euclidean's iterative procedure for finding the error-location polynomial and error-value evaluator.

i	$\mathbf{Z}_0^{(i)}(X)$	$q_i(X)$	$\sigma_i(X)$
-1	X^{2t}	—	0
0	$\mathbf{S}(X)$	—	1
1			
2			
⋮			
i			
⋮			

EXAMPLE 7.5

Consider the triple-error-correcting RS code of length $n = 15$ over $GF(2^4)$ whose generator polynomial has $\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5$, and α^6 as roots; that is,

$$\mathbf{g}(X) = (X + \alpha)(X + \alpha^2)((X + \alpha^3)(X + \alpha^4)(X + \alpha^5)(X + \alpha^6)).$$

Suppose the received polynomial is

$$\mathbf{r}(X) = \alpha^7 X^3 + \alpha^{11} X^{10}.$$

The syndrome components are

$$\begin{aligned} S_1 &= \mathbf{r}(\alpha) = \alpha^{10} + \alpha^{21} = \alpha^7, \\ S_2 &= \mathbf{r}(\alpha^2) = \alpha^{13} + \alpha^{31} = \alpha^{12}, \\ S_3 &= \mathbf{r}(\alpha^3) = \alpha^{16} + \alpha^{41} = \alpha^6, \\ S_4 &= \mathbf{r}(\alpha^4) = \alpha^{19} + \alpha^{51} = \alpha^{12}, \\ S_5 &= \mathbf{r}(\alpha^5) = \alpha^7 + \alpha = \alpha^{14}, \\ S_6 &= \mathbf{r}(\alpha^6) = \alpha^{10} + \alpha^{11} = \alpha^{14}. \end{aligned}$$

Hence, the syndrome polynomial is

$$\mathbf{S}(X) = \alpha^7 + \alpha^{12}X + \alpha^6X^2 + \alpha^{12}X^3 + \alpha^{14}X^4 + \alpha^{14}X^5.$$

Using the Euclidean algorithm, we find

$$\begin{aligned} \sigma(X) &= \alpha^{11} + \alpha^8X + \alpha^9X^2 \\ &= \alpha^{11}(1 + \alpha^{12}X + \alpha^{13}X^2), \end{aligned}$$

and

$$\mathbf{Z}_0(X) = \alpha^3 + \alpha^2X,$$

as shown in the Table 7.5.

TABLE 7.5: Steps for finding the error-location polynomial and error-value evaluator of the RS code given in Example 7.5.

i	$Z_0^{(i)}(X)$	$q_i(X)$	$\sigma_i(X)$
-1	X^6	—	0
0	$S(X)$	—	1
1	$\alpha^8 + \alpha^3 X + \alpha^5 X^2 + \alpha^5 X^3 + \alpha^6 X^4$	$\alpha + \alpha X$	$\alpha + \alpha X$
2	$\alpha^3 + \alpha^2 X$	$\alpha^{11} + \alpha^8 X$	$\alpha^{11} + \alpha^8 X + \alpha^9 X^2$

From $\sigma(X)$, we find that the roots are α^5 and α^{12} . Hence, the error location numbers are α^{10} and α^3 . The error values at these locations are

$$e_3 = \frac{-Z_0(\alpha^{-3})}{\sigma'(\alpha^{-3})} = \frac{\alpha^3 + \alpha^2 \alpha^{-3}}{\alpha^{11} \cdot \alpha^3 (1 + \alpha^{10} \alpha^{-3})} = \frac{1}{\alpha^8} = \alpha^7,$$

$$e_{10} = \frac{-Z_0(\alpha^{-10})}{\sigma'(\alpha^{-10})} = \frac{\alpha^3 + \alpha^2 \cdot \alpha^{-10}}{\alpha^{11} \cdot \alpha^{10} (1 + \alpha^3 \alpha^{-10})} = \frac{\alpha^4}{\alpha^8} = \alpha^{11}.$$

Therefore, the error polynomial is $e(X) = \alpha^7 X^3 + \alpha^{11} X^{10}$, and the decoded codeword $v(X) = r(X) - e(X)$ is the all-zero codeword.

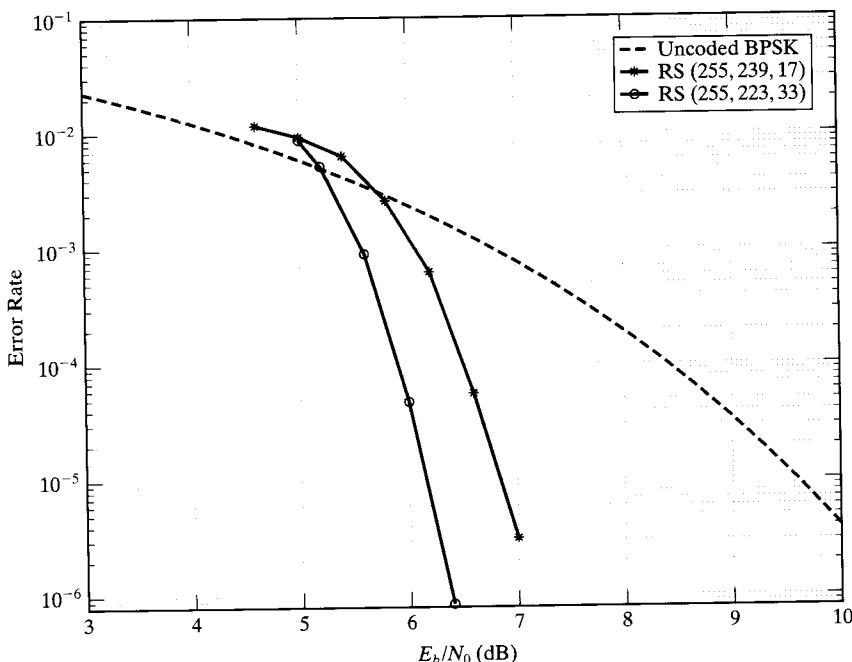


FIGURE 7.3: Error performances for (255, 223, 33) and (255, 239, 17) RS codes.

The two most commonly used RS codes for error control in data communication and storage systems are the (255, 223, 33) and the (255, 239, 17) RS codes over $GF(2^8)$. The (255, 223, 33) RS code is also a NASA standard code for space and satellite communications. Both codes are decoded with either the Berlekamp algorithm or the Euclidean algorithm. Their error performances with BPSK transmission over the AWGN channel are shown in Figure 7.3.

7.6 FREQUENCY-DOMAIN DECODING

So far, BCH and RS codes have been decoded in the time domain; however, these codes also can be decoded in the frequency domain. In this section, we first give a spectral description of these codes and then present a frequency-domain decoding algorithm for them [8, 17].

Consider the Galois field $GF(q)$ with characteristic p (see Section 2.2). Let 1 be the unit element of $GF(q)$. Then, p is the smallest positive integer such that the sum

$$\sum_{i=1}^p 1 = \underbrace{1 + 1 + \dots + 1}_p = 0.$$

For any nonnegative integer n , the sum

$$\sum_{i=1}^n 1 = \underbrace{1 + 1 + \dots + 1}_n = \lambda,$$

where λ is the remainder resulting from dividing n by p . Mathematically, we write

$$\lambda = n \text{(modulo } p\text{)}. \quad (7.54)$$

Note that λ is an element in $GF(q)$.

Let $\mathbf{v}(X) = v_0 + v_1 X + \dots + v_{n-1} X^{n-1}$ be a polynomial over $GF(q)$, where n divides $q^m - 1$, and $n \neq 1$. Let α be an element of order n in $GF(q^m)$. Then, $\alpha^n = 1$ and is a root of $X^n - 1$. The Galois field Fourier transform of $\mathbf{v}(X)$ is defined as the polynomial

$$\mathbf{V}(X) = V_0 + V_1 X + \dots + V_{n-1} X^{n-1} \quad (7.55)$$

over $GF(q^m)$, where for $0 \leq j < n$,

$$V_j = \mathbf{v}(\alpha^j) = \sum_{i=0}^{n-1} v_i \alpha^{ij}. \quad (7.56)$$

The coefficient V_j is called the j th *spectral component* of $\mathbf{V}(X)$.

Given $\mathbf{V}(X)$, the polynomial $\mathbf{v}(X)$ can be determined by taking the inverse Fourier transform of $\mathbf{V}(X)$, as shown in Theorem 7.1.

THEOREM 7.1 Let $\mathbf{V}(X) = V_0 + V_1X + \dots + V_{n-1}X^{n-1}$ be the Galois field Fourier transform of $\mathbf{v}(X) = v_0 + v_1X + \dots + v_{n-1}X^{n-1}$. Then,

$$\begin{aligned} v_i &= \frac{1}{n(\text{modulo } p)} \mathbf{V}(\alpha^{-i}) \\ &= \frac{1}{n(\text{modulo } p)} \sum_{j=0}^{n-1} V_j \alpha^{-ij}. \end{aligned} \quad (7.57)$$

Proof. Factor $X^n - 1$ as

$$X^n - 1 = (X - 1)(X^{n-1} + X^{n-2} + \dots + X + 1).$$

Because α is a root of $X^n - 1$, and $\alpha \neq 1$, α must be a root of $X^{n-1} + X^{n-2} + \dots + X + 1$; that is,

$$1 + \alpha + \dots + \alpha^{n-2} + \alpha^{n-1} = 0.$$

For any integer r that is not a multiple of n (i.e., $r \neq 0$ (modulo p)), $(\alpha^r)^n = 1$, and hence α^r is also a root of $X^n - 1$. Since $\alpha^r \neq 1$, α^r must be a root of $X^{n-1} + X^{n-2} + \dots + X + 1$, and hence

$$\sum_{j=0}^{n-1} \alpha^{rj} = 0. \quad (7.58)$$

Now, consider $\mathbf{V}(\alpha^{-i})$. It follows from (7.55) and (7.56) that

$$\begin{aligned} \mathbf{V}(\alpha^{-i}) &= \sum_{j=0}^{n-1} \alpha^{-ij} \sum_{l=0}^{n-1} v_l \alpha^{lj} \\ &= \sum_{l=0}^{n-1} v_l \sum_{j=0}^{n-1} \alpha^{(l-i)j}. \end{aligned} \quad (7.59)$$

It follows from (7.58) that for $l \neq i$, the second sum on the right side of (7.59) is equal to 0. For $l = i$, the second sum becomes $1 + 1 + \dots + 1 = n$ (modulo p). Consequently, (7.59) becomes

$$\mathbf{V}(\alpha^{-i}) = v_i \cdot n(\text{modulo } p). \quad (7.60)$$

Note that p divides q and does not divide $q^m - 1$. Because n is a factor of $q^m - 1$, then n and p are relatively prime. Therefore, $n(\text{modulo } p) \neq 0$. It then follows from (7.60) that

$$v_i = \frac{1}{n(\text{modulo } p)} \mathbf{V}(\alpha^{-i}).$$

This completes the proof. Q.E.D.

The polynomials $\mathbf{v}(X)$ and $\mathbf{V}(X)$ form a *transform pair*. $\mathbf{V}(X)$ is the *spectrum polynomial* (or simply spectrum) of $\mathbf{v}(X)$. From (7.56) and (7.57), we readily see that the transform pair have the following properties:

1. The j th spectral component V_j is zero if and only if α^j is a root of $\mathbf{v}(X)$.
2. The i th component of $\mathbf{v}(X)$ is zero if only if α^{-i} is a root of $\mathbf{V}(X)$.

Now, we are ready to characterize BCH and RS codes in the frequency domain. Consider a primitive q -ary t -error-correcting BCH code of length $n = q^m - 1$ whose generator polynomial $\mathbf{g}(X)$ has $\alpha, \alpha^2, \dots, \alpha^{2t}$ as roots. Recall that a polynomial $\mathbf{v}(X)$ of degree $n - 1$ or less over $GF(q)$ is a code polynomial if and only if $\mathbf{v}(X)$ is divisible by $\mathbf{g}(X)$. This is equivalent to saying that $\mathbf{v}(X)$ is a code polynomial if and only if $\mathbf{v}(X)$ has $\alpha, \alpha^2, \dots, \alpha^{2t}$ as roots. Let $\mathbf{v}(X) = v_0 + v_1 X + \dots + v_{n-1} X^{n-1}$ be a code polynomial in a primitive q -ary t -error-correcting BCH code of length $n = q^m - 1$, and let $\mathbf{V}(X) = V_0 + V_1 X + \dots + V_{n-1} X^{n-1}$ be its Fourier transform. It follows from the first property of the transform pair $(\mathbf{v}(X), \mathbf{V}(X))$ that the $2t$ consecutive spectral components of $\mathbf{V}(X)$ from position X to position X^{2t} are zero; that is, $V_1 = V_2 = \dots = V_{2t} = 0$. Consequently, a primitive q -ary t -error-correcting BCH code of length $n = q^m - 1$ is the set of polynomials of degree $n - 1$ or less over $GF(q)$ whose Fourier transforms have $2t$ consecutive zero spectral components from position X to position X^{2t} . This description is a frequency-domain characterization of a BCH code.

For a q -ary RS code, both $\mathbf{v}(X)$ and its Fourier transform $\mathbf{V}(X)$ are polynomials over $GF(q)$. In the frequency domain, a t -error-correcting RS code with symbols from $GF(q)$ consists of all the polynomials

$$\mathbf{V}(X) = V_0 + V_1 X + \dots + V_{n-1} X^{n-1}$$

of degree of $n - 1$ or less for which

$$V_1 = V_2 = \dots = V_{2t} = 0.$$

EXAMPLE 7.6

Again, we consider the triple-error-correcting RS code of length 15 over $GF(2^4)$ with generator polynomial

$$\begin{aligned}\mathbf{g}(X) &= (X + \alpha)(X + \alpha^2)(X + \alpha^3)(X + \alpha^4)(X + \alpha^5)(X + \alpha^6) \\ &= \alpha^6 + \alpha^9 X + \alpha^6 X^2 + \alpha^4 X^3 + \alpha^{14} X^4 + \alpha^{10} X^5 + X^6.\end{aligned}$$

Substituting X with α^i for $0 \leq i < 15$ in $\mathbf{g}(X)$, we obtain the Fourier transform of $\mathbf{g}(X)$:

$$\mathbf{G}(X) = \alpha^5 + \alpha^{11} X^7 + \alpha X^8 + \alpha^{10} X^9 + \alpha^3 X^{10} + \alpha^7 X^{11} + \alpha^9 X^{12} + \alpha^7 X^{13} + \alpha^4 X^{14}.$$

(Table 2.8 for $GF(2^4)$ is used for computations.) Because $\mathbf{g}(X)$ has α to α^6 as roots, $\mathbf{G}(X)$ has zero spectral components from X^1 to X^6 ; that is, $G_1 = G_2 = \dots = G_6 = 0$. Now, consider the code polynomial

$$\mathbf{v}(X) = (X^8 + X + 1)\mathbf{g}(X), \quad (7.61)$$

which has α to α^6 and α^{10} as roots. The Fourier transform of $\mathbf{v}(X)$ is

$$\mathbf{V}(X) = \alpha^5 + \alpha^{13}X^7 + \alpha^{11}X^8 + \alpha^{12}X^9 + \alpha^8X^{11} + \alpha^{10}X^{12} + X^{13} + \alpha^8X^{14}.$$

We see that $\mathbf{V}(X)$ has seven zero spectral components at locations X^1 to X^6 and X^{10} .

Before we discuss decoding of BCH and RS codes in the frequency domain, we present an important property of Galois field Fourier transforms. Let

$$\mathbf{a}(X) = a_0 + a_1X + \dots + a_{n-1}X^{n-1},$$

$$\mathbf{b}(X) = b_0 + b_1X + \dots + b_{n-1}X^{n-1}$$

be two polynomials over $GF(q)$. We define the following product of $\mathbf{a}(X)$ and $\mathbf{b}(X)$:

$$\begin{aligned}\mathbf{c}(X) &\stackrel{\Delta}{=} \mathbf{a}(X) \cdot \mathbf{b}(X) \\ &= c_0 + c_1X + \dots + c_{n-1}X^{n-1},\end{aligned}$$

where $c_i = a_i \cdot b_i$ for $0 \leq i < n$. Let

$$\mathbf{A}(X) = A_0 + A_1X + \dots + A_{n-1}X^{n-1}$$

and

$$\mathbf{B}(X) = B_0 + B_1X + \dots + B_{n-1}X^{n-1}$$

be the Fourier transforms of $\mathbf{a}(X)$ and $\mathbf{b}(X)$, respectively. Then, the Fourier transform of the product polynomial $\mathbf{c}(X) = \mathbf{a}(X) \cdot \mathbf{b}(X)$ is the convolution of $\mathbf{A}(X)$ and $\mathbf{B}(X)$ given in Theorem 7.2.

THEOREM 7.2 The Fourier transform of $\mathbf{c}(X) = \mathbf{a}(X) \cdot \mathbf{b}(X)$ is given by

$$\mathbf{C}(X) = C_0 + C_1X + \dots + C_{n-1}X^{n-1},$$

where for $0 \leq j < n$,

$$C_j = \frac{1}{n(\text{modulo } p)} \sum_{k=0}^{n-1} A_k B_{j-k}. \quad (7.62)$$

Proof. Taking the Fourier transform of $\mathbf{c}(X)$, we have

$$C_j = \sum_{i=0}^{n-1} c_i \alpha^{ij} = \sum_{i=0}^{n-1} a_i b_i \alpha^{ij}. \quad (7.63)$$

Expressing a_i in terms of the inverse transform of $\mathbf{A}(X)$, we have

$$a_i = \frac{1}{n(\text{modulo } p)} \sum_{k=0}^{n-1} A_k \alpha^{-ik}. \quad (7.64)$$

Combining (7.63) and (7.64), we have

$$C_j = \frac{1}{n(\text{modulo } p)} \sum_{k=0}^{n-1} A_k \sum_{i=0}^{n-1} b_i \alpha^{i(j-k)}; \quad (7.65)$$

however,

$$\sum_{i=0}^{n-1} b_i \alpha^{i(j-k)} = B_{j-k}. \quad (7.66)$$

From (7.65) and (7.66), we have

$$C_j = \frac{1}{n(\text{modulo } p)} \sum_{k=0}^{n-1} A_k B_{j-k}. \quad (7.67)$$

This completes the proof. Q.E.D.

Now, we consider decoding of BCH and RS codes in the frequency domain. Let $\mathbf{r}(X) = r_0 + r_1 X + \dots + r_{n-1} X^{n-1}$ be the received polynomial, where $n = q^m - 1$. Then, $\mathbf{r}(X) = \mathbf{v}(X) + \mathbf{e}(X)$, where $\mathbf{v}(X)$ and $\mathbf{e}(X)$ are the transmitted code polynomial and the error polynomial, respectively. The Fourier transform of $\mathbf{r}(X)$ is

$$\mathbf{R}(X) = R_0 + R_1 X + \dots + R_{n-1} X^{n-1},$$

where

$$R_j = \mathbf{r}(\alpha^j) = \sum_{i=0}^{n-1} r_i \alpha^{ij}. \quad (7.68)$$

Let $\mathbf{V}(X) = V_0 + V_1 X + \dots + V_{n-1} X^{n-1}$, and $\mathbf{E}(X) = E_0 + E_1 X + \dots + E_{n-1} X^{n-1}$ be the Fourier transforms of $\mathbf{v}(X)$ and $\mathbf{e}(X)$, respectively. Then,

$$\mathbf{R}(X) = \mathbf{V}(X) + \mathbf{E}(X),$$

with

$$R_j = V_j + E_j \quad (7.69)$$

for $0 \leq j < n$. Because $\mathbf{v}(X)$ is a code polynomial that has $\alpha, \alpha^2, \dots, \alpha^{2t}$ as roots, then

$$V_j = 0 \quad (7.70)$$

for $1 \leq j \leq 2t$. From (7.69) and (7.70), we find that for $1 \leq j \leq 2t$,

$$R_j = E_j. \quad (7.71)$$

Let $\mathbf{S} = (S_1, S_2, \dots, S_{2t})$ be the syndrome of $\mathbf{r}(X)$. Then, for $1 \leq j \leq 2t$,

$$S_j = \mathbf{r}(\alpha^j). \quad (7.72)$$

It follows from (7.68), (7.71), and (7.72) that

$$R_j = E_j = S_j = \mathbf{r}(\alpha^j) \quad (7.73)$$

for $1 \leq j \leq 2t$. This result says that the $2t$ spectral components R_1, R_2, \dots, R_{2t} of $\mathbf{R}(X)$ are the $2t$ syndrome components and are equal to the $2t$ spectral components E_1, E_2, \dots, E_{2t} of the Fourier transform $\mathbf{E}(X)$ of the error polynomial $\mathbf{e}(X)$. If we can determine the spectral components $E_0, E_{2t+1}, \dots, E_{n-1}$, then $\mathbf{E}(X)$ is determined, and the inverse transform of $\mathbf{E}(X)$ gives the error polynomial $\mathbf{e}(X)$. Decoding is accomplished by subtracting $\mathbf{e}(X)$ from $\mathbf{r}(X)$.

Suppose there are $v \leq t$ errors, and

$$\mathbf{e}(X) = e_{j_1} X^{j_1} + e_{j_2} X^{j_2} + \dots + e_{j_v} X^{j_v}. \quad (7.74)$$

The error-location numbers are then $\alpha^{j_1}, \alpha^{j_2}, \dots, \alpha^{j_v}$. The error-location polynomial is

$$\begin{aligned} \sigma(X) &= (1 - \alpha^{j_1} X)(1 - \alpha^{j_2} X) \dots (1 - \alpha^{j_v} X). \\ &= \sigma_0 + \sigma_1 X + \dots + \sigma_v X^v, \end{aligned}$$

which has $\alpha^{-j_1}, \alpha^{-j_2}, \dots, \alpha^{-j_v}$ as roots. Hence, for $1 \leq i \leq v$,

$$\sigma(\alpha^{-j_i}) = 0. \quad (7.75)$$

Note that $\sigma(X)$ is a polynomial over $GF(q^m)$. We may regard $\sigma(X)$ as the Fourier transform of a polynomial

$$\lambda(X) = \lambda_0 + \lambda_1 X + \dots + \lambda_{n-1} X^{n-1}$$

over $GF(q)$, where

$$\lambda_j = \frac{1}{n(\text{modulo } p)} \sigma(\alpha^{-j}) \quad (7.76)$$

for $0 \leq j < n$. It follows from (7.75) and (7.76) that for $1 \leq i \leq v$,

$$\lambda_{j_i} = 0. \quad (7.77)$$

Consider the product $\lambda(X) \cdot \mathbf{e}(X) = \sum_{j=0}^{n-1} \lambda_j \cdot e_j X$ as defined earlier in this section. From (7.74) and (7.77), we readily see that

$$\lambda(X) \cdot \mathbf{e}(X) = 0; \quad (7.78)$$

that is, $\lambda_j \cdot e_j = 0$ for $0 \leq j < n-1$. Taking the Fourier transform of $\lambda(X) \cdot \mathbf{e}(X)$ and using (7.62; Theorem 7.2) and (7.78), we have

$$\sum_{k=0}^{n-1} \sigma_k E_{j-k} = 0 \quad (7.79)$$

for $0 \leq j < n$. Since the degree of $\sigma(X)$ is v , $\sigma_k = 0$ for $k > v$. Then, (7.79) becomes

$$\sigma_0 E_j + \sigma_1 E_{j-1} + \dots + \sigma_v E_{j-v} = 0 \quad (7.80)$$

for $0 \leq j < n - 1$. Because $\sigma_0 = 1$, the preceding equation can be put in the following form:

$$E_j = -(\sigma_1 E_{j-1} + \sigma_2 E_{j-2} + \dots + \sigma_v E_{j-v}) \quad (7.81)$$

for $0 \leq j < n$. Since E_1, E_2, \dots, E_{2t} are already known (see (7.73)), it follows from (7.81) that we obtain the following recursive equation for computing E_{2t+1} to E_{n-1} :

$$E_{l+t} = -(\sigma_1 E_{l+t-1} + \sigma_2 E_{l+t-2} + \dots + \sigma_v E_{l+t-v}) \quad (7.82)$$

for $t + 1 \leq l \leq n - 1 - t$. Setting $j = v$ in (7.81), we obtain

$$E_v = -(\sigma_1 E_{v-1} + \sigma_2 E_{v-2} + \dots + \sigma_v E_0).$$

From this equation, we find that

$$E_0 = -\frac{1}{\sigma_v}(E_v + \sigma_1 E_{v-1} + \dots + \sigma_{v-1} E_1). \quad (7.83)$$

From (7.82) and (7.83), we can determine the entire $\mathbf{E}(X)$. Taking the inverse transform of $\mathbf{V}(X) = \mathbf{R}(X) - \mathbf{E}(X)$, we obtain the decoded code polynomial $\mathbf{v}(X)$, which completes the decoding.

The error-location polynomial can be computed by using the Berlekamp iterative algorithm. The decoding consists of the following steps:

1. Take the Fourier transform $\mathbf{R}(X)$ of $\mathbf{r}(X)$.
2. Find $\sigma(X)$.
3. Compute $\mathbf{E}(X)$.
4. Take the inverse transform $\mathbf{v}(X)$ of $\mathbf{V}(X) = \mathbf{R}(X) - \mathbf{E}(X)$.

A transform decoder is depicted in Figure 7.4.

EXAMPLE 7.7

Again, consider the (15, 9) RS code over $GF(2^4)$ given in Example 7.2. Suppose a code polynomial $\mathbf{v}(X)$ is transmitted, and $\mathbf{r}(X) = \alpha^7 X^3 + \alpha^3 X^6 + \alpha^4 X^{12}$ is received. The Fourier transform of $\mathbf{r}(X)$ is

$$\begin{aligned} \mathbf{R}(X) = & \alpha^{12} X + X^2 + \alpha^{14} X^3 + \alpha^{10} X^4 + \alpha^{12} X^6 \\ & + X^7 + \alpha^{14} X^8 + \alpha^{10} X^9 + \alpha^{12} X^{11} \\ & + X^{12} + \alpha^{14} X^{13} + \alpha^{10} X^{14}. \end{aligned}$$

The coefficients of powers X to X^6 give the syndrome components; that is, $S_1 = \alpha^{12}$, $S_2 = 1$, $S_3 = \alpha^{14}$, $S_4 = \alpha^{10}$, $S_5 = 0$, and $S_6 = \alpha^{12}$. They are also the spectral components E_1, E_2, E_3, E_4, E_5 , and E_6 of the error spectral polynomial $\mathbf{E}(X)$. Using the Berlekamp algorithm based on the syndrome $(S_1, S_2, S_3, S_4, S_5, S_6)$, we find the error-location polynomial

$$\sigma(X) = 1 + \alpha^7 X + \alpha^4 X^2 + \alpha^6 X^3$$

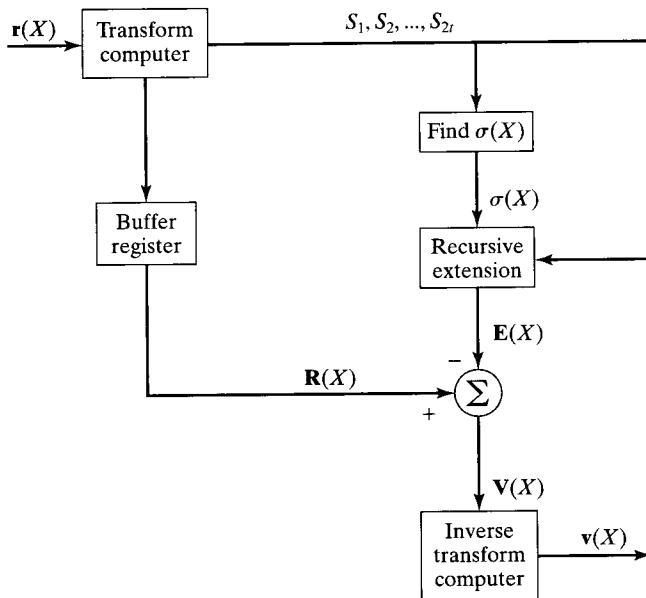


FIGURE 7.4: A transform decoder for a q -ary BCH or RS code.

(see Example 7.2). From (7.82), we obtain the following recursion equation for computing spectral components E_7 to E_{14} of $\mathbf{E}(X)$:

$$\begin{aligned} E_{l+3} &= \sigma_1 E_{l+2} + \sigma_2 E_{l+1} + \sigma_3 E_l \\ &= \alpha^7 E_{l+2} + \alpha^4 E_{l+1} + \alpha^6 E_l \end{aligned}$$

for $4 \leq l \leq 11$. We compute the spectral component E_0 from

$$\begin{aligned} E_0 &= \frac{1}{\sigma_3} (E_3 + \sigma_1 E_2 + \sigma_2 E_1) \\ &= \alpha^{-6} (E_3 + \alpha^7 E_2 + \alpha^4 E_1) \\ &= \alpha^{-6} (\alpha^{14} + \alpha^7 + \alpha^{16}) \\ &= 0. \end{aligned}$$

The resultant error spectral polynomial is

$$\begin{aligned} \mathbf{E}(X) &= \alpha^{12} X + X^2 + \alpha^{14} X^3 + \alpha^{10} X^4 + \alpha^{12} X^6 \\ &\quad + X^7 + \alpha^{14} X^8 + \alpha^{10} X^9 + \alpha^{12} X^{11} \\ &\quad + X^{12} + \alpha^{14} X^{13} + \alpha^{10} X^{14}. \end{aligned}$$

We find that $\mathbf{E}(X) = \mathbf{R}(X)$, and $\mathbf{V}(X) = 0$. Therefore, the decoded codeword is the all-zero codeword. The inverse transform of $\mathbf{E}(X)$ is $\mathbf{e}(X) = \alpha^7 X^3 + \alpha^3 X^6 + \alpha^4 X^{12}$. This is exactly the same result as given in Example 7.2.

7.7 CORRECTION OF ERRORS AND ERASURES

A q -ary t -error-correcting BCH (or RS) code can be used to correct all combinations of v symbol errors and e symbol erasures provided that the inequality

$$v + e/2 \leq t \quad (7.84)$$

holds. Each of the decoding algorithms presented in the last three sections can be modified to do the job.

Suppose the received polynomial $\mathbf{r}(X)$ contains v symbol errors at positions $X^{i_1}, X^{i_2}, \dots, X^{i_v}$, and e symbol erasures at positions $X^{j_1}, X^{j_2}, \dots, X^{j_e}$. Because the erased positions are known, decoding is to find the locations and values of the errors and the values of the erased symbols. The erasure-location numbers corresponding to the erased positions $X^{j_1}, X^{j_2}, \dots, X^{j_e}$ are $\alpha^{j_1}, \alpha^{j_2}, \dots, \alpha^{j_e}$. We form the erasure-location polynomial:

$$\beta(X) \triangleq \prod_{l=1}^e (1 - \alpha^{j_l} X). \quad (7.85)$$

Now, we fill the e erased positions in $\mathbf{r}(X)$ with zeros (or arbitrary symbols from $GF(q)$). This substitution of e zeros into the erased positions in $\mathbf{r}(X)$ can introduce up to e additional errors. Let $\mathbf{r}^*(X)$ denote the modified received polynomial. Let

$$\sigma(X) \triangleq \prod_{k=1}^v (1 - \alpha^{i_k} X) \quad (7.86)$$

be the error-location polynomial for the errors in $\mathbf{r}(X)$ at positions $X^{i_1}, X^{i_2}, \dots, X^{i_v}$. Then, the error-location polynomial for the modified received polynomial $\mathbf{r}^*(X)$ is

$$\gamma(X) = \sigma(X)\beta(X), \quad (7.87)$$

for which $\beta(X)$ is known. Now, decoding is to find $\sigma(X)$ and the error-value evaluator $\mathbf{Z}_0(X)$ for $\mathbf{r}^*(X)$.

We compute the syndrome polynomial

$$\mathbf{S}(X) = S_1 + S_2 X + \cdots + S_{2t} X^{2t-1}$$

from the modified received polynomial $\mathbf{r}^*(X)$. Then, the key equation becomes

$$\sigma(X)\beta(X)\mathbf{S}(X) \equiv \mathbf{Z}_0(X) \pmod{X^{2t}}. \quad (7.88)$$

The decoding problem is to find the solution $(\sigma(X), \mathbf{Z}_0(X))$ of this equation such that $\sigma(X)$ has minimum degree v , and $\deg \mathbf{Z}_0(X) < v + e$. Since $\beta(X)$ and $\mathbf{S}(X)$ are known, we can combine them. Let

$$\begin{aligned} \mathbf{T}(X) &\triangleq [\beta(X)\mathbf{S}(X)]_{2t} \\ &= T_1 + T_2 X + \cdots + T_{2t} X^{2t-1} \end{aligned} \quad (7.89)$$

denote the polynomial that consists of the $2t$ terms of $\beta(X)\mathbf{S}(X)$ from X^0 to X^{2t-1} . Then, we can write the key equation of (7.88) as

$$\sigma(X)\mathbf{T}(X) \equiv \mathbf{Z}_0(X) \bmod X^{2t}. \quad (7.90)$$

This key equation may be solved by using either Euclid's or Berlekamp's algorithm.

The Euclidean algorithm for error/erasure decoding consists of the following:

1. Compute the erasure-location polynomial $\beta(X)$ using the erasure information from the received polynomial $\mathbf{r}(X)$.
2. Form the modified received polynomial $\mathbf{r}^*(X)$ by replacing the erased symbols with zeros. Compute the syndrome polynomial $\mathbf{S}(X)$ from $\mathbf{r}^*(X)$.
3. Compute the modified syndrome polynomial $\mathbf{T}(X) = [\beta(X) \mathbf{S}(X)]_{2t}$.
4. Set the following initial conditions:

$$\begin{aligned}\mathbf{Z}_0^{(-1)}(X) &= X^{2t}, \quad \mathbf{Z}_0^{(0)}(X) = \mathbf{T}(X), \\ \sigma^{(-1)}(X) &= 0, \quad \text{and} \quad \sigma^{(0)}(X) = 1.\end{aligned}$$

5. Execute the Euclidean algorithm iteratively as described in Section 7.5 until a step ρ is reached for which

$$\deg \mathbf{Z}_0^{(\rho)}(X) < \begin{cases} t + e/2, & \text{for even } e, \\ t + (e - 1)/2, & \text{for odd } e. \end{cases} \quad (7.91)$$

Then, set $\sigma(X) = \sigma^{(\rho)}(X)$, and $\mathbf{Z}_0(X) = \mathbf{Z}_0^{(\rho)}(X)$.

6. Find the roots of $\sigma(X)$ and determine the error locations in $\mathbf{r}(X)$.
7. Determine the values of errors and erasures from $\mathbf{Z}_0(X)$ and $\gamma(X) = \sigma(X)\beta(X)$. The error values are given by

$$e_{i_k} = \frac{-\mathbf{Z}_0(\alpha^{-i_k})}{\gamma'(\alpha^{-i_k})} \quad (7.92)$$

for $1 \leq k \leq v$, and the values of the erased symbols are given by

$$f_{j_l} = \frac{-\mathbf{Z}_0(\alpha^{-j_l})}{\gamma'(\alpha^{-j_l})} \quad (7.93)$$

for $1 \leq l \leq e$, where $\gamma'(X)$ is the derivative of the overall error/erasure-location polynomial $\gamma(X) = \sigma(X)\beta(X)$; that is,

$$\begin{aligned}\gamma'(X) &= \frac{d}{dX} \gamma(X) \\ &= -a \sum_{l=1}^{v+e} \alpha^{j_l} \prod_{i=1, i \neq l}^{v+e} (1 - \alpha^{j_i} X).\end{aligned} \quad (7.94)$$

(a is the constant $\neq 1$ that may appear in $\sigma(X)$ when the Euclidean algorithm is used).

EXAMPLE 7.8

Again consider the triple-error-correcting RS code of length 15 over $GF(2^4)$ generated by $\mathbf{g}(X) = (X + \alpha)(X + \alpha^2)(X + \alpha^3)(X + \alpha^4)(X + \alpha^5)(X + \alpha^6)$. This code is capable of correcting all combinations of two or fewer errors and two or fewer erasures. Suppose the all-zero codeword is transmitted, and the received vector is

$$\mathbf{r} = (000 * 00 * 00\alpha 00\alpha^4 00),$$

where $*$ denotes an erasure. The received polynomial is

$$\mathbf{r}(X) = (*)X^3 + (*)X^6 + \alpha X^9 + \alpha^4 X^{12}.$$

Because the erased positions are X^3 and X^6 , the erasure-location polynomial is

$$\begin{aligned}\beta(X) &= (1 + \alpha^3 X)(1 + \alpha^6 X) \\ &= 1 + \alpha^2 X + \alpha^9 X^2.\end{aligned}$$

Replacing the erased symbols with zeros, we obtain the following modified received polynomial:

$$\mathbf{r}^*(X) = \alpha X^9 + \alpha^4 X^{12}.$$

The syndrome components computed from $\mathbf{r}^*(X)$ are

$$\begin{array}{ll} S_1 = \mathbf{r}^*(\alpha) = \alpha^8, & S_4 = \mathbf{r}^*(\alpha^4) = 0, \\ S_2 = \mathbf{r}^*(\alpha^2) = \alpha^{11}, & S_5 = \mathbf{r}^*(\alpha^5) = 1, \\ S_3 = \mathbf{r}^*(\alpha^3) = \alpha^9, & S_6 = \mathbf{r}^*(\alpha^6) = \alpha^8. \end{array}$$

The syndrome polynomial is then

$$\mathbf{S}(X) = \alpha^8 + \alpha^{11}X + \alpha^9X^2 + X^4 + \alpha^8X^5,$$

and the modified syndrome polynomial is

$$\begin{aligned}\mathbf{T}(X) &= [\beta(X)\mathbf{S}(X)]_{2t} \\ &= \alpha^8 + \alpha^{14}X + \alpha^4X^2 + \alpha^3X^3 + \alpha^{14}X^4 + X^5.\end{aligned}$$

Using the Euclidean decoding algorithm, we set the initial conditions as follows:

$$\begin{aligned}\mathbf{Z}_0^{(-1)}(X) &= X^6, \quad \mathbf{Z}_0^{(0)}(X) = \mathbf{T}(X), \\ \sigma^{(-1)}(X) &= 0, \quad \text{and} \quad \sigma^{(0)}(X) = 1.\end{aligned}$$

Since $t = 3$ and $e = 2$, the algorithm terminates when $\deg \mathbf{Z}_0(X) < 4$. Executing the algorithm, we obtain Table 7.6. The error-location polynomial is

$$\begin{aligned}\sigma(X) &= \alpha(1 + \alpha^8X + \alpha^6X^2) \\ &= \alpha(1 + \alpha^9X)(1 + \alpha^{12}X).\end{aligned}$$

The two roots of $\sigma(X)$ are α^{-9} and α^{-12} . The reciprocals of these two roots give the error locations, α^9 and α^{12} . The error-value evaluator is

$$\mathbf{Z}_0(X) = \alpha^9 + \alpha^8X + \alpha X^2 + \alpha X^3.$$

The overall error/erasure-location polynomial is

$$\begin{aligned}\gamma(X) &= \sigma(X)\beta(X) \\ &= \alpha(1 + \alpha^3X)(1 + \alpha^6X)(1 + \alpha^9X)(1 + \alpha^{12}X),\end{aligned}$$

and its derivative is

$$\begin{aligned}\gamma'(X) &= \alpha^4(1 + \alpha^6X)(1 + \alpha^9X)(1 + \alpha^{12}X) \\ &\quad + \alpha^7(1 + \alpha^3X)(1 + \alpha^9X)(1 + \alpha^{12}X) \\ &\quad + \alpha^{10}(1 + \alpha^3X)(1 + \alpha^6X)(1 + \alpha^{12}X) \\ &\quad + \alpha^{13}(1 + \alpha^3X)(1 + \alpha^6X)(1 + \alpha^9X).\end{aligned}$$

It follows from (7.92) and (7.93) that the error values at positions X^9 and X^{12} are

$$\begin{aligned}e_9 &= \frac{-\mathbf{Z}_0(\alpha^{-9})}{\gamma'(\alpha^{-9})} = \frac{\alpha^{13}}{\alpha^{12}} = \alpha, \\ e_{12} &= \frac{-\mathbf{Z}_0(\alpha^{-12})}{\gamma'(\alpha^{-12})} = \frac{\alpha^3}{\alpha^{14}} = \alpha^{-11} = \alpha^4,\end{aligned}$$

and the values of the erased symbols at positions X^3 and X^6 are

$$\begin{aligned}f_3 &= \frac{-\mathbf{Z}_0(\alpha^{-3})}{\gamma'(\alpha^{-3})} = \frac{0}{\alpha^8} = 0, \\ f_6 &= \frac{-\mathbf{Z}_0(\alpha^{-6})}{\gamma'(\alpha^{-6})} = \frac{0}{1} = 0.\end{aligned}$$

Then, the estimated error polynomial is

$$\mathbf{e}(X) = \alpha X^9 + \alpha^4 X^{12}.$$

Subtracting $\mathbf{e}(X)$ from $\mathbf{r}^*(X)$, we obtain the decoded code polynomial $\mathbf{v}(X) = \mathbf{0}$, which is the transmitted code polynomial.

EXAMPLE 7.9

Consider the (63, 55, 9) RS code generated by

$$\begin{aligned}\mathbf{g}(X) &= (X + \alpha)(X + \alpha^2)(X + \alpha^3)(X + \alpha^4)(X + \alpha^5)(X + \alpha^6)(X + \alpha^7)(X + \alpha^8) \\ &= X^8 + \alpha^{43}X^7 + \alpha^{59}X^6 + \alpha^{31}X^5 + \alpha^{10}X^4 + \alpha^{40}X^3 + \alpha^{14}X^2 + \alpha^7X + \alpha^{36}.\end{aligned}$$

TABLE 7.6: Steps for finding the error-location polynomial and error-value evaluator of the RS code given in Example 7.6.

i	$\mathbf{Z}_0^{(i)}(X)$	$\mathbf{q}_i(X)$	$\sigma(X)$
-1	X^6	—	0
0	$\mathbf{T}(X)$	—	1
1	$\alpha^7 + \alpha^3X + X^2 + \alpha^{10}X^3 + \alpha^8X^4$	$\alpha^{14} + X$	$\alpha^{14} + X$
	$\alpha^9 + \alpha^8X + \alpha X^2 + \alpha X^3$	$\alpha^5 + \alpha^7X$	$\alpha + \alpha^9X + \alpha^7X^2$

This code is capable of correcting all combinations of three or fewer errors and two or fewer erasures. Suppose the all-zero codeword is transmitted, and the received vector is

$$\mathbf{r} = (000000\alpha^{15} 0000000000000\alpha^{37} 0000000 * 000 \\ 00\alpha^4 0000000000000000 * 000000000).$$

The received polynomial is

$$\mathbf{r} = \alpha^{15}X^6 + \alpha^{37}X^{20} + (*)X^{28} + \alpha^4X^{34} + (*)X^{53}.$$

Because the erased positions are X^{28} and X^{53} , the erasure-location polynomial is

$$\begin{aligned}\beta(X) &= (1 + \alpha^{28}X)(1 + \alpha^{53}X) \\ &= 1 + \alpha^{39}X + \alpha^{18}X^2.\end{aligned}$$

Replacing the erased symbols with zeros, we obtain the following modified received polynomial:

$$\mathbf{r}^*(X) = \alpha^{15}X^6 + \alpha^{37}X^{20} + \alpha^4X^{34}.$$

The syndrome components computed from $\mathbf{r}^*(X)$ are

$$\begin{aligned}S_1 &= \mathbf{r}^*(\alpha) = \alpha^{19}, & S_5 &= \mathbf{r}^*(\alpha^5) = \alpha^{43}, \\ S_2 &= \mathbf{r}^*(\alpha^2) = \alpha, & S_6 &= \mathbf{r}^*(\alpha^6) = \alpha^4, \\ S_3 &= \mathbf{r}^*(\alpha^3) = 1, & S_7 &= \mathbf{r}^*(\alpha^7) = \alpha^{58}, \\ S_4 &= \mathbf{r}^*(\alpha^4) = \alpha^{22}, & S_8 &= \mathbf{r}^*(\alpha^8) = \alpha^{28}.\end{aligned}$$

The syndrome polynomial is then

$$\mathbf{S}(X) = \alpha^{19} + \alpha X + X^2 + \alpha^{22}X^3 + \alpha^{43}X^4 + \alpha^4X^5 + \alpha^{58}X^6 + \alpha^{28}X^7,$$

and the modified syndrome polynomial is

$$\begin{aligned}\mathbf{T}(X) &= [\beta(X)\mathbf{S}(X)]_{2t} \\ &= \alpha^{19} + \alpha^{59}X + \alpha X^2 + \alpha^{41}X^3 + \alpha^{32}X^4 + \alpha^{62}X^5 + \alpha^{60}X^6 + \alpha^{48}X^7.\end{aligned}$$

Using the Euclidean decoding algorithm, we set the initial condition as follows:

$$\mathbf{Z}_0^{(-1)}(X) = X^8, \quad \mathbf{Z}_0^{(0)}(X) = \mathbf{T}(X),$$

$$\sigma^{(-1)}(X) = 0, \quad \text{and} \quad \sigma^{(0)}(X) = 1.$$

TABLE 7.7: Steps finding the error-location polynomial and error-value evaluator of the (63,55,9) RS code over $GF(2^6)$ given in Example 7.9.

i	$\mathbf{Z}_0^{(i)}(X)$	$\mathbf{q}_i(X)$	$\sigma(X)$
-1	X^8	—	0
0	$\mathbf{T}(X)$	—	1
1	$\alpha^{46} + \alpha^{48}X + \alpha^{58}X^2 + \alpha^{30}X^3 + \alpha^{25}X^4 + \alpha^5X^5 + \alpha^{12}X^6$	$\alpha^{27} + \alpha^{15}X$	$\alpha^{27} + \alpha^{15}X$
2	$\alpha^{57} + \alpha^{31}X + \alpha^{56}X^2 + \alpha^{44}X^3 + \alpha^{17}X^4 + \alpha^{19}X^5$	$\alpha^{22} + \alpha^{36}X$	$\alpha^{38} + \alpha^{44}X + \alpha^{51}X^2$
3	$\alpha^3 + \alpha^{53}X + \alpha^{30}X^2 + \alpha^{24}X^3 + \alpha^{13}X^4$	$\alpha^{48} + \alpha^{56}X$	$\alpha^{47} + \alpha^{22}X + \alpha^{42}X^2 + \alpha^{44}X^3$

Since $t = 4$ and $e = 2$, the algorithm terminates when $\deg \mathbf{Z}_0(X) < 5$. Executing the algorithm, we obtain Table 7.7. The error-location polynomial is

$$\begin{aligned}\sigma(X) &= \alpha^{47}(1 + \alpha^{38}X + \alpha^{58}X^2 + \alpha^{60}X^3) \\ &= \alpha^{47}(1 + \alpha^6X)(1 + \alpha^{20}X)(1 + \alpha^{34}X).\end{aligned}$$

The three roots of $\sigma(X)$ are α^{-6} , α^{-20} , and α^{-34} . The reciprocals of these three roots give the error locations, α^6 , α^{20} , and α^{34} . The error-value evaluator is

$$\mathbf{Z}_0(X) = \alpha^3 + \alpha^{53}X + \alpha^{30}X^2 + \alpha^{24}X^3 + \alpha^{13}X^4.$$

The overall error/erasure-location polynomial is

$$\begin{aligned}\gamma(X) &= \sigma(X)\beta(X) \\ &= \alpha^{47}(1 + \alpha^6X)(1 + \alpha^{20}X)(1 + \alpha^{28}X)(1 + \alpha^{34}X)(1 + \alpha^{53}X) \\ &= \alpha^{47} + \alpha^{28}X + \alpha^{18}X^2 + \alpha^{48}X^3 + \alpha^{12}X^4 + \alpha^{62}X^5,\end{aligned}$$

and its derivative is

$$\gamma'(X) = \alpha^{28} + \alpha^{48}X^2 + \alpha^{62}X^4.$$

The error values at positions X^6 , X^{20} , and X^{34} are

$$e_6 = \frac{-\mathbf{Z}_0(\alpha^{-6})}{\gamma'(\alpha^{-6})} = \frac{\alpha^{39}}{\alpha^{24}} = \alpha^{15},$$

$$e_{20} = \frac{-\mathbf{Z}_0(\alpha^{-20})}{\gamma'(\alpha^{-20})} = \frac{\alpha^6}{\alpha^{32}} = \alpha^{37},$$

$$e_{34} = \frac{-\mathbf{Z}_0(\alpha^{-34})}{\gamma'(\alpha^{-34})} = \frac{\alpha^{61}}{\alpha^{57}} = \alpha^4,$$

and the values of the erased symbols at positions X^{28} and X^{53} are

$$f_{28} = \frac{-\mathbf{Z}_0(\alpha^{-28})}{\mathbf{y}'(\alpha^{-28})} = \frac{0}{\alpha^{29}} = 0,$$

$$f_{53} = \frac{-\mathbf{Z}_0(\alpha^{-53})}{\mathbf{y}'(\alpha^{-53})} = \frac{0}{\alpha^{13}} = 0.$$

Then, the estimated error polynomial is

$$\mathbf{e}(X) = \alpha^{15}X^6 + \alpha^{37}X^{20} + \alpha^4X^{34}.$$

Subtracting $\mathbf{e}(X)$ from $\mathbf{r}^*(X)$, we obtain the decoded code polynomial $\mathbf{v}(X) = \mathbf{0}$, which is the transmitted code polynomial.

PROBLEMS

- 7.1** Consider the triple-error-correcting RS code given in Example 7.2. Find the code polynomial for the message

$$\mathbf{a}(X) = 1 + \alpha^5X + \alpha X^4 + \alpha^7X^8.$$

- 7.2** Using the Galois field $GF(2^5)$ given in Appendix A, find the generator polynomials of the double-error-correcting and triple-error-correcting RS codes of length 31.

- 7.3** Using the Galois field $GF(2^6)$ given in Table 6.2, find the generator polynomials of double-error-correcting and triple-error-correcting RS codes of length 63.

- 7.4** Consider the triple-error-correcting RS code of length 15 given in Example 7.2. Decode the received polynomial

$$\mathbf{r}(X) = \alpha^4X^3 + \alpha^9X^8 + \alpha^3X^{13}$$

using the Berlekamp algorithm.

- 7.5** Continue Problem 7.4. Decode the received polynomial with the Euclidean algorithm.

- 7.6** Consider the triple-error-correcting RS code of length 31 constructed in Problem 7.2. Decode the received polynomial

$$\mathbf{r}(X) = \alpha^2 + \alpha^{21}X^{12} + \alpha^7X^{20}$$

using the Euclidean algorithm.

- 7.7** Continue Problem 7.6. Decode the received polynomial in the frequency domain using transform decoding.

- 7.8** For the same RS code of Problem 7.6, decode the following received polynomial with two erasures:

$$\mathbf{r}(X) = (*)X^3 + \alpha^5X^7 + (*)X^{18} + \alpha^3X^{21}$$

with the Euclidean algorithm.

- 7.9** Prove that the dual code of a RS code is also a RS code.

- 7.10** Prove that the $(2^m - 1, k)$ RS code with minimum distance d contains the primitive binary BCH code of length $2^m - 1$ with designed distance d as a subcode. This subcode is called a *subfield subcode*.

- 7.11** Let α be a primitive element in $GF(2^m)$. Consider the $(2^m - 1, k)$ RS code of length of $2^m - 1$ and minimum distance d generated by

$$\mathbf{g}(X) = (X - \alpha)(X - \alpha^2)\dots(X - \alpha^{d-1}).$$

Prove that extending each codeword $v = (v_0, v_1, \dots, v_{2^m-2})$ by adding an overall parity-check symbol

$$v_\infty = - \sum_{i=0}^{2^m-2} v_i$$

produces a $(2^m, k)$ code with a minimum distance of $d + 1$.

- 7.12** Consider a t -symbol error-correcting RS code over $GF(2^m)$ with the following parity-check matrix:

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{n-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & \dots & (\alpha^2)^{n-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & \dots & (\alpha^{2t})^{n-1} \end{bmatrix},$$

where $n = 2^m - 1$, and α is a primitive element in $GF(2^m)$. Consider the extended Reed–Solomon code with the following parity-check matrix:

$$\mathbf{H}_1 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 1 & 0 \end{bmatrix}$$

Prove that the extended code also has a minimum distance of $2t + 1$.

- 7.13** Let $\mathbf{a}(X) = a_0 + a_1X + \dots + a_{k-1}X^{k-1}$ be a polynomial of degree $k - 1$ or less over $GF(2^m)$. There are $(2^m)^k$ such polynomials. Let α be a primitive element in $GF(2^m)$. For each polynomial $\mathbf{a}(X)$, form the following polynomial of degree $2^m - 2$ or less over $GF(2^m)$:

$$\mathbf{v}(X) = \mathbf{a}(1) + \mathbf{a}(\alpha)X + \mathbf{a}(\alpha^2)X^2 + \dots + \mathbf{a}(\alpha^{2^m-2})X^{2^m-2}.$$

Prove that the set $\{\mathbf{v}(X)\}$ forms the $(2^m - 1, k)$ RS code over $GF(2^m)$. (Hint: Show that $\mathbf{v}(X)$ has $\alpha, \alpha^2, \dots, \alpha^{2^m-k-1}$ as roots). This original definition of a RS code is given by Reed and Solomon [1].

BIBLIOGRAPHY

1. I. S. Reed and G. Solomon, “Polynomial Codes over Certain Fields,” *J. Soc. Ind. Appl. Math.*, 8: 300–304, June 1960.
2. D. Gorenstein and N. Zierler, “A Class of Cyclic Linear Error-Correcting Codes in p^m Symbols,” *J. Soc. Ind. Appl. Math.*, 9: 107–214, June 1961.
3. R. T. Chien, “Cyclic Decoding Procedure for the Bose–Chaudhuri–Hocquenghem Codes,” *IEEE Trans. Inf. Theory*, IT-10: 357–63, October 1964.

4. G. D. Forney, "On Decoding BCH Codes," *IEEE Trans. Inf. Theory*, IT-11: 549–57, October 1965.
5. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
6. Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A Method for Solving Key Equation for Decoding Goppa Codes," *Inf. Control*, 27: 87–99, January 1975.
7. W. C. Gore, "Transmitting Binary Symbols with Reed–Solomon Codes," *Proc. Conf. Infor. Sci. and Syst.*, Princeton, N.J., 495–97, 1973.
8. R. E. Blahut, "Transform Techniques for Error-Control Codes," *IBM J. Res. Dev.*, 23(3), 299–315 May 1979.
9. T. Kasami and S. Lin, "On the Probability of Undetected Error for the Maximum Distance Separable Codes," *IEEE Trans. Commun.*, COM-32: 998–1006, September 1984.
10. E. F. Assmus, Jr., H. F. Mattson, Jr., and R. J. Turyn, "Cyclic Codes," *Scientific Report No. AFCRL-65-332*, Air Force Cambridge Research Labs, Bedford, Mass., April 1965.
11. T. Kasami, S. Lin, and W. W. Peterson, "Some Results on Weight Distributions of BCH Codes," *IEEE Trans. Inf. Theory*, IT-12(2): 274, April 1966.
12. G. D. Forney, Jr., *Concatenated Codes*, MIT Press, Cambridge, 1966.
13. J. L. Massey, "Shift-Register Synthesis and BCH Decoding," *IEEE Trans. Inf. Theory*, IT-15: 122–27, January 1969.
14. W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2d ed., MIT Press, Cambridge, 1970.
15. F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North Holland, Amsterdam, 1977.
16. G. C. Clark, Jr., and J. B. Cain, *Error-Correcting Coding for Digital Communications*, Plenum Press, New York, 1981.
17. R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, Reading, Mass., 1983.
18. A. M. Michelson and A. H. Levesque, *Error-Control Techniques for Digital Communication*, John Wiley, New York, 1985.
19. S. B. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice Hall, Englewood Cliffs, N.J., 1995.
20. S. B. Wicker and V. K. Bhargava, *Reed–Solomon Codes and Their Applications*, IEEE Press, New York, 1994.

21. T. Kasami, S. Lin, and W. W. Peterson, "Some Results on Cyclic Codes Which Are Invariant under the Affine Group," *Scientific Report AFCRL-66-622*, Air Force Cambridge Research Labs, Bedford, Mass., 1966.
22. J. K. Wolf, "Adding Two Information Symbols to Certain Nonbinary BCH Codes and Some Applications," *Bell Syst. Tech. J.*, 48: 2405–24, 1969.
23. M. Morii and M. Kasahara, "Generalized Key-Equation of Remainder Decoding Algorithm for Reed–Solomon Codes," *IEEE Trans. Inf. Theory*, IT-38 (6): 1801–7, November 1992.

CHAPTER 8

Majority-Logic Decodable and Finite Geometry Codes

Majority-logic decoding is a simple and effective scheme for decoding certain classes of block codes, especially for decoding certain classes of cyclic codes. The first majority-logic decoding algorithm was devised in 1954 by Reed [1] for the class of RM codes presented in Chapter 4. Reed's algorithm was later extended and generalized by many coding theorists. The first unified formulation of majority-logic decoding was due to Massey [2].

Most majority-logic decodable codes found so far are cyclic codes. Important cyclic codes of this category are codes constructed based on finite geometries, namely, Euclidean and projective geometries. These codes are called *finite geometry codes* and they contain punctured RM codes in cyclic form as a subclass. A special subclass of finite geometry codes forms a special subclass of low-density parity-check codes that will be discussed in Chapter 17. Finite geometry codes were first investigated by Rudolph [3] in 1967. Rudolph's work was later extended and generalized by many coding researchers, from the late 1960s to the late 1970s.

In this chapter we first introduce majority-logic decoding based on orthogonal parity-check sums formed from the parity-check matrix or the dual space of a code. Then, we present several classes of cyclic majority-logic decodable codes.

8.1 ONE-STEP MAJORITY-LOGIC DECODING

Consider an (n, k) cyclic code C with parity-check matrix \mathbf{H} . The row space of \mathbf{H} is an $(n, n - k)$ cyclic code, denoted by C_d , which is the dual code of C , or the null space of C . For any codeword \mathbf{v} in C and any codeword \mathbf{w} in C_d , the inner product of \mathbf{v} and \mathbf{w} is zero; that is,

$$\mathbf{w} \cdot \mathbf{v} = w_0 v_0 + w_1 v_1 + \cdots + w_{n-1} v_{n-1} = 0. \quad (8.1)$$

In fact, an n -tuple \mathbf{v} is a codeword in C if and only if for any vector \mathbf{w} in C_d , $\mathbf{w} \cdot \mathbf{v} = 0$. The equality of (8.1) is called a *parity-check equation*. Clearly, there are $2^{(n-k)}$ such parity-check equations.

Now, suppose that a codeword \mathbf{v} in C is transmitted. Let $\mathbf{e} = (e_0, e_1, \dots, e_{n-1})$ and $\mathbf{r} = (r_0, r_1, \dots, r_{n-1})$ be the error vector and the received vector, respectively. Then,

$$\mathbf{r} = \mathbf{v} + \mathbf{e}. \quad (8.2)$$

For any vector \mathbf{w} in the dual code C_d , we can form the following linear sum of the received digits:

$$A = \mathbf{w} \cdot \mathbf{r} = w_0 r_0 + w_1 r_1 + \cdots + w_{n-1} r_{n-1}, \quad (8.3)$$

which is called a *parity-check sum* or simply *check-sum*. If the received vector \mathbf{r} is a codeword in C , this parity-check sum, A , *must be zero*; however, if \mathbf{r} is not a

codeword in C , then A may not be zero. Combining (8.2) and (8.3) and using the fact that $\mathbf{w} \cdot \mathbf{v} = 0$, we obtain the following relationship between the check-sum A and error digits in \mathbf{e} :

$$A = w_0 e_0 + w_1 e_1 + \cdots + w_{n-1} e_{n-1}. \quad (8.4)$$

An error digit e_l is said to be *checked* by the check-sum A if the coefficient $w_l = 1$. In the following, we show that certain properly formed check-sums can be used for estimating the error digits in \mathbf{e} .

Suppose that there exist J vectors in the dual code C_d ,

$$\mathbf{w}_1 = (w_{10}, w_{11}, \dots, w_{1,n-1}),$$

$$\mathbf{w}_2 = (w_{20}, w_{21}, \dots, w_{2,n-1}),$$

$$\vdots$$

$$\mathbf{w}_J = (w_{J0}, w_{J1}, \dots, w_{J,n-1}),$$

that have the following properties:

1. The $(n - 1)$ th component of each vector is a 1; that is,

$$w_{1,n-1} = w_{2,n-1} = \cdots = w_{J,n-1} = 1.$$

2. For $i \neq n - 1$, there is at most one vector whose i th component is a 1; for example, if $w_{1,i} = 1$, then $w_{2,i} = w_{3,i} = \cdots = w_{J,i} = 0$.

These J vectors are said to be *orthogonal* on the $(n - 1)$ th digit position. We call them *orthogonal vectors*. Now, we form J parity-check sums from these J orthogonal vectors:

$$\begin{aligned} A_1 &= \mathbf{w}_1 \cdot \mathbf{r} = w_{10} r_0 + w_{11} r_1 + \cdots + w_{1,n-1} r_{n-1} \\ A_2 &= \mathbf{w}_2 \cdot \mathbf{r} = w_{20} r_0 + w_{21} r_1 + \cdots + w_{2,n-1} r_{n-1} \\ &\vdots \\ A_J &= \mathbf{w}_J \cdot \mathbf{r} = w_{J0} r_0 + w_{J1} r_1 + \cdots + w_{J,n-1} r_{n-1}. \end{aligned} \quad (8.5)$$

Because $w_{1,n-1} = w_{2,n-1} = \cdots = w_{J,n-1} = 1$, these J check-sums are related to the error digits in the following manner:

$$\begin{aligned} A_1 &= w_{10} e_0 + w_{11} e_1 + \cdots + w_{1,n-2} e_{n-2} + e_{n-1} \\ A_2 &= w_{20} e_0 + w_{21} e_1 + \cdots + w_{2,n-2} e_{n-2} + e_{n-1} \\ &\vdots \\ A_J &= w_{J,0} e_0 + w_{J,1} e_1 + \cdots + w_{J,n-2} e_{n-2} + e_{n-1}. \end{aligned} \quad (8.6)$$

We see that the error digit e_{n-1} is checked by all the preceding check-sums. Because of the second property of the orthogonal vectors $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_J$ any error digit

other than e_{n-1} is checked by at most one check-sum. These J check-sums are said to be orthogonal on the error digit e_{n-1} . Since $w_{i,j} = 0$, or 1, each of the foregoing check-sums orthogonal on e_{n-1} is of the form

$$A_j = e_{n-1} + \sum_{i \neq n-1} e_i.$$

If all the error digits in the sum of A_j are zero for $i \neq n-1$, the value of e_{n-1} is equal to A_j (i.e., $e_{n-1} = A_j$). Based on this fact, the parity-check sums orthogonal on e_{n-1} can be used to estimate e_{n-1} or to decode the received digit r_{n-1} .

Suppose that there are $\lfloor J/2 \rfloor$ or fewer errors in the error vector $\mathbf{e} = (e_0, e_1, \dots, e_{n-1})$ (i.e., $\lfloor J/2 \rfloor$ or fewer components of \mathbf{e} are 1). If $e_{n-1} = 1$, the other nonzero error digits can distribute among at most $\lfloor J/2 \rfloor - 1$ check-sums orthogonal on e_{n-1} . Hence, at least $J - \lfloor J/2 \rfloor + 1$, or more than half of the check-sums orthogonal on e_{n-1} , are equal to $e_{n-1} = 1$; however, if $e_{n-1} = 0$, the nonzero error digits can distribute among at most $\lfloor J/2 \rfloor$ check-sums. Hence, at least $J - \lfloor J/2 \rfloor$ or at least half of the check sums orthogonal on e_{n-1} are equal to $e_{n-1} = 0$. Thus, the value of e_{n-1} is equal to the value assumed by a clear majority of the parity-check sums orthogonal on e_{n-1} ; if no value is assumed by a clear majority of the parity-check sums (i.e., there is a tie), the error digit e_{n-1} is zero. Based on the preceding facts, an algorithm for decoding e_{n-1} can be formulated as follows:

The error digit e_{n-1} is decoded as 1 if a clear majority of the parity-check sums orthogonal on e_{n-1} is 1; otherwise, e_{n-1} is decoded as 0.

Correct decoding of e_{n-1} is guaranteed if there are $\lfloor J/2 \rfloor$ or fewer errors in the error vector \mathbf{e} . If it is possible to form J parity-check sums orthogonal on e_{n-1} , it is possible to form J parity-check sums orthogonal on any error digit because of the cyclic symmetry of the code. The decoding of other error digits is identical to the decoding of e_{n-1} . The decoding algorithm just described is called *one-step majority-logic decoding* [2]. If J is the maximum number of parity-check sums orthogonal on e_{n-1} (or any error digit) that can be formed, then, by one-step majority-logic decoding, any error pattern of $\lfloor J/2 \rfloor$ or fewer errors can be corrected. The parameter $t_{ML} = \lfloor J/2 \rfloor$ is called the *majority-logic error-correcting capability* of the code. Let d_{min} be the minimum distance of the code. Clearly, the one-step majority-logic decoding is effective for this code only if $t_{ML} = \lfloor J/2 \rfloor$ is equal to or close to the error-correcting capability $t = \lfloor (d_{min} - 1)/2 \rfloor$ of the code; in other words, J should be equal to or close to $d_{min} - 1$.

DEFINITION 8.1 A cyclic code with minimum distance d_{min} is said to be *completely orthogonalizable* in one step if and only if it is possible to form $J = d_{min} - 1$ parity-check sums orthogonal on an error digit.

At this point, a clarifying example will be helpful.

EXAMPLE 8.1

Consider a $(15, 7)$ cyclic code generated by the polynomial

$$\mathbf{g}(X) = 1 + X^4 + X^6 + X^7 + X^8.$$

The parity-check matrix of this code (in systematic form) is found as follows:

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \\ \mathbf{h}_4 \\ \mathbf{h}_5 \\ \mathbf{h}_6 \\ \mathbf{h}_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Consider the following linear combinations of the rows of \mathbf{H} :

Digit positions:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$w_1 =$	$\mathbf{h}_3 = (0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1)$,														
$w_2 =$	$\mathbf{h}_1 + \mathbf{h}_5 = (0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1)$,														
$w_3 =$	$\mathbf{h}_0 + \mathbf{h}_2 + \mathbf{h}_6 = (1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)$,														
$w_4 =$	$\mathbf{h}_7 = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1)$.														

We see that all four vectors have a 1 at digit position 14 (or X^{14}), and at any other digit position, no more than one vector has a 1. Therefore, these four vectors are orthogonal on the digit position 14. Let \mathbf{r} be the received vector. The four parity-check sums formed from these orthogonal vectors are related to the error digits as follows:

$$\begin{array}{lllll} A_1 = \mathbf{w}_1 \cdot \mathbf{r} = & e_3 & & +e_{11} + e_{12} & +e_{14} \\ A_2 = \mathbf{w}_2 \cdot \mathbf{r} = & e_1 & +e_5 & & +e_{13} + e_{14} \\ A_3 = \mathbf{w}_3 \cdot \mathbf{r} = & e_0 & +e_2 & +e_6 & +e_{14} \\ A_4 = \mathbf{w}_4 \cdot \mathbf{r} = & & & e_7 + e_8 + e_{10} & +e_{14}. \end{array}$$

We see that e_{14} is checked by all four check-sums, and no other error digit is checked by more than one check-sum. If $e_{14} = 1$, and if there is one or no error occurring among the other 14 digit positions, then at least three (majority) of the four sums, A_1, A_2, A_3 , and A_4 , are equal to $e_{14} = 1$. If $e_{14} = 0$ and if there are two or fewer errors occurring among the other 14 digit positions, then at least two of the four check-sums are equal to $e_{14} = 0$. Hence, if there are two or fewer errors in \mathbf{e} , the one-step majority-logic decoding always results in correct decoding of e_{14} . Because the code is cyclic, four parity-check sums orthogonal on any error digit can be formed. It can be checked that four is the maximum number of parity-check sums orthogonal on any error digit that can be formed. Thus, by one-step majority-logic decoding, the code is capable of correcting any error pattern with two or fewer errors. It can be shown that there exists at least one error pattern with three errors that cannot be corrected. Consider an error pattern \mathbf{e} with three errors, e_0, e_3 , and e_8 (i.e., $e_0 = e_3 = e_8 = 1$). From the four parity-check sums orthogonal on e_{14} , we have $A_1 = 1, A_2 = 0, A_3 = 1$, and $A_4 = 1$. Because the majority of the four sums is 1, according to the decoding rule, e_{14} is decoded as 1. This results in an incorrect decoding. The code given in this example is actually a BCH code with a minimum distance of exactly 5. Therefore, it is completely orthogonalizable.

Given an (n, k) cyclic code C for which J parity-check sums orthogonal on an error digit can be formed, the one-step majority-logic decoding of the code can easily be implemented. First, from the null space C_d of the code, we determine a set of J vectors $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_J$ that are orthogonal on the highest-order digit position, X^{n-1} . Then, J parity-check sums A_1, A_2, \dots, A_J orthogonal on the error digit e_{n-1} are formed from these J orthogonal vectors and the received vector \mathbf{r} . From (8.5), we see that the vector \mathbf{w}_j tells what received digits should be summed up to from the check-sum A_j . The J check-sums can be formed by using J multi-input modulo-2 adders. Once these J check-sums are formed, they are used as inputs to a J -input majority-logic gate. The output of a majority-logic gate is 1 if and only if more than half its inputs are 1; otherwise, the output is 0. The output is the estimated value of e_{n-1} . A general one-step majority-logic decoder is shown in Figure 8.1. This decoder is called the type-II one-step majority-logic decoder [2]. The error correction procedure is as follows:

- Step 1.** With gate 1 turned on and gate 2 turned off, the received vector \mathbf{r} is read into the buffer register.
- Step 2.** The J parity-check sums orthogonal on e_{n-1} are formed by summing the appropriate received digits.
- Step 3.** The J orthogonal check sums are fed into a majority-logic gate. The first received digit r_{n-1} is read out of the buffer and is corrected by the output of the majority-logic gate.
- Step 4.** At the end of step 3, the buffer register has been shifted one place to the right with gate 2 on. Now, the second received digit is in the rightmost stage of the buffer register and is corrected in exactly the

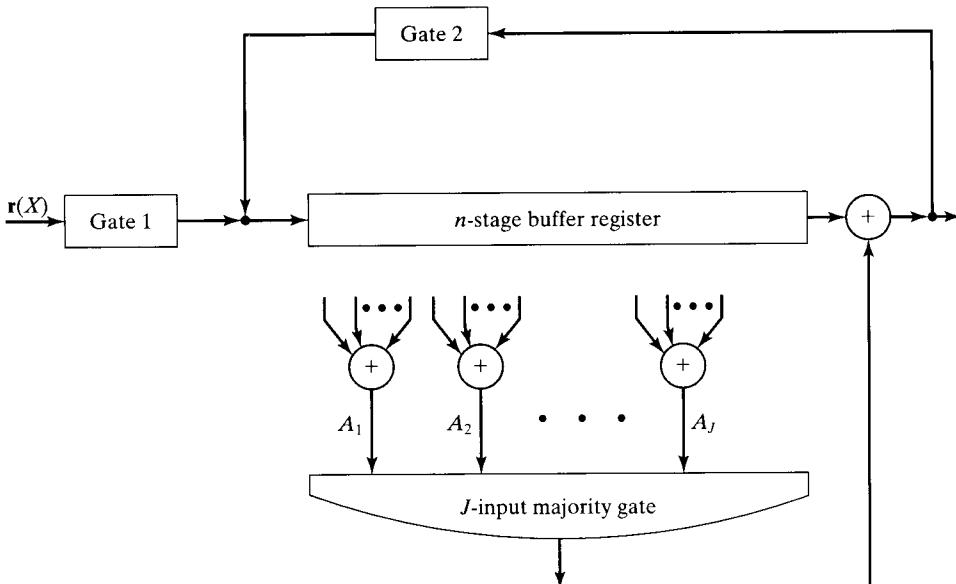


FIGURE 8.1: General type-II one-step majority-logic decoder.

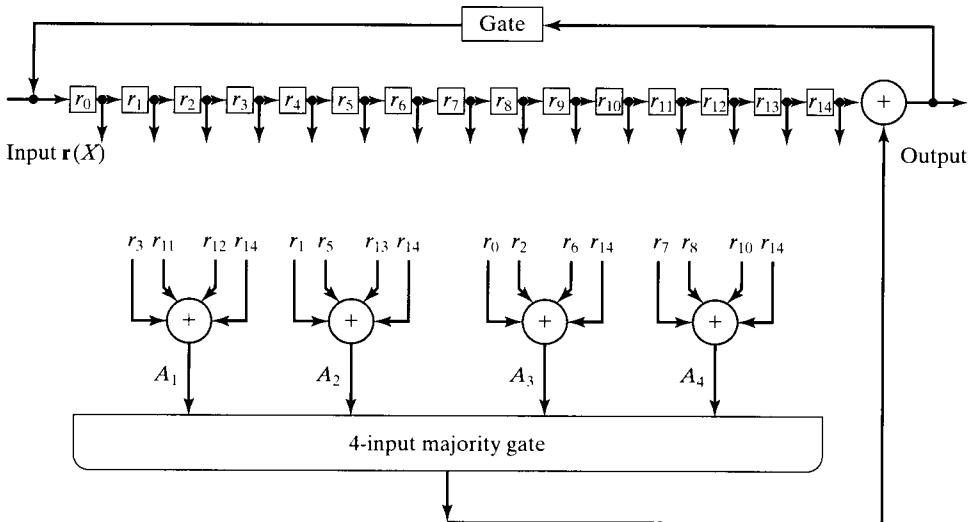


FIGURE 8.2: Type-II one-step majority-logic decoder for the (15, 7) BCH code.

same manner as was the first received digit. The decoder repeats step 2 and 3.

Step 5. The received vector is decoded digit by digit in the same manner until a total of n shifts.

If the received vector \mathbf{r} contains $\lfloor J/2 \rfloor$ or fewer errors, the buffer register should contain the transmitted code vector, and the inputs to the majority-logic gate should all be zero at the completion of the decoding operation. If not all the inputs to the majority gate are zero, an *uncorrectable error pattern* has been detected.

The type-II one-step majority-logic decoder for the (15, 7) BCH code considered in Example 8.1 is shown in Figure 8.2.

The parity-check sums orthogonal on an error digit also can be formed from the syndrome digits. Let

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \mathbf{h}_2 \\ \vdots \\ \mathbf{h}_{n-k-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & p_{00} & p_{01} & p_{0,k-1} \\ 0 & 1 & 0 & 0 & \cdots & 0 & p_{10} & p_{11} & p_{1,k-1} \\ 0 & 0 & 1 & 0 & \cdots & 0 & p_{20} & p_{21} & p_{2,k-1} \\ \vdots & & & & & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & p_{n-k-1,0} & p_{n-k-1,1} & \cdots & p_{n-k-1,k-1} \end{bmatrix}$$

be the parity-check matrix for an (n, k) cyclic code C in systematic form. Because the orthogonal vectors $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_J$ are vectors in the row space of \mathbf{H} , they are linear combinations of rows of \mathbf{H} . Let

$$\begin{aligned} \mathbf{w}_j &= (w_{j0}, w_{j1}, \dots, w_{j,n-1}) \\ &= a_{j0}\mathbf{h}_0 + a_{j1}\mathbf{h}_1 + \cdots + a_{j,n-k-1}\mathbf{h}_{n-k-1}. \end{aligned}$$

Because of the systematic structure of \mathbf{H} , we see that

$$w_{j0} = a_{j0}, \quad w_{j1} = a_{j1}, \dots, w_{j,n-k-1} = a_{j,n-k-1}. \quad (8.7)$$

Let $\mathbf{r} = (r_0, r_1, \dots, r_{n-1})$ be the received vector. Then, the syndrome of \mathbf{r} is

$$\mathbf{s} = (s_0, s_1, \dots, s_{n-k-1}) = \mathbf{r} \cdot \mathbf{H}^T,$$

where the i th syndrome digit is

$$s_i = \mathbf{r} \cdot \mathbf{h}_i \quad (8.8)$$

for $0 \leq i < n - k$. Now, consider the parity-check sum

$$\begin{aligned} A_j &= \mathbf{w}_j \cdot \mathbf{r} \\ &= (a_{j0}\mathbf{h}_0 + a_{j1}\mathbf{h}_1 + \dots + a_{j,n-k-1}\mathbf{h}_{n-k-1}) \cdot \mathbf{r} \\ &= a_{j0}\mathbf{r} \cdot \mathbf{h}_0 + a_{j1}\mathbf{r} \cdot \mathbf{h}_1 + \dots + a_{j,n-k-1}\mathbf{r} \cdot \mathbf{h}_{n-k-1}. \end{aligned} \quad (8.9)$$

From (8.7), (8.8), and (8.9), we obtain

$$A_j = w_{j0}s_0 + w_{j1}s_1 + \dots + w_{j,n-k-1}s_{n-k-1}. \quad (8.10)$$

Thus, the check-sum A_j is simply a *linear sum of the syndrome digits whose coefficients are the first $n - k$ digits of the orthogonal vector \mathbf{w}_j* . Based on (8.10), we obtain a different implementation of the one-step majority-logic decoding, as shown in Figure 8.3 (the received vector can be shifted into the syndrome register from the right end). This decoder is called the type-I one-step majority-logic decoder [2]. The error correction procedure is as follows:

- Step 1.** The syndrome is computed as usual by shifting the received polynomial $\mathbf{r}(X)$ into the syndrome register.
- Step 2.** The J parity-check sums orthogonal on e_{n-1} are formed by taking proper sums of the syndrome digits. These J check-sums are fed into a J -input majority-logic gate.
- Step 3.** The first received digit is read out of the buffer register and is corrected by the output of the majority gate. At the same time the syndrome register is also shifted once (with gate 2 on), and the effect e_{n-1} on the syndrome is removed (with gate 3 on). The new contents in the syndrome register form the syndrome of the altered received vector cyclically shifted one place to the right.
- Step 4.** The new syndrome formed in step 3 is used to decode the next received digit r_{n-2} . The decoder repeats steps 2 and 3. The received digit r_{n-2} is corrected in exactly the same manner as the first received digit r_{n-1} was corrected.
- Step 5.** The decoder decodes the received vector \mathbf{r} digit by digit in the same manner until a total of n shifts of the buffer and the syndrome registers.

At the completion of the decoding operation, the syndrome register should contain only zeros if the decoder output is a codeword. If the syndrome register

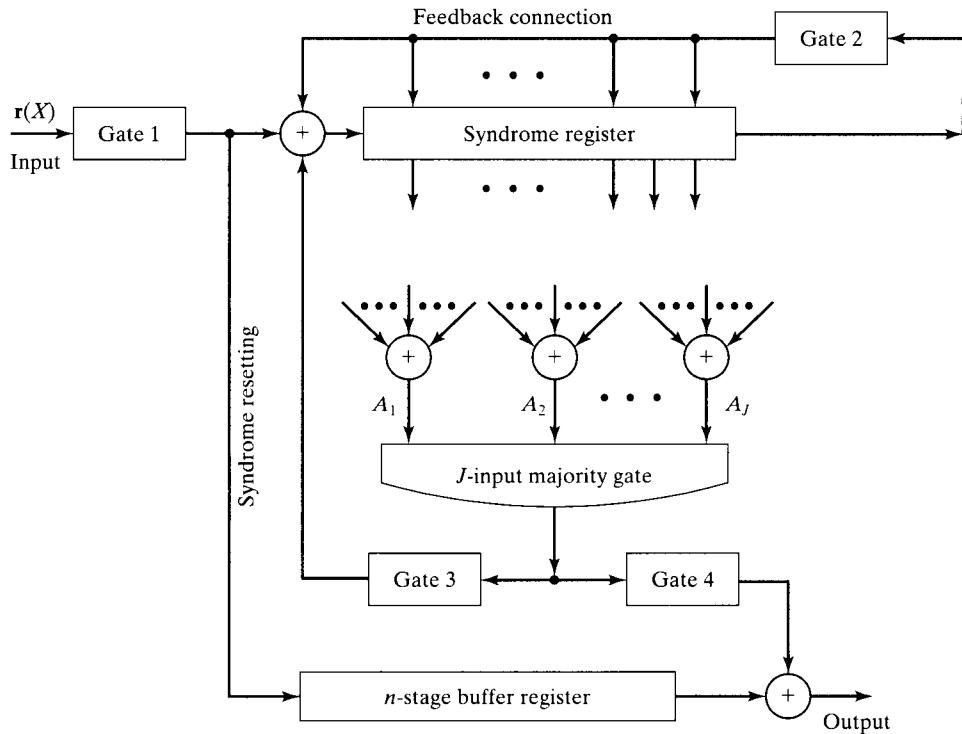


FIGURE 8.3: General type-I one-step majority-logic decoder.

does not contain all zeros at the end of the decoding, an uncorrectable error pattern has been detected. If we are interested in decoding only the received message digits but not the received parity digits, the buffer register needs store only the k received message digits, and it consists of only k stages. In this case, both type-I and type-II decoders require roughly the same amount of complexity.

EXAMPLE 8.2

Consider the $(15, 7)$ BCH code given in Example 8.1. From the vectors $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$, and \mathbf{w}_4 that are orthogonal on the digit position 14, we find that the parity-check sums orthogonal on e_{14} are equal to the following sums of syndrome digits:

$$A_1 = s_3, \quad A_2 = s_1 + s_5, \quad A_3 = s_0 + s_2 + s_6, \quad A_4 = s_7.$$

Based on these sums we construct the type-I one-step majority-logic decoder for the $(15, 7)$ BCH code as shown in Figure 8.4. Suppose that the all-zero codeword $(0, 0, \dots, 0)$ is transmitted, and $\mathbf{r}(X) = X^{13} + X^{14}$ is received. Clearly, there are two errors at locations X^{13} and X^{14} . After the entire received polynomial has entered the syndrome register, the syndrome register contains $(0\ 0\ 1\ 1\ 1\ 0\ 0\ 1)$. The four parity-check sums orthogonal on e_{14} are

$$A_1 = 1, \quad A_2 = 0, \quad A_3 = 1, \quad A_4 = 1.$$

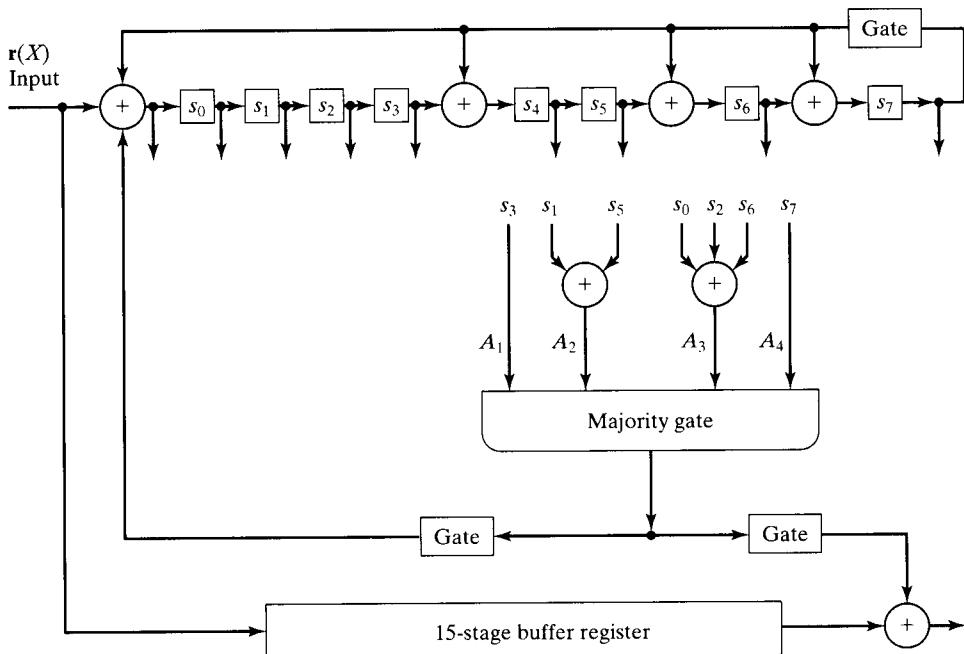


FIGURE 8.4: Type-I one-step majority-logic decoder for (15, 7) BCH code.

Because the majority of these four sums is 1, the output of the majority-logic gate is 1, which is the value of e_{14} . Simultaneously, the buffer and syndrome registers are shifted once; the highest-order received digit $r_{14} = 1$ is then corrected by the output of the majority-logic gate, and the new contents in the syndrome register are (0 0 0 1 0 1 1 1). The new parity-check sums are now

$$A_1^{(1)} = 1, \quad A_2^{(1)} = 1, \quad A_3^{(1)} = 1, \quad A_4^{(1)} = 1.$$

Again, the output of the majority-logic gate is 1, which is the value of e_{13} . Both the buffer and syndrome registers are shifted once more; the received digit r_{13} will be corrected, and the syndrome register will contain only zeros. At this point, both errors have been corrected, and the next 13 received digits are error-free.

One-step majority-logic decoding is most efficient for codes that are completely orthogonalizable, or for codes with large J compared with $d_{\min} - 1$. When J is small compared with $d_{\min} - 1$, one-step majority-logic decoding becomes very inefficient, and much of the error-correcting capability of the code is sacrificed. Given a code C , one would like to know the maximum number of parity-check sums orthogonal on an error digit that can be formed. This question is answered by Theorem 8.1.

THEOREM 8.1 Let C be an (n, k) cyclic code whose dual code C_d has minimum distance δ . Then, the number of parity-check sums orthogonal on an

error digit that can be formed, J , is upper bounded by

$$J \leq \left\lfloor \frac{n-1}{\delta-1} \right\rfloor. \quad (8.11)$$

Proof. Suppose that there exist J vectors $\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_J$ in the dual code of C that are orthogonal on the highest-order digit position, X^{n-1} . Because each of these J vectors has a weight of at least δ , the total number of 1's in these J vectors is at least $J\delta$; however, because of the orthogonal structure of these J vectors, the total number of 1's in them cannot exceed $J + (n - 1)$. Therefore, we have $J\delta \leq J + (n - 1)$. This implies that $J \leq (n - 1)/(\delta - 1)$. Because J is an integer, we must have $J \leq \lfloor (n - 1)/(\delta - 1) \rfloor$. Q.E.D.

The dual code of the $(15, 7)$ BCH code has a minimum distance of 4. Therefore, the maximum number of parity-check sums orthogonal on an error digit is upper bounded by $\lfloor 14/3 \rfloor = 4$. This proves our claim in Example 8.1 that $J = 4$ is the maximum number of parity-check sums orthogonal on an error digit that can be formed for the $(15, 7)$ BCH code.

If it is possible to form J parity-check sums orthogonal on an error digit for a cyclic code, then the code has a minimum distance of at least $J + 1$ (Massey bound [2]). The proof of this statement is left as a problem.

As we pointed out earlier in this section, one-step majority-logic decoding is most effective for cyclic codes that are completely orthogonalizable. Unfortunately, there exist very few good cyclic codes in this category. The double-error-correcting $(15, 7)$ code considered in Example 8.1 is the only known BCH code that is completely orthogonalizable in one step. Several small classes of one-step majority-logic decodable cyclic codes are presented in the next two sections. Two of the classes are proved to be completely orthogonalizable.

8.2 A CLASS OF ONE-STEP MAJORITY-LOGIC DECODABLE CODES

In this section we present a class of one-step majority-logic decodable cyclic codes whose construction is based on a certain symmetry property.

Let C be an (n, k) cyclic code generated by $\mathbf{g}(X)$, where $n = 2^m - 1$. We may extend each vector $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ in C by adding an *overall parity-check digit*, denoted by v_∞ , to its left. The overall parity-check digit v_∞ is defined as the modulo-2 sum of all the digits of \mathbf{v} (i.e., $v_\infty = v_0 + v_1 + \dots + v_{n-1}$). Adding v_∞ to \mathbf{v} results in the following vector of $n + 1 = 2^m$ components:

$$\mathbf{v}_e = (v_\infty, v_0, v_1, \dots, v_{n-1}).$$

The overall parity-check digit is 1 if the weight of \mathbf{v} is odd, and it is 0 if the weight of \mathbf{v} is even. The 2^k extended vectors form an $(n + 1, k)$ linear code, denoted by C_e , which is called an *extension* of C . Clearly, the codewords of C_e have even weight.

Let α be a primitive element in the Galois field $GF(2^m)$. We may number the components of a vector $\mathbf{v}_e = (v_\infty, v_0, v_1, \dots, v_{2^m-2})$ in C_e by the elements of $GF(2^m)$ as follows: the component v_∞ is numbered $\alpha^\infty = 0$, the component v_0 is numbered $\alpha = 1$, and for $1 \leq i < 2^m - 1$, the component v_i is numbered α^i . We

call these numbers the *location numbers*. Let Y denote the location of a component of \mathbf{v}_e . Consider a permutation that carries the component of \mathbf{v}_e at the location Y to the location $Z = aY + b$, where a and b are elements from the field of $GF(2^m)$, and $a \neq 0$. This permutation is called an *affine permutation* [14]. Application of an affine permutation to a vector of 2^m components results in another vector of 2^m components.

EXAMPLE 8.3

Consider the following vector of 16 components, which are numbered with the elements of $GF(2^4)$ (using Table 2.8):

$$\begin{array}{cccccccccccccccc} \alpha^\infty & \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\ (1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0) \end{array}$$

Now, we apply the affine permutation

$$Z = \alpha Y + \alpha^{14}$$

to the components of the preceding vector. The resultant vector is

$$\begin{array}{cccccccccccccccc} \alpha^\infty & \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\ (0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1) \end{array}$$

For example, the component at the location $Y = \alpha^8$ is carried to the location

$$Z = \alpha \cdot \alpha^8 + \alpha^{14} = \alpha^9 + \alpha^{14} = \alpha^4.$$

An extended cyclic code C_e of length 2^m is said to be *invariant* under the group of affine permutations if every affine permutation carries every codeword in C_e into another codeword in C_e . In the following discussion we state a necessary and sufficient condition for an extended cyclic code of length 2^m to be invariant under the affine permutations.

Let h be a nonnegative integer less than 2^m . The radix-2 (binary) expansion of h is

$$h = \delta_0 + \delta_1 2 + \delta_2 2^2 + \cdots + \delta_{m-1} 2^{m-1},$$

where $\delta_i = 0$ or 1 for $0 \leq i < m$. Let h' be another nonnegative integer less than 2^m whose radix-2 expansion is

$$h' = \delta'_0 + \delta'_1 2 + \delta'_2 2^2 + \cdots + \delta'_{m-1} 2^{m-1}.$$

The integer h' is said to be a *descendant* of h if $\delta'_i \leq \delta_i$ for $0 \leq i < m$.

EXAMPLE 8.4

Let $m = 5$. The integer 21 has the following radix-2 expansion:

$$21 = 1 + 0 \cdot 2 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4.$$

The following integers are proper descendants of 21:

$$\begin{aligned} 20 &= 0 + 0 \cdot 2 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4, \\ 17 &= 1 + 0 \cdot 2 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4, \\ 16 &= 0 + 0 \cdot 2 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4, \\ 5 &= 1 + 0 \cdot 2 + 1 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4, \\ 4 &= 0 + 0 \cdot 2 + 1 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4, \\ 1 &= 1 + 0 \cdot 2 + 0 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4, \\ 0 &= 0 + 0 \cdot 2 + 0 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4. \end{aligned}$$

Let $\Delta(h)$ denote the set of all nonzero proper descendants of h . The following theorem characterizes a necessary and sufficient condition for the extension C_e of a cyclic code C of length $2^m - 1$ to be invariant under the affine group of permutations.

THEOREM 8.2 [4, 5] Let C be a cyclic code of length $n = 2^m - 1$ generated by $\mathbf{g}(X)$. Let C_e be the extended code obtained from C by appending an overall parity-check digit. Let α be a primitive element of the Galois field $GF(2^m)$. Then, the extended code C_e is invariant under the affine permutations if and only if for every α^h that is a root of the generator polynomial $\mathbf{g}(X)$ of C and for every h' in $\Delta(h)$, $\alpha^{h'}$ is also a root of $\mathbf{g}(X)$, and $\alpha^0 = 1$ is not a root of $\mathbf{g}(X)$.

The proof of this theorem is omitted here. For a proof, the reader is referred to [4] and [5]. A cyclic code of length $2^m - 1$ whose generator polynomial satisfies the conditions given in Theorem 8.2 is said to have the *doubly transitive invariant (DTI) property*. RM codes and extended primitive BCH codes are invariant under the affine permutations [4, 5].

Given a code C_e of length $n = 2^m$ that is invariant under the affine permutations, the code C obtained by deleting the first digit from each vector of C_e is cyclic. To see this, we apply the permutation $Z = \alpha Y$ to a vector $(v_\infty, v_0, v_1, \dots, v_{2^m-2})$ in C_e . This permutation keeps the component v_∞ at the same location α^∞ but *cyclically shifts* the other $2^m - 1$ components one place to the right. The resultant vector is

$$(v_\infty, v_{2^m-2}, v_0, v_1, \dots, v_{2^m-3}),$$

which is also in C_e . Clearly, if we delete v_∞ from each vector of C_e , we obtain a cyclic code of length $2^m - 1$.

Now, we are ready to present a class of one-step majority-logic decodable codes whose dual codes have the DTI property. Let J and L be two factors of $2^m - 1$ such that $J \cdot L = 2^m - 1$. Clearly, both J and L are odd. The polynomial $X^{2^m-1} + 1$ can be factored as follows:

$$X^{2^m-1} + 1 = (1 + X^J)(1 + X^J + X^{2J} + \dots + X^{(L-1)J}).$$

Let

$$\pi(X) = 1 + X^J + X^{2J} + \dots + X^{(L-1)J}. \quad (8.12)$$

Section 8.2 A Class of One-Step Majority-

From Theorem 2.12 we know that the $2^m - 1$ nonzero elements of $GF(2^m)$ form the $2^m - 1$ roots of $X^{2^m-1} + 1$. Let α be a primitive element of $GF(2^m)$. Because $(\alpha^L)^J = \alpha^{2^m-1} = 1$, the polynomial $X^J + 1$ has $\alpha^0 = 1, \alpha^L, \alpha^{2L}, \dots, \alpha^{(J-1)L}$ as all its roots. Therefore, the polynomial $\pi(X)$ has α^h as a root if and only if h is not a multiple of L , and $0 < h < 2^m - 1$.

Now, we form a polynomial $H(X)$ over $GF(2)$ as follows: $H(X)$ has α^h as a root if and only if (1) α^h is a root of $\pi(X)$ and (2) for every h' in $\Delta(h)$, $\alpha^{h'}$ is also a root of $\pi(X)$. Let α^i be a root of $H(X)$. Let $\phi_i(X)$ be the minimal polynomial of α^i . Then,

$$H(X) = \text{LCM}\{\text{minimal polynomials } \phi_i(X) \text{ of the roots of } H(X)\}. \quad (8.13)$$

It is clear that $H(X)$ divides $\pi(X)$ and is a factor of $X^{2^m-1} + 1$. Let C' be the cyclic code of length $2^m - 1$ generated by $H(X)$. It follows from Theorem 8.2 that C' has the DTI property. Thus, the extended code C'_e of C' is invariant under the group of affine permutations. Let C be the dual code of C' . Then, C is also cyclic. Since $H(X)$ divides $X^{2^m-1} + 1$, we have

$$X^{2^m-1} + 1 = G(X)H(H).$$

Let k be the degree of $H(X)$. Then, the degree of $G(X)$ is $2^m - 1 - k$. The generator polynomial of C is

$$\mathbf{g}(X) = X^{2^m-k-1}G(X^{-1}), \quad (8.14)$$

which is the reciprocal of $G(X)$. Next, we will show that the code C is one-step majority-logic decodable and is capable of correcting $t_{ML} = \lfloor J/2 \rfloor$ or fewer errors where $J = (2^m - 1)/L$.

First, we need to determine J vectors from C' (the dual of C) that are orthogonal on the digit at location α^{2^m-2} . Because $\pi(X)$ is a multiple of $H(X)$ and has degree less than $2^m - 1$, it is a code polynomial in C' generated by $H(X)$. Clearly, the polynomials $X\pi(X), X^2\pi(X), \dots, X^{J-1}\pi(X)$ are also code polynomials in C' . From (8.12) we see that, for $i \neq j$, $X^i\pi(X)$ and $X^j\pi(X)$ do not have any common term. Let $\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_{J-1}$ be the J corresponding codewords of $\pi(X), X\pi(X), \dots, X^{J-1}\pi(X)$. The weight of each of these vectors is L . Adding an overall parity-check digit to each of these vectors, we obtain J vectors $\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{J-1}$ of length 2^m that are codewords in the extension C'_e of C' . Since L is odd, the overall parity-check digit of each \mathbf{u}_i is 1. Thus, the J vectors $\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{J-1}$ have the following properties:

1. They all have 1 at location α^∞ (the overall parity-check digit position).
2. One and only one vector has a 1 at the location α^j for $0 \leq j < 2^m - 1$.

Therefore, they form J vectors orthogonal on the digit at location α^∞ . Now, we apply the affine permutation

$$Z = \alpha Y + \alpha^{2^m-2}$$

to $\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{J-1}$. This permutation carries $\mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{J-1}$ into J vectors $\mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_{J-1}$, which are also in C'_e (since C'_e is invariant under the group of affine permutations). Note that the permutation $Z = \alpha Y + \alpha^{2^m-2}$ carries the component of \mathbf{u}_i at location α^∞ to location α^{2^m-2} . Thus, the vectors $\mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_{J-1}$ are

orthogonal on the digit at location α^{2^m-2} . Deleting the digit at location α^∞ from $\mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_{J-1}$, we obtain J vectors $\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_{J-1}$ of length $2^m - 1$, which are vectors in C' and are orthogonal on the digit at location α^{2^m-2} . From these J vectors we can form J parity-check sums orthogonal on the error digit e_{2^m-2} . Therefore, the cyclic code C generated by

$$\mathbf{g}(X) = X^{2^m - k - 1} G(X^{-1})$$

is one-step majority-logic decodable and is capable of correcting $t_{ML} = \lfloor J/2 \rfloor$ or fewer errors. For convenience, we call this code a *type-0* one-step majority-logic decodable DTI code.

EXAMPLE 8.5

Let $m = 5$. We can factor the polynomial $X^{2^4-1} + 1 = X^{15} + 1$ as

$$X^{15} + 1 = (1 + X^5)(1 + X^5 + X^{10}).$$

Thus, $J = 5$, $L = 3$, and $\pi(X) = 1 + X^5 + X^{10}$. Let α be a primitive element in $GF(2^4)$ (use Table 2.8) whose minimal polynomial is $\phi_1(X) = 1 + X + X^4$. Because $\alpha^{15} = 1$, the polynomial $1 + X^5$ has $1, \alpha^3, \alpha^6, \alpha^9$, and α^{12} as all its roots. The polynomial $\pi(X)$ has $\alpha, \alpha^2, \alpha^4, \alpha^5, \alpha^7, \alpha^8, \alpha^{10}, \alpha^{11}, \alpha^{13}$ and α^{14} as roots. Next, we determine the polynomial $H(X)$. From the conditions on the roots of $H(X)$, we find that $H(X)$ has $\alpha, \alpha^2, \alpha^4, \alpha^5, \alpha^8$, and α^{10} as its roots. The roots $\alpha, \alpha^2, \alpha^4$, and α^8 are conjugates, and they have the same minimal polynomial, $\phi_1(X) = 1 + X + X^4$. The roots α^5 and α^{10} are conjugates, and they have $\phi_5(X) = 1 + X + X^2$ as their minimal polynomial. Hence,

$$H(X) = \phi_1(X)\phi_5(X) = (1 + X + X^4)(1 + X + X^2) \\ = 1 + X^3 + X^4 + X^5 + X^6.$$

We can easily check that $H(X)$ divides $\pi(X)$, and, in fact, $\pi(X) = (1 + X^3 + X^4)H(X)$. Also, $H(X)$ divides $X^{15} + 1$, and $X^{15} + 1 = (1 + X^3 + X^4 + X^5 + X^8 + X^9)H(X)$. Thus, $G(X) = 1 + X^3 + X^4 + X^5 + X^8 + X^9$. The polynomial $H(X)$ generates a $(15, 9)$ cyclic code C' , which has the DTI property. The polynomials $\pi(X)$, $X\pi(X)$, $X^2\pi(X)$, $X^3\pi(X)$, and $X^4\pi(X)$ are code polynomials in C' . The dual code of C' , C , is generated by

$$g(X) = X^9 G(X^{-1}) = 1 + X + X^4 + X^5 + X^6 + X^9.$$

Thus, C is a $(15, 6)$ cyclic code.

To decode C , we need to determine parity-check sums orthogonal on e_{14} . The vectors corresponding to $\pi(X)$, $X\pi(X)$, $X^2\pi(X)$, $X^3\pi(X)$, and $X^4\pi(X)$ are

Location Numbers

which are codewords in C' . Adding an overall parity-check digit to these vectors, we obtain the following vectors:

Location Numbers

α^∞	α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
$\mathbf{u}_0 = (1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0)$															
$\mathbf{u}_1 = (1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0)$															
$\mathbf{u}_2 = (1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0)$															
$\mathbf{u}_3 = (1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1)$															
$\mathbf{u}_4 = (1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1)$															

which are vectors in C'_e (the extension of C'). Now, we apply the affine permutation $Z = \alpha Y + \alpha^{14}$ to permute the components of $\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$, and \mathbf{u}_4 . The permutation results in the following vectors:

Location Numbers

α^∞	α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
$\mathbf{z}_0 = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0)$															
$\mathbf{z}_1 = (0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)$															
$\mathbf{z}_2 = (0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)$															
$\mathbf{z}_3 = (1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1)$															
$\mathbf{z}_4 = (0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0)$															

which are also in C'_e . Deleting the overall parity-check digits from these vectors, we obtain the following vectors in C' :

Location Numbers

α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	
$\mathbf{w}_0 = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1)$															
$\mathbf{w}_1 = (0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)$															
$\mathbf{w}_2 = (1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)$															
$\mathbf{w}_3 = (0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1)$															
$\mathbf{w}_4 = (0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1)$															

We see that these vectors are orthogonal on the digit at location α^{14} .

Let $\mathbf{r} = (r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14})$ be the received vector. Then, the parity-check sums orthogonal on e_{14} are

$$\begin{aligned} A_0 &= \mathbf{r} \cdot \mathbf{w}_0 = r_7 + r_8 + r_{10} + r_{14}, \\ A_1 &= \mathbf{r} \cdot \mathbf{w}_1 = r_1 + r_5 + r_{13} + r_{14}, \\ A_2 &= \mathbf{r} \cdot \mathbf{w}_2 = r_0 + r_2 + r_6 + r_{14}, \\ A_3 &= \mathbf{r} \cdot \mathbf{w}_3 = r_4 + r_9 + r_{14}, \\ A_4 &= \mathbf{r} \cdot \mathbf{w}_4 = r_3 + r_{11} + r_{12} + r_{14}. \end{aligned}$$

Therefore, the $(15, 6)$ cyclic code C generated by $\mathbf{g}(X) = 1 + X + X^4 + X^5 + X^6 + X^9$ is one-step majority-logic decodable and is capable of correcting $t_{ML} = \lfloor 5/2 \rfloor = 2$ or fewer errors. It also corrects many error patterns of three errors. The code has a minimum distance of at least $J + 1 = 5 + 1 = 6$; however, the minimum distance of the code is exactly 6. Hence, the code is completely orthogonalizable.

Recall that $\pi(X)$ has α^h as a root if and only if h is not a multiple of L , and $0 < h < 2^m - 1$. Therefore, $\pi(X)$ has the following consecutive powers of α as roots: $\alpha, \alpha^2, \dots, \alpha^{L-1}$. Because any descendant h' of an integer h is less than h , if $h < L$ and h' in $\Delta(h)$, both α^h and $\alpha^{h'}$ are roots of $\pi(X)$. Consequently, the polynomial $H(X)$ also has $\alpha, \alpha^2, \dots, \alpha^{L-1}$ as roots. Using the argument that proves the minimum distance of a BCH code, we can show that the minimum distance of C' generated by $H(X)$ is at least L ; however, since $\pi(X)$ is a code polynomial of weight L in C' , the minimum distance of C' is exactly L . It follows from Theorem 8.1 that the number of parity-check sums orthogonal on an error digit that can be formed for C is upper bounded by

$$\left\lfloor \frac{2^m - 2}{L - 1} \right\rfloor; \quad (8.15)$$

however, $J = (2^m - 1)/L$. Therefore, for large L , J is either equal to or close to the upper bound of (8.15).

In general, it is not known whether the type-0 DTI codes are completely orthogonalizable. There are a number of special cases for which we can prove that the codes are completely orthogonalizable.

The type-0 DTI codes may be modified so that the resultant codes are also one-step majority-logic decodable, and $J - 1$ parity-check sums orthogonal on an error digit can be formed. Recall that the polynomial $H(X)$ does not have $(X + 1)$ as a factor (i.e., it does not have $\alpha^0 = 1$ as a root). Let

$$H_1(X) = (X + 1)H(X). \quad (8.16)$$

The cyclic code C'_1 generated by $H_1(X)$ is a subcode of C' generated by $H(X)$. In fact, C'_1 consists of the *even-weight* vectors of C' as codewords. Recall that the J orthogonal vectors $\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_{J-1}$ in C' are obtained from the vectors $\mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_{J-1}$ by deleting the digit at location α^∞ . Because $\mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_{J-1}$ are orthogonal on the digit at location $\alpha^{2^m - 2}$, there is one and only one vector \mathbf{z}_i that has a 1 at location α^∞ . Since $\mathbf{z}_0, \mathbf{z}_1, \dots, \mathbf{z}_{J-1}$ all have weight $L + 1$ which is even, all but one of the orthogonal vectors $\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_{J-1}$ have weight $L + 1$. These $J - 1$ even-weight orthogonal vectors are in C'_1 . Therefore, the dual code of C'_1 , denoted by C_1 , is one-step majority-logic decodable, and $J - 1$ parity-check sums orthogonal on an error digit can be formed. Let

$$G_1(X) = \frac{G(X)}{X + 1}. \quad (8.17)$$

Then, the generator polynomial for C_1 is

$$\mathbf{g}_1(X) = X^{2^m - k - 2} G_1(X^{-1}) = \frac{\mathbf{g}(X)}{X + 1}, \quad (8.18)$$

where $\mathbf{g}(X)$ is given by (8.14). C_1 is called a *type-1* DTI code, and its dimension is one greater than that of its corresponding type-0 DTI code.

EXAMPLE 8.6

For $m = 4$ and $J = 5$, the type-1 DTI code C_1 that corresponds to the type-0 DTI code given in Example 8.5 is generated by

$$\begin{aligned}\mathbf{g}_1(X) &= \frac{1 + X + X^4 + X^5 + X^6 + X^9}{1 + X} \\ &= 1 + X^4 + X^6 + X^7 + X^8.\end{aligned}$$

It is interesting to note that this code is the $(15, 7)$ BCH code. From Example 8.5 we see that \mathbf{w}_3 has odd weight, and therefore it is not a vector in C'_1 (the dual of C_1). Hence, the four orthogonal vectors in C'_1 are

$$\begin{aligned}\mathbf{w}_0 &= (0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1), \\ \mathbf{w}_1 &= (0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1), \\ \mathbf{w}_2 &= (1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1), \\ \mathbf{w}_4 &= (0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1),\end{aligned}$$

which are the same four orthogonal vectors given in Example 8.1.

Because the dual code of type-1 DTI code C_1 has a minimum distance of $L + 1$, the number of parity-check sums orthogonal on an error digit that can be formed is upper bounded by

$$\left\lfloor \frac{2^m - 2}{L} \right\rfloor = \left\lfloor \frac{2^m - 1}{L} - \frac{1}{L} \right\rfloor = \left\lfloor J - \frac{1}{L} \right\rfloor = J - 1.$$

Therefore, the number of parity-check sums orthogonal on an error digit that can be formed for a type-1 DTI code is equal to its upper bound. Since J is odd, $\lfloor J/2 \rfloor = \lfloor (J - 1)/2 \rfloor$. Thus, both type-0 and type-1 DTI codes have the same majority-logic error-correcting capability.

In general, there is no simple formula for enumerating the number of parity-check digits of the one-step majority-logic decodable DTI codes (type-0 or type-1); however, for two special cases, exact formulas for $n - k$ can be obtained [6]:

Case I. For $m = 2sl$ and $J = 2^l + 1$, the number of parity-check digits of the type-1 DTI code of length $2^m - 1$ is

$$n - k = (2^{s+1} - 1)^l - 1.$$

Case II. For $m = \lambda l$ and $J = 2^l - 1$, the number of parity-check digits of the type-1 DTI code of length $2^m - 1$ is

$$n - k = 2^m - (2^\lambda - 1)^l - 1.$$

A list of one-step majority-logic decodable type-1 DTI codes is given in Table 8.1.

Short-length DTI codes are comparable with BCH codes in efficiency. For example, there exists a $(63, 37)$ one-step majority-logic decodable type-1 DTI code that is capable of correcting four or fewer errors. The corresponding four-error-correcting BCH code of the same length is a $(63, 39)$ code that has two information

TABLE 8.1: Some one-step majority-logic decodable type-1 DTI codes.

n	k	t_{ML}	n	k	t_{ML}
15	9	1	2047	1211	11
	7	2		573	44
63	49	1	4095	3969	1
	37	4		3871	2
	13	10		3753	4
255	225	1		3611	6
	207	2		3367	32
	175	8		2707	17
	37	25		2262	19
	21	42		2074	22
511	343	3		1649	45
	139	36		1393	52
1023	961	1		1377	136
	833	5		406	292
	781	16		101	409
	151	46		43	682
	30	170			

digits more than the (63, 37) type-1 DTI code; however, the decoding circuit for the (63, 39) BCH code is much more complex than for the (63, 37) DTI code. For large block lengths, the DTI codes are much less efficient than the BCH codes of the same length and the same error-correcting capability.

8.3 OTHER ONE-STEP MAJORITY-LOGIC DECODABLE CODES

There are two other small classes of one-step majority-logic decodable cyclic codes: the maximum-length codes and the difference-set codes. Both classes have been proved to be completely orthogonalizable.

8.3.1 Maximum-Length Codes

For any integer $m \geq 3$, there exists a nontrivial maximum-length code with the following parameters:

$$\begin{aligned} \text{Block length:} & \quad n = 2^m - 1, \\ \text{Number of information digits:} & \quad k = m, \\ \text{Minimum distance:} & \quad d = 2^{m-1}. \end{aligned}$$

The generator polynomial of this code is

$$\mathbf{g}(X) = \frac{X^n + 1}{\mathbf{p}(X)}, \quad (8.19)$$

where $\mathbf{p}(X)$ is a primitive polynomial of degree m . This code consists of the all-zero codeword and $2^m - 1$ codewords of weight 2^{m-1} (see Problem 8.11). Maximum-length

codes were first shown to be majority-logic decodable by Yale [7] and Zierler [8] independently. The dual code of the maximum-length code is a $(2^m - 1, 2^m - m - 1)$ cyclic code generated by the reciprocal of the parity polynomial $\mathbf{p}(X)$,

$$\mathbf{p}^*(X) = X^m \mathbf{p}(X^{-1}).$$

Because $\mathbf{p}^*(X)$ is also a primitive polynomial of degree m , the dual code is thus a Hamming code. Therefore, the null space of the maximum-length code contains vectors of weight 3 (this is the minimum weight). Now, consider the following set of distinct code polynomials:

$$Q = \{\mathbf{w}(X) = X^i + X^j + X^{n-1} : 0 \leq i < j < n - 1\} \quad (8.20)$$

in the Hamming code generated by $\mathbf{p}^*(X)$. No two polynomials in Q can have any common terms except the term X^{n-1} . Otherwise, the sum of these two polynomials would be a code polynomial of only two terms in the Hamming code. This is impossible, since the minimum weight of a Hamming code is 3. Therefore, the set Q contains polynomials orthogonal on the highest-order-digit position X^{n-1} . To find $\mathbf{w}(X)$, we start with a polynomial $X^{n-1} + X^j$ for $0 \leq j < n - 1$ and then determine X^i such that $X^{n-1} + X^j + X^i$ is divisible by $\mathbf{p}^*(X)$, as follows: Divide $X^{n-1} + X^j$ by $\mathbf{p}^*(X)$ step-by-step with long division until a single term X^i appears at the end of a certain step. Then, $\mathbf{w}(X) = X^{n-1} + X^j + X^i$ is a polynomial orthogonal on digit position X^{n-1} . Clearly, if we started with $X^{n-1} + X^i$, we would obtain the same polynomial $\mathbf{w}(X)$. Thus, we can find $(n - 1)/2 = 2^{m-1} - 1$ polynomials orthogonal on digit position X^{n-1} . That is, $J = 2^{m-1} - 1$ parity-check sums orthogonal on e_{n-1} can be formed. Because the maximum-length code generated by $\mathbf{g}(X)$ of (8.19) has a minimum distance of exactly 2^{m-1} , it is completely orthogonalizable. The code is capable of correcting $t_{ML} = 2^{m-2} - 1$ or fewer errors with one-step majority-logic decoding.

EXAMPLE 8.7

Consider the maximum-length code with $m = 4$ and parity polynomial $\mathbf{p}(X) = 1 + X + X^4$. This code has block length $n = 15$ and minimum distance $d = 8$. The generator polynomial of this code is

$$\begin{aligned} \mathbf{g}(X) &= \frac{X^{15} + 1}{\mathbf{p}(X)} \\ &= 1 + X + X^2 + X^3 + X^5 + X^7 + X^8 + X^{11}. \end{aligned}$$

It is a $(15, 4)$ code. The null space of this code is generated by

$$\mathbf{p}^*(X) = X^4 \mathbf{p}(X^{-1}) = X^4 + X^3 + 1.$$

We divide $X^{14} + X^{13}$ by $\mathbf{p}^*(X) = X^4 + X^3 + 1$ with long division as follows:

$$\begin{array}{r} X^{10} \\ \hline X^4 + X^3 + 1 | X^{14} + X^{13} \\ \hline X^{14} + X^{13} + X^{10} \\ \hline X^{10} \text{ (stop).} \end{array}$$

A single term, X^{10} , appears at the end of the first step of the long division. Then, $\mathbf{w}_1(X) = X^{14} + X^{13} + X^{10}$ is a polynomial orthogonal on X^{14} . Now, we divide $X^{14} + X^{12}$ by $\mathbf{p}^*(X)$:

$$\begin{array}{r}
 \begin{array}{c} X^{10} + X^9 + X^6 \\ | \\ X^{14} + X^{12} \\ X^{14} + X^{13} \\ \hline X^{13} + X^{12} + X^{10} \\ X^{13} + X^{12} \\ \hline X^{10} + X^9 \\ X^{10} + X^9 + X^6 \\ \hline X^6 \end{array} \\
 X^4 + X^3 + 1 \quad | \quad X^{14} + X^{12} + X^{10} + X^9
 \end{array}$$

(stop).

Then, $\mathbf{w}_2(X) = X^{14} + X^{12} + X^6$ is another polynomial orthogonal on X^{14} . The rest of the polynomials orthogonal on X^{14} can be found in the same manner; they are

$$\begin{aligned}
 \mathbf{w}_3(X) &= 1 + X^{11} + X^{14}, & \mathbf{w}_4(X) &= X^4 + X^9 + X^{14}, \\
 \mathbf{w}_5(X) &= X + X^8 + X^{14}, & \mathbf{w}_6(X) &= X^5 + X^7 + X^{14}, \\
 \mathbf{w}_7(X) &= X^2 + X^3 + X^{14}.
 \end{aligned}$$

From the set of polynomials orthogonal on X^{14} , we obtain the following seven parity-check sums orthogonal on e_{14} :

$$\begin{aligned}
 A_1 &= e_{10} + e_{13} + e_{14}, \\
 A_2 &= e_6 + e_{12} + e_{14}, \\
 A_3 &= e_0 + e_{11} + e_{14}, \\
 A_4 &= e_4 + e_9 + e_{14}, \\
 A_5 &= e_1 + e_8 + e_{14}, \\
 A_6 &= e_5 + e_7 + e_{14}, \\
 A_7 &= e_2 + e_3 + e_{14}.
 \end{aligned}$$

In terms of syndrome bits, we have $A_1 = s_{10}$, $A_2 = s_6$, $A_3 = s_0$, $A_4 = s_4 + s_9$, $A_5 = s_1 + s_8$, $A_6 = s_5 + s_7$, and $A_7 = s_2 + s_3$. The code is capable of correcting three or fewer errors by one-step majority-logic decoding.

8.3.2 Difference-Set Codes

The formulation of difference-set codes is based on the construction of a *perfect difference set*. Let $P = \{l_0, l_1, l_2, \dots, l_q\}$ be a set of $q + 1$ nonnegative integers such that

$$0 \leq l_0 < l_1 < l_2 < \dots < l_q \leq q(q + 1).$$

From this set of integers it is possible to form $q(q + 1)$ ordered differences as follows:

$$D = \{l_j - l_i : j \neq i\}.$$

Obviously, half the differences in D are positive, and the other half are negative. The set P is said to be a *perfect simple difference set of order q* if and only if it has the following properties:

1. All the positive differences in D are distinct.
2. All the negative differences in D are distinct.
3. If $l_j - l_i$ is a negative difference in D , then $q(q+1) + 1 + (l_j - l_i)$ is not equal to any positive difference in D .

Clearly, it follows from the definition that $P' = \{0, l_1 - l_0, l_2 - l_0, \dots, l_q - l_0\}$ is also a perfect simple difference set.

EXAMPLE 8.8

Consider the set $P = \{0, 2, 7, 8, 11\}$ with $q = 4$. The $4 \cdot 5 = 20$ ordered differences are

$$D = \{2, 7, 8, 11, 5, 6, 9, 1, 4, 3, -2, -7, -8, -11, -5, -6, -9, -1, -4, -3\}.$$

It can be checked easily that P satisfies all three properties of a perfect simple difference set.

Singer [9] has constructed perfect difference sets for order $q = p^s$, where p is a prime and s is any positive integer (see also [10]). In the following discussion we shall be concerned only with $q = 2^s$.

Let $P = \{l_0 = 0, l_1, l_2, \dots, l_{2^s}\}$ be a perfect simple difference set of order 2^s . We define the polynomial

$$\mathbf{z}(X) = 1 + X^{l_1} + X^{l_2} + \dots + X^{l_{2^s}}. \quad (8.21)$$

Let $n = 2^s(2^s + 1) + 1$ and $\mathbf{h}(X)$ be the greatest common divisor of $\mathbf{z}(X)$ and $X^n + 1$; that is,

$$\begin{aligned} \mathbf{h}(X) &= \text{GCD}\{\mathbf{z}(X), X^n + 1\} \\ &= 1 + h_1X + h_2X^2 + \dots + h_{k-1}X^{k-1} + X^k. \end{aligned} \quad (8.22)$$

Then a difference-set code of length n is defined as the cyclic code generated by

$$\begin{aligned} \mathbf{g}(X) &= \frac{X^n + 1}{\mathbf{h}(X)} \\ &= 1 + g_1X + g_2X^2 + \dots + X^{n-k}. \end{aligned} \quad (8.23)$$

This code has the following parameters:

Code length: $n = 2^{2s} + 2^s + 1$

Number of parity-check digits: $n - k = 3^s + 1$

Minimum distance: $d = 2^s + 2$.

Difference-set codes were discovered by Rudolph [11] and Weldon [12] independently. The formula for the number of parity-check digits was derived by Graham and MacWilliams [13].

EXAMPLE 8.9

In Example 8.8 we showed that the set $P = \{0, 2, 7, 8, 11\}$ is a perfect simple difference set of order $q = 2^2$. Let $\mathbf{z}(X) = 1 + X^2 + X^7 + X^8 + X^{11}$. Then,

$$\begin{aligned}\mathbf{h}(X) &= \text{GCD}\{1 + X^2 + X^7 + X^8 + X^{11}, 1 + X^{21}\} \\ &= 1 + X^2 + X^7 + X^8 + X^{11}.\end{aligned}$$

The generator polynomial of the difference-set code of length $n = 21$ is

$$\begin{aligned}\mathbf{g}(X) &= \frac{X^{21} + 1}{\mathbf{h}(X)} \\ &= 1 + X^2 + X^4 + X^6 + X^7 + X^{10}.\end{aligned}$$

Thus, the code is a $(21, 11)$ cyclic code.

Let $\mathbf{h}^*(X) = X^k \mathbf{h}(X^{-1})$ be the reciprocal polynomial of $\mathbf{h}(X)$. Then, the $(n, n - k)$ cyclic code generated by $\mathbf{h}^*(X)$ is the null space of the difference-set code generated by $\mathbf{g}(X)$ of (8.23). Let

$$\begin{aligned}\mathbf{z}^*(X) &= X^{l_{2^s}} \mathbf{z}(X^{-1}) \\ &= 1 + \cdots + X^{l_{2^s} - l_2} + X^{l_{2^s} - l_1} + X^{l_{2^s}}.\end{aligned}\tag{8.24}$$

Because $\mathbf{z}(X)$ is divisible by $\mathbf{h}(X)$, $\mathbf{z}^*(X)$ is divisible by $\mathbf{h}^*(X)$. Thus, $\mathbf{z}^*(X)$ is in the null space of the difference-set code generated by $\mathbf{g}(X)$ of (8.23). Let

$$\begin{aligned}\mathbf{w}_0(X) &= X^{n-1-l_{2^s}} \mathbf{z}^*(X) \\ &= X^{n-1-l_{2^s}} + \cdots + X^{n-1-l_2} + X^{n-1-l_1} + X^{n-1}.\end{aligned}$$

Obviously, $\mathbf{w}_0(X)$ is divisible by $\mathbf{h}^*(X)$ and is also in the null space of the difference-set code generated by $\mathbf{g}(X)$ of (8.23). Now, let

$$\begin{aligned}\mathbf{w}_i(X) &= X^{l_i - l_{i-1} - 1} + X^{l_i - l_{i-2} - 1} + \cdots + X^{l_i - l_1 - 1} + X^{l_i - 1} \\ &\quad + X^{n-1-l_{2^s}+l_i} + X^{n-1-l_{2^s-1}+l_i} + \cdots + X^{n-1}\end{aligned}\tag{8.25}$$

be the vector obtained by shifting $\mathbf{w}_0(X)$ cyclically to the right l_i times. Because $\{l_0 = 0, l_1, l_2, \dots, l_{2^s}\}$ is a perfect difference set, no two polynomials $\mathbf{w}_i(X)$ and $\mathbf{w}_j(X)$ for $i \neq j$ can have any common term except X^{n-1} . Thus, $\mathbf{w}_0(X), \mathbf{w}_1(X), \dots, \mathbf{w}_{2^s}(X)$ form a set of $J = 2^s + 1$ polynomials orthogonal on the digit at position X^{n-1} . Since the code generated by $\mathbf{g}(X)$ of (8.23) is proved to have a minimum distance of $2^s + 2$, it is completely orthogonalizable and is capable of correcting $t_{ML} = 2^{s-1}$ or fewer errors.

EXAMPLE 8.10

Consider the code given in Example 8.9, which is specified by the perfect difference set $P = \{0, 2, 7, 8, 11\}$ of order 2^2 . Thus, we have

$$\mathbf{z}^*(X) = X^{11} \mathbf{z}(X^{-1}) = 1 + X^3 + X^4 + X^9 + X^{11}$$

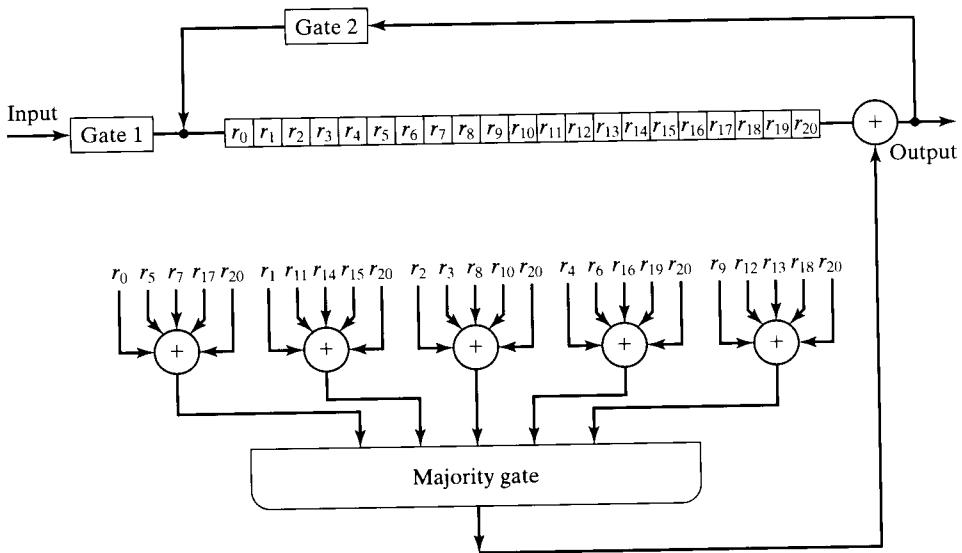


FIGURE 8.5: Type-II majority-logic decoder for the (21, 11) difference-set code.

and

$$\mathbf{w}_0(X) = X^9 \mathbf{z}^*(X) = X^9 + X^{12} + X^{13} + X^{18} + X^{20}.$$

By shifting $\mathbf{w}_0(X)$ cyclically to the right 2 times, 7 times, 8 times, and 11 times, we obtain

$$\begin{aligned}\mathbf{w}_1(X) &= X + X^{11} + X^{14} + X^{15} + X^{20}, \\ \mathbf{w}_2(X) &= X^4 + X^6 + X^{16} + X^{19} + X^{20}, \\ \mathbf{w}_3(X) &= 1 + X^5 + X^7 + X^{17} + X^{20}, \\ \mathbf{w}_4(X) &= X^2 + X^3 + X^8 + X^{10} + X^{20}.\end{aligned}$$

Clearly, $\mathbf{w}_0(X)$, $\mathbf{w}_1(X)$, $\mathbf{w}_2(X)$, $\mathbf{w}_3(X)$, and $\mathbf{w}_4(X)$ are five polynomials orthogonal on X^{20} . From these five orthogonal polynomials, we can form the following five parity-check sums orthogonal on e_{20} :

$$\begin{aligned}A_1 &= s_9 &= e_9 + e_{12} + e_{13} + e_{18} + e_{20}, \\ A_2 &= s_1 &= e_1 + e_{11} + e_{14} + e_{15} + e_{20}, \\ A_3 &= s_4 + s_6 &= e_4 + e_6 + e_{16} + e_{19} + e_{20}, \\ A_4 &= s_0 + s_5 + s_7 &= e_0 + e_5 + e_7 + e_{17} + e_{20}, \\ A_5 &= s_2 + s_3 + s_8 &= e_2 + e_3 + e_8 + e_{10} + e_{20}.\end{aligned}$$

A type-II majority-logic decoder for this code is shown in Figure 8.5. The construction of a type-I decoder for this code is left as an exercise.

Difference-set codes are nearly as powerful as the best known cyclic codes in the range of practical interest. Unfortunately, there are relatively few codes with useful parameters in this class. A list of the first few codes with their generator

TABLE 8.2: A list of binary difference-set cyclic codes.

s	n	k	d	t	Generator polynomial, $g(X)^*$	Associated difference set
1	7	3	4	1	0, 2, 3, 4	0, 2, 3
2	21	11	6	2	0, 2, 4, 6, 7, 10	0, 2, 7, 8, 11
3	73	45	10	4	0, 2, 4, 6, 8, 12, 16, 22, 25, 28	0, 2, 10, 24, 25, 29, 36, 42, 45
4	273	191	18	8	0, 4, 10, 18, 22, 24, 34, 36, 40, 48, 52, 56, 66, 67, 71, 76, 77, 82	0, 18, 24, 46, 50, 67, 103, 112, 115, 126, 128, 159, 166, 167, 186, 196, 201
5	1057	813	34	16	0, 1, 3, 4, 5, 11, 14, 17, 18, 22, 23, 26, 27, 28, 32, 33, 35, 37, 39, 41, 43, 45, 47, 48, 51, 52, 55, 59, 62, 68, 70, 71, 72, 74, 75, 76, 79, 81, 83, 88, 95, 98, 101, 103, 105, 106, 108, 111, 114, 115, 116, 120, 121, 122, 123, 124, 126, 129, 131, 132, 135, 137, 138, 141, 142, 146, 147, 149, 150, 151, 153, 154, 155, 158, 160, 161, 164, 165, 166, 167, 169, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 186, 188, 189, 191, 193, 194, 195, 198, 199, 200, 201, 202, 203, 208, 209, 210, 211, 212, 214, 216, 222, 224, 226, 228, 232, 234, 236, 242, 244	0, 1, 3, 7, 15, 31, 54, 63, 109, 127, 138, 219, 255, 277, 298, 338, 348, 439, 452, 511, 528, 555, 597, 677, 697, 702, 792, 897, 905, 924, 990, 1023

*Each generator polynomial is represented by the exponents of its nonzero terms. For example, {0, 2, 3, 4} represents $g(X) = 1 + X^2 + X^3 + X^4$.

polynomials and their corresponding perfect simple difference sets is given in Table 8.2.

Other one-step majority-logic decodable cyclic codes will be presented in Section 8.5.

8.4 MULTIPLE-STEP MAJORITY-LOGIC DECODING

The one-step majority-logic decoding for a cyclic code is based on the condition that a set of J parity-check sums orthogonal on a single error digit can be formed. This decoding method is effective for codes that are completely orthogonalizable or for codes with large J compared with their minimum distance d_{\min} . Unfortunately, only several small classes of cyclic codes are known to be in this category; however, the concept of parity-check sums orthogonal on a single error digit can be generalized in such a way that many cyclic codes can be decoded by employing several *levels* of majority-logic gates.

Let $E = \{e_{i_1}, e_{i_2}, \dots, e_{i_M}\}$ be a set of M error digits, where $0 \leq i_1 < i_2 < \dots < i_M < n$. The integer M is called the *size* of E .

DEFINITION 8.2 A set of J parity-check sums A_1, A_2, \dots, A_J is said to be orthogonal on the set E if and only if (1) every error digit e_{i_l} in E is checked by every check-sum A_j for $1 \leq j \leq J$, and (2) no other error digit is checked by more than one check-sum.

For example, the following four parity-check sums are orthogonal on the set $E = \{e_6, e_8\}$:

$$\begin{array}{llll} A_1 = e_0 & +e_2 & +e_6 & +e_8, \\ A_2 = & e_3 + e_4 & +e_6 & +e_8, \\ A_3 = & e_1 & +e_6 + e_7 & +e_8, \\ A_4 = & & e_5 + e_6 & +e_8. \end{array}$$

Following the same argument employed for one-step majority-logic decoding, we can correctly determine the sum of error digits in E , $e_{i_1} + e_{i_2} + \dots + e_{i_M}$ from the check-sums A_1, A_2, \dots, A_J orthogonal on E provided that there are $\lfloor J/2 \rfloor$ or fewer errors in the error pattern e . This sum of error digits in E may be regarded as an *additional* check-sum and so can be used for decoding.

Consider an (n, k) cyclic code C that is used for error control in a communication (or storage) system. Let $\mathbf{e} = (e_0, e_1, \dots, e_{n-1})$ denote the error vector that occurs during the transmission of a codeword \mathbf{v} in C . Let $E_1^1, E_2^1, \dots, E_i^1, \dots$ be some properly selected sets of error digits of \mathbf{e} . Let $S(E_i^1)$ denote the modulo-2 sum of the error digits in E_i^1 . Suppose that for each set E_i^1 it is possible to form at least J parity-check sums orthogonal on it. Then, the sum $S(E_i^1)$ can be estimated from these J orthogonal check-sums. The estimation can be done by a J -input majority-logic gate with the J orthogonal check-sums as inputs. The estimated value of $S(E_i^1)$ is the output of a majority-logic gate, which is 1 if and only if more than half of the inputs are 1; otherwise, it is 0. The estimation is correct provided that there are $\lfloor J/2 \rfloor$ or fewer errors in the error vector \mathbf{e} . The sums $S(E_1^1), S(E_2^1), \dots, S(E_i^1), \dots$ (possibly together with other check-sums) are then used to estimate the sums of error digits in the second selected sets, $E_1^2, E_2^2, \dots, E_i^2, \dots$, whose size is smaller than that of the first selected sets. Suppose that for each set E_i^2 it is possible to form J or more check-sums orthogonal on it. Then, the sum $S(E_i^2)$ can be determined correctly from the check-sums orthogonal on E_i^2 provided that there are no more than $\lfloor J/2 \rfloor$ errors in \mathbf{e} . Once the sums, $S(E_1^2), S(E_2^2), \dots, S(E_i^2), \dots$, are determined, they (maybe together with other check-sums) are used to estimate the sums of error digits in the third selected sets, $E_1^3, E_2^3, \dots, E_i^3, \dots$, whose size is smaller than that of the second selected sets. The process of estimating check-sums from known check-sums is called *orthogonalization* [2]. The orthogonalization process continues until a set of J or more check-sums orthogonal on only a single error digit, say e_{n-1} , is obtained. Then, the value of e_{n-1} can be estimated from these orthogonal check-sums. Because of the cyclic structure of the code, other error digits can be estimated in the same manner and by the same circuitry. A code is said to be *L-step orthogonalizable* (or *L-step majority-logic decodable*) if L steps of orthogonalization are required to

make a decoding decision on an error digit. The decoding process is called *L-step majority-logic decoding*. A code is said to be *completely L-step orthogonalizable* if J is 1 less than the minimum distance of the code (i.e., $J = d_{\min} - 1$). Because majority-logic gates are used to estimate selected sums of error digits at each step of orthogonalization, a total of L levels of majority-logic gates are required for decoding. The number of gates required at each level depends on the structure of the code.

The following two examples are used to illustrate the notions of multiple-step majority-logic decoding.

EXAMPLE 8.11

Consider the $(7, 4)$ cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$. This is a Hamming code. The parity-check matrix (in systematic form) is found as follows:

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \mathbf{h}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

We see that the vectors \mathbf{h}_0 and \mathbf{h}_2 are orthogonal on digit positions 5 and 6 (or X^5 and X^6). We also see that the vectors $\mathbf{h}_0 + \mathbf{h}_1$ and \mathbf{h}_2 are orthogonal on digit positions 4 and 6. Let $E_1^1 = \{e_5, e_6\}$ and $E_2^1 = \{e_4, e_6\}$ be two selected sets. Let $\mathbf{r} = (r_0, r_1, r_2, r_3, r_4, r_5, r_6)$ be the received vector. Then, the parity-check sums formed from \mathbf{h}_0 and \mathbf{h}_2 are

$$\begin{aligned} A_1 &= \mathbf{r} \cdot \mathbf{h}_0 = e_0 &+ e_3 &+ e_5 + e_6 \\ A_2 &= \mathbf{r} \cdot \mathbf{h}_2 = &e_2 &+ e_4 &+ e_5 + e_6 \end{aligned}$$

and the parity-check sums formed from $\mathbf{h}_0 + \mathbf{h}_1$ and \mathbf{h}_2 are

$$\begin{aligned} B_1 &= \mathbf{r} \cdot (\mathbf{h}_0 + \mathbf{h}_1) = e_0 + e_1 &+ e_4 &+ e_6 \\ B_2 &= \mathbf{r} \cdot \mathbf{h}_2 = &e_2 &+ e_4 &+ e_5 &+ e_6. \end{aligned}$$

The parity-check sums A_1 and A_2 are orthogonal on the set $E_1^1 = \{e_5, e_6\}$, and the parity-check sums B_1 and B_2 are orthogonal on the set $E_2^1 = \{e_4, e_6\}$. Therefore, the sum $S(E_1^1) = e_5 + e_6$ can be estimated from A_1 and A_2 , and the sum $S(E_2^1) = e_4 + e_6$ can be estimated from B_1 and B_2 . The sums $S(E_1^1)$ and $S(E_2^1)$ will be correctly estimated provided that there is no more than one error in the error vector \mathbf{e} . Now, let $E_1^2 = \{e_6\}$. We see that $S(E_1^1)$ and $S(E_1^2)$ are orthogonal on e_6 . Hence, e_6 can be estimated from $S(E_1^1)$ and $S(E_1^2)$. The value of e_6 will be estimated correctly provided that there is no more than one error in \mathbf{e} . Therefore, the $(7, 4)$ Hamming code can be decoded with two steps of orthogonalization, and it is two-step majority-logic decodable. Because its minimum distance is 3 and $J = 2$, it is two-step completely orthogonalizable. A type-II decoder for this code is shown in Figure 8.6.

Let $\mathbf{s} = (s_0, s_1, s_2) = \mathbf{r} \cdot \mathbf{H}^T$ be the syndrome of the received vector \mathbf{r} . Then, we can form the parity-check sums A_1 , A_2 , B_1 , and B_2 from the syndrome digits as follows:

$$\begin{aligned} A_1 &= s_0, &A_2 &= s_2, \\ B_1 &= s_0 + s_1, &B_2 &= s_2. \end{aligned}$$

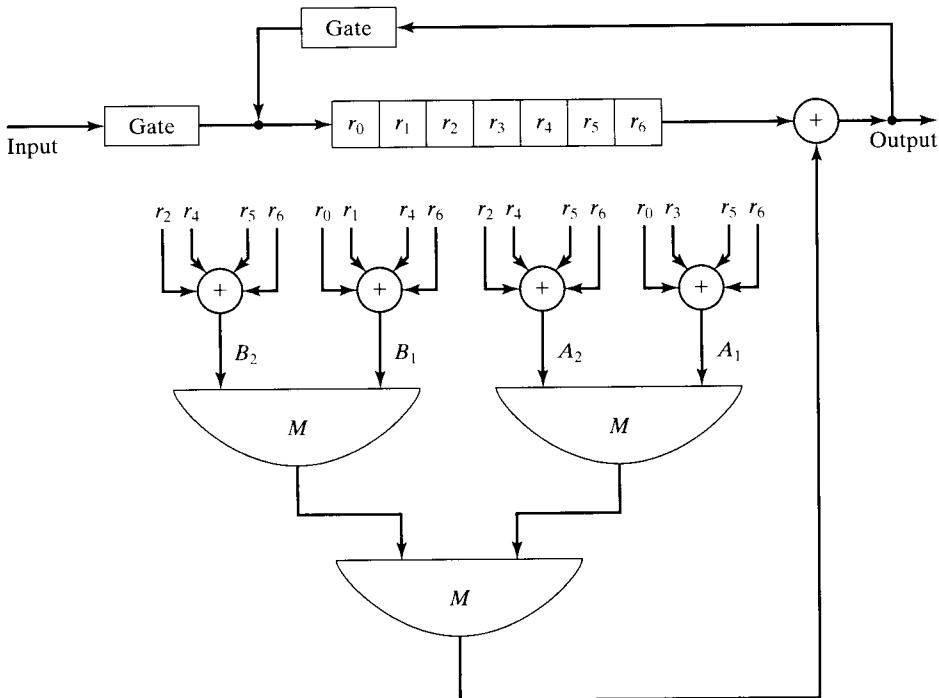


FIGURE 8.6: Type-II two-step majority-logic decoder for the (7, 4) Hamming code.

Based on these check-sums, we may construct a type-I majority-logic decoder for the (7, 4) Hamming code.

EXAMPLE 8.12

Consider the triple-error-correcting (15, 5) BCH code whose generator polynomial is

$$\mathbf{g}(X) = 1 + X + X^2 + X^4 + X^5 + X^8 + X^{10}.$$

The parity-check matrix (in systematic form) is

$$\mathbf{H} = \left[\begin{array}{c} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \\ \mathbf{h}_4 \\ \mathbf{h}_5 \\ \mathbf{h}_6 \\ \mathbf{h}_7 \\ \mathbf{h}_8 \\ \mathbf{h}_9 \end{array} \right] = \left[\begin{array}{ccccccccccccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{array} \right].$$

Let

$$\begin{aligned} E_1^1 &= \{e_{13}, e_{14}\}, & E_2^1 &= \{e_{12}, e_{14}\}, \\ E_3^1 &= \{e_{11}, e_{14}\}, & E_4^1 &= \{e_{10}, e_{14}\}, \\ E_5^1 &= \{e_5, e_{14}\}, & E_6^1 &= \{e_2, e_{14}\} \end{aligned}$$

be six selected sets of error digits. For each of the preceding sets it is possible to find six parity-check sums orthogonal on it. Let $\mathbf{r} = (r_0, r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11}, r_{12}, r_{13}, r_{14})$ be the received vector. By taking proper combinations of the rows \mathbf{H} , we find the following parity-check sums orthogonal on $E_1^1, E_2^1, E_3^1, E_4^1, E_5^1$, and E_6^1 :

1. Check-sums orthogonal on $E_1^1 = \{e_{13}, e_{14}\}$:

$$\begin{aligned} A_{11} &= \mathbf{r} \cdot \mathbf{h}_4 & &= e_4 + e_{10} + e_{13} + e_{14} \\ A_{12} &= \mathbf{r} \cdot \mathbf{h}_7 & &= e_7 + e_{12} + e_{13} + e_{14} \\ A_{13} &= \mathbf{r} \cdot \mathbf{h}_9 & &= e_9 + e_{11} + e_{13} + e_{14} \\ A_{14} &= \mathbf{r} \cdot (\mathbf{h}_0 + \mathbf{h}_8) & &= e_0 + e_8 + e_{13} + e_{14} \\ A_{15} &= \mathbf{r} \cdot (\mathbf{h}_1 + \mathbf{h}_5) & &= e_1 + e_5 + e_{13} + e_{14} \\ A_{16} &= \mathbf{r} \cdot (\mathbf{h}_3 + \mathbf{h}_6) & &= e_3 + e_6 + e_{13} + e_{14}. \end{aligned}$$

2. Check-sums orthogonal on $E_2^1 = \{e_{12}, e_{14}\}$:

$$\begin{aligned} A_{21} &= \mathbf{r} \cdot \mathbf{h}_0 & &= e_0 + e_{10} + e_{12} + e_{14} \\ A_{22} &= \mathbf{r} \cdot \mathbf{h}_3 & &= e_3 + e_{11} + e_{12} + e_{14} \\ A_{23} &= \mathbf{r} \cdot \mathbf{h}_7 & &= e_7 + e_{13} + e_{12} + e_{14} \\ A_{24} &= \mathbf{r} \cdot (\mathbf{h}_1 + \mathbf{h}_2) & &= e_1 + e_2 + e_{12} + e_{14} \\ A_{25} &= \mathbf{r} \cdot (\mathbf{h}_4 + \mathbf{h}_8) & &= e_4 + e_8 + e_{12} + e_{14} \\ A_{26} &= \mathbf{r} \cdot (\mathbf{h}_6 + \mathbf{h}_9) & &= e_6 + e_9 + e_{12} + e_{14}. \end{aligned}$$

3. Check-sums orthogonal on $E_3^1 = \{e_{11}, e_{14}\}$:

$$\begin{aligned} A_{31} &= \mathbf{r} \cdot \mathbf{h}_3 & &= e_3 + e_{12} + e_{11} + e_{14} \\ A_{32} &= \mathbf{r} \cdot \mathbf{h}_9 & &= e_9 + e_{13} + e_{11} + e_{14} \\ A_{33} &= \mathbf{r} \cdot (\mathbf{h}_0 + \mathbf{h}_5) & &= e_0 + e_5 + e_{11} + e_{14} \\ A_{34} &= \mathbf{r} \cdot (\mathbf{h}_1 + \mathbf{h}_8) & &= e_1 + e_8 + e_{11} + e_{14} \\ A_{35} &= \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_4) & &= e_2 + e_4 + e_{11} + e_{14} \\ A_{36} &= \mathbf{r} \cdot (\mathbf{h}_6 + \mathbf{h}_7) & &= e_6 + e_7 + e_{11} + e_{14}. \end{aligned}$$

4. Check-sums orthogonal on $E_4^1 = \{e_{10}, e_{14}\}$:

$$\begin{aligned} A_{41} &= \mathbf{r} \cdot \mathbf{h}_0 & &= e_0 + e_{12} + e_{10} + e_{14} \\ A_{42} &= \mathbf{r} \cdot \mathbf{h}_4 & &= e_4 + e_{13} + e_{10} + e_{14} \\ A_{43} &= \mathbf{r} \cdot (\mathbf{h}_1 + \mathbf{h}_6) & &= e_1 + e_6 + e_{10} + e_{14} \\ A_{44} &= \mathbf{r} \cdot (\mathbf{h}_3 + \mathbf{h}_5) & &= e_3 + e_5 + e_{10} + e_{14} \\ A_{45} &= \mathbf{r} \cdot (\mathbf{h}_7 + \mathbf{h}_8) & &= e_7 + e_8 + e_{10} + e_{14} \\ A_{46} &= \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_9) & &= e_2 + e_9 + e_{10} + e_{14}. \end{aligned}$$

5. Check-sums orthogonal on $E_5^1 = \{e_5, e_{14}\}$:

$$\begin{aligned} A_{51} &= \mathbf{r} \cdot (\mathbf{h}_0 + \mathbf{h}_5) &= e_0 + e_{11} + e_5 + e_{14} \\ A_{52} &= \mathbf{r} \cdot (\mathbf{h}_1 + \mathbf{h}_5) &= e_1 + e_{13} + e_5 + e_{14} \\ A_{53} &= \mathbf{r} \cdot (\mathbf{h}_3 + \mathbf{h}_5) &= e_3 + e_{10} + e_5 + e_{14} \\ A_{54} &= \mathbf{r} \cdot (\mathbf{h}_4 + \mathbf{h}_5 + \mathbf{h}_6) &= e_4 + e_6 + e_5 + e_{14} \\ A_{55} &= \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_5 + \mathbf{h}_7) &= e_2 + e_7 + e_5 + e_{14} \\ A_{56} &= \mathbf{r} \cdot (\mathbf{h}_5 + \mathbf{h}_8 + \mathbf{h}_9) &= e_8 + e_9 + e_5 + e_{14}. \end{aligned}$$

6. Check-sums orthogonal on $E_6^1 = \{e_2, e_{14}\}$:

$$\begin{aligned} A_{61} &= \mathbf{r} \cdot (\mathbf{h}_1 + \mathbf{h}_2) &= e_1 + e_{12} + e_2 + e_{14} \\ A_{62} &= \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_4) &= e_4 + e_{11} + e_2 + e_{14} \\ A_{63} &= \mathbf{r} \cdot (\mathbf{h}_0 + \mathbf{h}_2 + \mathbf{h}_6) &= e_0 + e_6 + e_2 + e_{14} \\ A_{64} &= \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_3 + \mathbf{h}_8) &= e_3 + e_8 + e_2 + e_{14} \\ A_{65} &= \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_5 + \mathbf{h}_7) &= e_5 + e_7 + e_2 + e_{14} \\ A_{66} &= \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_9) &= e_9 + e_{10} + e_2 + e_{14}. \end{aligned}$$

From the foregoing orthogonal check-sums, the sums $S(E_1^1) = e_{13} + e_{14}$, $S(E_2^1) = e_{12} + e_{14}$, $S(E_3^1) = e_{11} + e_{14}$, $S(E_4^1) = e_{10} + e_{14}$, $S(E_5^1) = e_5 + e_{14}$, and $S(E_6^1) = e_2 + e_{14}$ can be correctly estimated provided that there are no more than three errors in the error vector \mathbf{e} . Let $E_1^2 = \{e_{14}\}$. We see that the error sums $S(E_1^1)$, $S(E_2^1)$, $S(E_3^1)$, $S(E_4^1)$, $S(E_5^1)$, and $S(E_6^1)$ are orthogonal on e_{14} . Hence, e_{14} can be estimated from these sums. Therefore, the (15, 5) BCH code is two-step orthogonalizable. Because $J = 6$, it is capable of correcting three or fewer errors with two-step majority-logic decoding. It is known that the code has a minimum distance of exactly 7. Hence, it is two-step completely orthogonalizable.

The type-II decoder for the (15, 5) BCH code is shown in Figure 8.7, where seven six-input majority-logic gates (connected in a tree form) are used. Construction of a type-I majority-logic decoder for the (15, 5) BCH code is left as an exercise (see Problem 8.12).

A general type-II L -step majority-logic decoder is shown in Figure 8.8. The error correction procedure is as follows:

- Step 1.** The received vector $\mathbf{r}(X)$ is read into the buffer register.
- Step 2.** Parity-check sums (no more than $(J)^L$ of them) orthogonal on certain properly selected sets of error digits are formed by summing appropriate sets of received digits. These check-sums are then fed into the first-level majority-logic gates (there are at most $(J)^{L-1}$ of them). The outputs of the first-level majority-logic gates are used to form inputs to the second-level majority-logic gates (there are at most $(J)^{L-2}$ of them). The outputs of the second-level majority-logic gates are then used to form inputs to third-level majority-logic gates (there are at most $(J)^{L-3}$ of them). This process continues until the last level is reached; there is only one gate at the last level. The J inputs to this

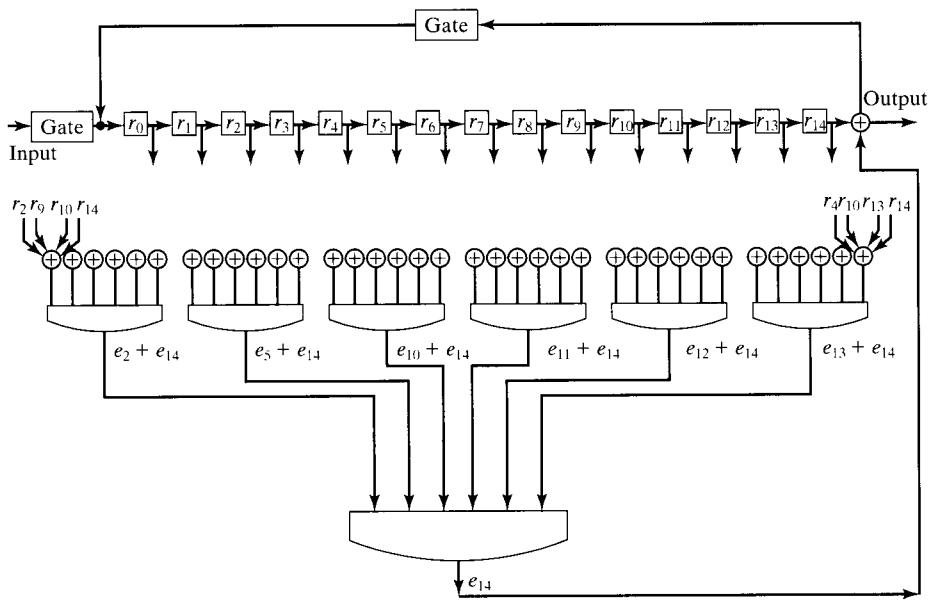
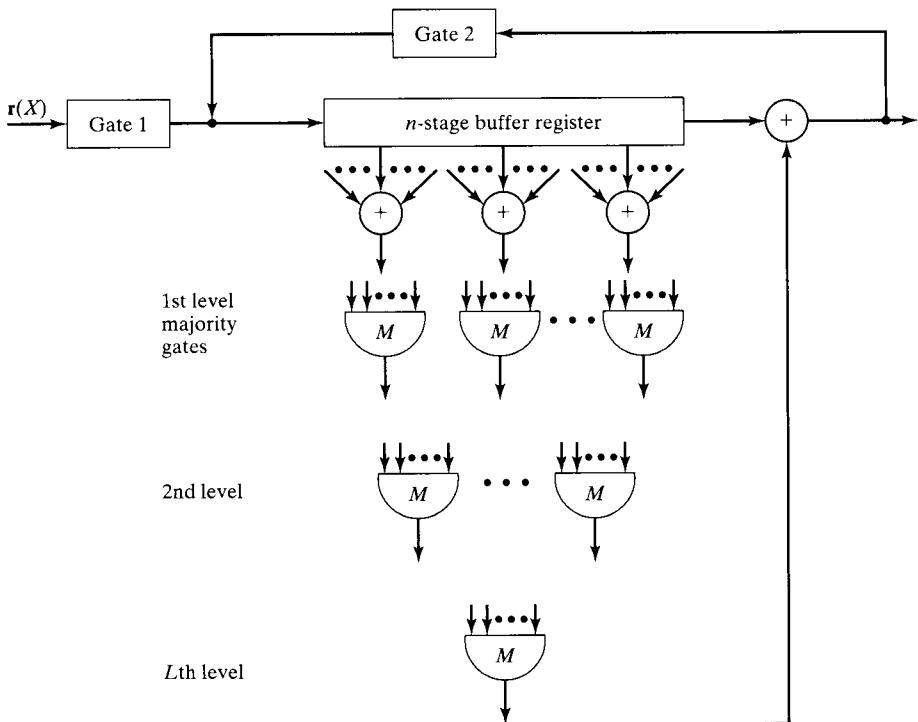


FIGURE 8.7: Type II two-step majority-logic decoder for the (15, 5) BCH code.

FIGURE 8.8: General type-II L -step majority-logic decoder.

gate are check-sums orthogonal on the highest-order error digit e_{n-1} . The output of this gate is used to correct the received digit r_{n-1} .

- Step 3.** The received digit r_{n-1} is read out of the buffer and is corrected by the last-level majority-logic gate.
- Step 4.** At the end of step 3, the buffer register has been shifted one place to the right. Now, the second-highest-order received digit r_{n-2} is in the rightmost stage of the buffer register, and it will be corrected in exactly the same manner as was the highest-order received digit r_{n-1} . The decoder repeats steps 2 and 3.
- Step 5.** The received vector is decoded digit by digit in the manner described until a total of n shifts.

A general type-I decoder for an L -step majority-logic decoder code is shown in Figure 8.9. Its decoding operation is identical to that of the type-I decoder for

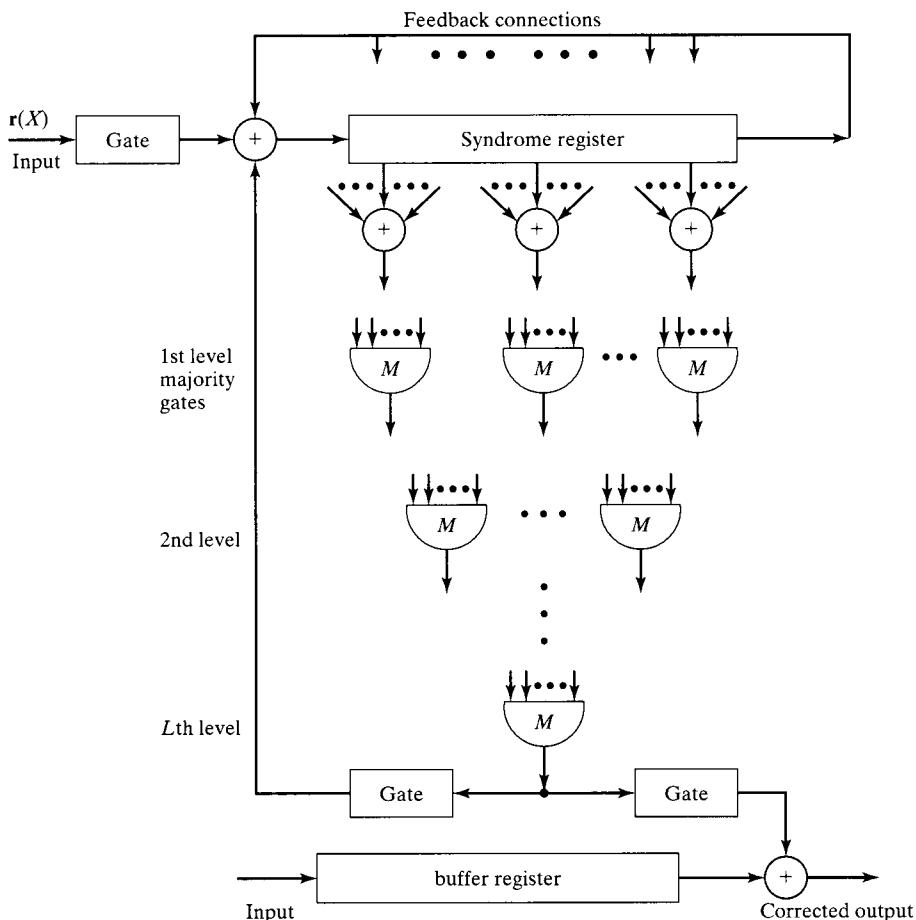


FIGURE 8.9: General type-I L -step majority-logic decoder.

a one-step majority-logic decodable code except that L levels of orthogonalization are required.

An L -step majority-logic decoder requires L levels of majority-logic gates. At the i th level, no more than $(J)^{L-i}$ gates are required. Thus, the total number of majority-logic gates needed is upper bounded by $1 + J + J^2 + \dots + J^{L-1}$. In fact, Massey [2] has proved that for an (n, k) L -step majority-logic decodable code no more than k majority-logic gates are ever required. Unfortunately, for a given L -step majority-logic decodable cyclic code, there is no known systematic method for minimizing the number of majority-logic gates except the trial-and-error method. For almost all the known classes of L -step majority-logic decodable codes, the rules for forming orthogonal parity-check sums require a total of $1 + J + J^2 + \dots + J^{L-1}$ majority-logic gates. Thus, the complexity is an exponential function of L . For large L , the decoder is likely to be impractical. Fortunately, there are many cyclic codes with useful parameters that can be decoded with a reasonably small L .

Several large classes of cyclic codes have been found to be L -step majority-logic decodable. The construction and the rules for orthogonalization of these codes are based on the properties of finite geometries, which are the subject of the next four sections.

8.5 EUCLIDEAN GEOMETRY

Consider all the m -tuples $(a_0, a_1, \dots, a_{m-1})$, with components a_i 's from the Galois field $GF(2^s)$. There are $(2^s)^m = 2^{ms}$ such m -tuples. These 2^{ms} m -tuples form a vector space over $GF(2^s)$. The vector addition and scalar multiplication are defined in the usual way:

$$(a_0, a_1, \dots, a_{m-1}) + (b_0, b_1, \dots, b_{m-1}) = (a_0 + b_0, a_1 + b_1, \dots, a_{m-1} + b_{m-1}),$$

$$\beta \cdot (a_0, a_1, \dots, a_{m-1}) = (\beta \cdot a_0, \beta \cdot a_1, \dots, \beta \cdot a_{m-1}).$$

where additions $a_i + b_i$ and multiplication $\beta \cdot a_i$ are carried out in $GF(2^s)$. In combinatorial mathematics, the 2^{ms} m -tuples over $GF(2^s)$ are also known to form an m -dimensional *Euclidean geometry* over $GF(2^s)$, denoted by $EG(m, 2^s)$ [14–16]. Each m -tuple $\mathbf{a} = (a_0, a_1, \dots, a_{m-1})$ is called a *point* in $EG(m, 2^s)$. The all-zero m -tuple, $\mathbf{0} = (0, 0, \dots, 0)$, is called the *origin* of the geometry $EG(m, 2^s)$.

Let \mathbf{a} be a nonorigin point in $EG(m, 2^s)$ (i.e., $\mathbf{a} \neq \mathbf{0}$). Then, the 2^s points $\{\beta\mathbf{a} : \beta \in GF(2^s)\}$ constitute a *line* (or 1-flat) in $EG(m, 2^s)$. For convenience, we use the notation $\{\beta\mathbf{a}\}$ to represent this line. Because this line contains the origin (with $\beta = 0$), we say that $\{\beta\mathbf{a}\}$ passes through the origin. Let \mathbf{a}_0 and \mathbf{a} be two linearly independent points in $EG(m, 2^s)$ (i.e., $\beta_0\mathbf{a}_0 + \beta\mathbf{a} \neq \mathbf{0}$ unless $\beta_0 = \beta = 0$). Then, the collection of the following 2^s points,

$$\{\mathbf{a}_0 + \beta\mathbf{a}\},$$

with $\beta \in GF(2^s)$, constitutes a line in $EG(m, 2^s)$ that passes through the point \mathbf{a}_0 . Line $\{\beta\mathbf{a}\}$ and the line $\{\mathbf{a}_0 + \beta\mathbf{a}\}$ do not have any point in common. Suppose that they have a common point. Then, for some β' and β'' in $GF(2^s)$,

$$\beta'\mathbf{a} = \mathbf{a}_0 + \beta''\mathbf{a}.$$

As a result, $\mathbf{a}_0 + (\beta'' - \beta')\mathbf{a} = 0$. This implies that \mathbf{a}_0 and \mathbf{a} are linearly dependent, which is a contradiction to our assumption that \mathbf{a}_0 and \mathbf{a} are two linearly independent points in $\text{EG}(m, 2^s)$. Therefore, $\{\beta\mathbf{a}\}$ and $\{\mathbf{a}_0 + \beta\mathbf{a}\}$ do not have any common points. We say that $\{\beta\mathbf{a}\}$ and $\{\mathbf{a}_0 + \beta\mathbf{a}\}$ are *parallel* lines. Note that $\{\beta\mathbf{a}\}$ is simply a one-dimensional subspace of the vector space of all the 2^{ms} m -tuples over $GF(2^s)$, and $\{\mathbf{a}_0 + \beta\mathbf{a}\}$ is simply a coset of $\{\beta\mathbf{a}\}$. Let \mathbf{b}_0 be a point not on line $\{\beta\mathbf{a}\}$ or on line $\{\mathbf{a}_0 + \beta\mathbf{a}\}$. The line $\{\mathbf{b}_0 + \beta\mathbf{a}\}$ passes through the point \mathbf{b}_0 and is parallel to both $\{\beta\mathbf{a}\}$ and $\{\mathbf{a}_0 + \beta\mathbf{a}\}$. In $\text{EG}(m, 2^s)$, for every line passing through the origin, there are $2^{(m-1)s} - 1$ lines parallel to it. A line $\{\beta\mathbf{a}\}$ and the $2^{(m-1)s} - 1$ lines parallel to it are said to form a *parallel bundle*. The $2^{(m-1)s}$ lines in a parallel bundle are parallel to each other. Basically, the $2^{(m-1)s}$ lines in a parallel bundle simply correspond to a one-dimensional subspace of the vector space of all the m -tuples over $GF(2)$ and its $2^{(m-1)s} - 1$ cosets.

Let \mathbf{a}_1 and \mathbf{a}_2 be two linearly independent points in $\text{EG}(m, 2^s)$. The lines $\{\mathbf{a}_0 + \beta\mathbf{a}_1\}$ and $\{\mathbf{a}_0 + \beta\mathbf{a}_2\}$ have only one point, \mathbf{a}_0 , in common. Suppose that they have another point besides \mathbf{a}_0 in common. Then, for some $\beta' \neq 0$ and $\beta'' \neq 0$, we have

$$\mathbf{a}_0 + \beta'\mathbf{a}_1 = \mathbf{a}_0 + \beta''\mathbf{a}_2.$$

This equality implies that $\beta'\mathbf{a}_1 - \beta''\mathbf{a}_2 = 0$ and that \mathbf{a}_1 and \mathbf{a}_2 are linearly dependent. This is a contradiction to the hypothesis that \mathbf{a}_1 and \mathbf{a}_2 are linearly independent points in $\text{EG}(m, 2^s)$. Therefore, $\{\mathbf{a}_0 + \beta\mathbf{a}_1\}$ and $\{\mathbf{a}_0 + \beta\mathbf{a}_2\}$ have only one point in common, and they both pass through the point \mathbf{a}_0 . We say that $\{\mathbf{a}_0 + \beta\mathbf{a}_1\}$ and $\{\mathbf{a}_0 + \beta\mathbf{a}_2\}$ intersect at the point \mathbf{a}_0 . Given a point \mathbf{a}_0 in $\text{EG}(m, 2^s)$, there are

$$\frac{2^{ms} - 1}{2^s - 1} \quad (8.26)$$

lines in $\text{EG}(m, 2^s)$ that intersect at \mathbf{a}_0 (including the line $\{\beta\mathbf{a}_0\}$ that passes through the origin). This is an important property that will be used to form orthogonal parity-check sums for the codes presented in the next section. Another important structural property of lines is that any two points are connected by a line. Let \mathbf{a}_1 and \mathbf{a}_2 be two points in $\text{EG}(m, 2^s)$. Suppose that \mathbf{a}_1 and \mathbf{a}_2 are linearly dependent. Then, $\mathbf{a}_2 = \beta_i\mathbf{a}_1$ for some element β_i in $GF(2^s)$. In this case \mathbf{a}_1 and \mathbf{a}_2 are connected by the line $\{\beta\mathbf{a}_1\}$. Suppose that \mathbf{a}_1 and \mathbf{a}_2 are linearly independent. Let $\mathbf{a}_3 = \mathbf{a}_1 + \mathbf{a}_2$. Then, $\mathbf{a}_2 = \mathbf{a}_1 + \mathbf{a}_3$, and \mathbf{a}_1 and \mathbf{a}_2 are connected by the line $\{\mathbf{a}_1 + \beta\mathbf{a}_3\}$. The total number of lines in $\text{EG}(m, 2^s)$ is

$$\frac{2^{(m-1)s}(2^{ms} - 1)}{2^s - 1}.$$

EXAMPLE 8.13

Let $m = 3$ and $s = 1$. Consider the Euclidean geometry $\text{EG}(3, 2)$ over $GF(2)$. There are eight points and 28 lines. Each point \mathbf{a}_i is a 3-tuple over $GF(2)$. Each line consists of two points $\{\mathbf{a}_i, \mathbf{a}_j\}$. The points and the lines are given in Table 8.3. Lines $\{\mathbf{a}_0, \mathbf{a}_1\}$, $\{\mathbf{a}_2, \mathbf{a}_3\}$, $\{\mathbf{a}_4, \mathbf{a}_5\}$, and $\{\mathbf{a}_6, \mathbf{a}_7\}$ are parallel and they form a parallel bundle. The lines that intersect at the point \mathbf{a}_2 are $\{\mathbf{a}_0, \mathbf{a}_2\}$, $\{\mathbf{a}_1, \mathbf{a}_2\}$, $\{\mathbf{a}_2, \mathbf{a}_3\}$, $\{\mathbf{a}_2, \mathbf{a}_4\}$, $\{\mathbf{a}_2, \mathbf{a}_5\}$, $\{\mathbf{a}_2, \mathbf{a}_6\}$, and $\{\mathbf{a}_2, \mathbf{a}_7\}$.

TABLE 8.3: Points and lines in EG(3, 2).

(a) Points in EG(3, 2)			
$\mathbf{a}_0 = (000)$	$\mathbf{a}_1 = (001)$	$\mathbf{a}_2 = (010)$	$\mathbf{a}_3 = (011)$
$\mathbf{a}_4 = (100)$	$\mathbf{a}_5 = (101)$	$\mathbf{a}_6 = (110)$	$\mathbf{a}_7 = (111)$
(b) Lines in EG(3, 2)			
$\{\mathbf{a}_0, \mathbf{a}_1\}$	$\{\mathbf{a}_1, \mathbf{a}_2\}$	$\{\mathbf{a}_2, \mathbf{a}_4\}$	$\{\mathbf{a}_3, \mathbf{a}_7\}$
$\{\mathbf{a}_0, \mathbf{a}_2\}$	$\{\mathbf{a}_1, \mathbf{a}_3\}$	$\{\mathbf{a}_2, \mathbf{a}_5\}$	$\{\mathbf{a}_4, \mathbf{a}_5\}$
$\{\mathbf{a}_0, \mathbf{a}_3\}$	$\{\mathbf{a}_1, \mathbf{a}_4\}$	$\{\mathbf{a}_2, \mathbf{a}_6\}$	$\{\mathbf{a}_4, \mathbf{a}_6\}$
$\{\mathbf{a}_0, \mathbf{a}_4\}$	$\{\mathbf{a}_1, \mathbf{a}_5\}$	$\{\mathbf{a}_2, \mathbf{a}_7\}$	$\{\mathbf{a}_4, \mathbf{a}_7\}$
$\{\mathbf{a}_0, \mathbf{a}_5\}$	$\{\mathbf{a}_1, \mathbf{a}_6\}$	$\{\mathbf{a}_3, \mathbf{a}_4\}$	$\{\mathbf{a}_5, \mathbf{a}_6\}$
$\{\mathbf{a}_0, \mathbf{a}_6\}$	$\{\mathbf{a}_1, \mathbf{a}_7\}$	$\{\mathbf{a}_3, \mathbf{a}_5\}$	$\{\mathbf{a}_5, \mathbf{a}_7\}$
$\{\mathbf{a}_0, \mathbf{a}_7\}$	$\{\mathbf{a}_2, \mathbf{a}_3\}$	$\{\mathbf{a}_3, \mathbf{a}_6\}$	$\{\mathbf{a}_6, \mathbf{a}_7\}$

Now, we extend the concept of lines to planes in $EG(m, 2^s)$. Let $\mathbf{a}_0, \mathbf{a}_1, \dots, \mathbf{a}_\mu$ be $\mu + 1$ linearly independent points in $EG(m, 2^s)$, where $\mu < m$. The 2^{ms} points of the form

$$\mathbf{a}_0 + \beta_1 \mathbf{a}_1 + \beta_2 \mathbf{a}_2 + \cdots + \beta_\mu \mathbf{a}_\mu,$$

with $\beta_i \in GF(2^s)$ for $1 \leq i \leq \mu$, constitute a μ -flat (or a μ -dimensional hyperplane) in $EG(m, 2^s)$ that passes through the point \mathbf{a}_0 . We denote this μ -flat by $\{\mathbf{a}_0 + \beta_1 \mathbf{a}_1 + \cdots + \beta_\mu \mathbf{a}_\mu\}$. The μ -flat that consists of the 2^{ms} points

$$\beta_1 \mathbf{a}_1 + \beta_2 \mathbf{a}_2 + \cdots + \beta_\mu \mathbf{a}_\mu$$

passes through the origin. We can readily prove that the μ -flats $\{\beta_1 \mathbf{a}_1 + \beta_2 \mathbf{a}_2 + \beta_3 \mathbf{a}_3 + \cdots + \beta_\mu \mathbf{a}_\mu\}$ and $\{\mathbf{a}_0 + \beta_1 \mathbf{a}_1 + \beta_2 \mathbf{a}_2 + \cdots + \beta_\mu \mathbf{a}_\mu\}$ do not have any point in common. We say that these two μ -flats are parallel. For any μ -flat passing through the origin, there are $2^{(m-\mu)s} - 1$ μ -flats in $EG(m, 2^s)$ parallel to it. These $2^{(m-\mu)s}$ parallel μ -flats form a parallel bundle. Note that a μ -flat in $EG(m, 2^s)$ is either a μ -dimensional subspace of the vector space of all the 2^{ms} m -tuples over $GF(2^s)$ or a coset of a μ -dimensional subspace. The $2^{(m-\mu)s}$ parallel μ -flats in a parallel bundle simply correspond to a μ -dimensional subspace of the vector space of all the m -tuples over $GF(2)$ and its $2^{(m-1)s} - 1$ cosets.

If $\mathbf{a}_{\mu+1}$ is not a point in the μ -flat $\{\mathbf{a}_0 + \beta_1 \mathbf{a}_1 + \cdots + \beta_\mu \mathbf{a}_\mu\}$, then the $(\mu + 1)$ -flat $\{\mathbf{a}_0 + \beta_1 \mathbf{a}_1 + \cdots + \beta_\mu \mathbf{a}_\mu + \beta_{\mu+1} \mathbf{a}_{\mu+1}\}$ contains the μ -flat $\{\mathbf{a}_0 + \beta_1 \mathbf{a}_1 + \cdots + \beta_\mu \mathbf{a}_\mu\}$. Let $\mathbf{b}_{\mu+1}$ be a point not in $\{\mathbf{a}_0 + \beta_1 \mathbf{a}_1 + \cdots + \beta_{\mu+1} \mathbf{a}_{\mu+1}\}$. Then, the two $(\mu + 1)$ -flats $\{\mathbf{a}_0 + \beta_1 \mathbf{a}_1 + \cdots + \beta_\mu \mathbf{a}_\mu + \beta_{\mu+1} \mathbf{a}_{\mu+1}\}$ and $\{\mathbf{a}_0 + \beta_1 \mathbf{a}_1 + \cdots + \beta_\mu \mathbf{a}_\mu + \beta_{\mu+1} \mathbf{b}_{\mu+1}\}$ intersect on the μ -flat $\{\mathbf{a}_0 + \beta_1 \mathbf{a}_1 + \cdots + \beta_\mu \mathbf{a}_\mu\}$ (i.e., they have the points in $\{\mathbf{a}_0 + \beta_1 \mathbf{a}_1 + \cdots + \beta_\mu \mathbf{a}_\mu\}$ as all their common points). Given a μ -flat F in $EG(m, 2^s)$, the number of $(\mu + 1)$ -flats in $EG(m, 2^s)$ that intersect on F is

$$\frac{2^{(m-\mu)s} - 1}{2^s - 1}. \quad (8.27)$$

TABLE 8.4: 2-Flats in EG(3, 2).

{a ₀ , a ₁ , a ₂ , a ₃ }	{a ₄ , a ₅ , a ₆ , a ₇ }	{a ₀ , a ₁ , a ₄ , a ₅ }	{a ₂ , a ₃ , a ₆ , a ₇ }
{a ₀ , a ₂ , a ₄ , a ₆ }	{a ₁ , a ₃ , a ₅ , a ₇ }	{a ₀ , a ₁ , a ₆ , a ₇ }	{a ₂ , a ₃ , a ₄ , a ₅ }
{a ₀ , a ₂ , a ₅ , a ₇ }	{a ₁ , a ₃ , a ₄ , a ₆ }	{a ₀ , a ₄ , a ₃ , a ₇ }	{a ₁ , a ₂ , a ₅ , a ₆ }
{a ₀ , a ₃ , a ₅ , a ₆ }	{a ₁ , a ₂ , a ₄ , a ₇ }		

Any point outside the μ -flat F is contained in one and only one of the $(\mu + 1)$ -flats that intersect on F . The number of μ -flats in $EG(m, 2^s)$ is

$$2^{(m-\mu)s} \prod_{i=1}^{\mu} \frac{2^{(m-i+1)s} - 1}{2^{(\mu-i+1)s} - 1}.$$

EXAMPLE 8.14

Consider the geometry $EG(3, 2)$ over $GF(2)$ given in Example 8.13. There are fourteen 2-flats, which are given in Table 8.4. The 2-flats that intersect on the line $\{a_1, a_3\}$ are $\{a_0, a_1, a_2, a_3\}$, $\{a_1, a_3, a_5, a_7\}$, and $\{a_1, a_3, a_4, a_6\}$. The 2-flats $\{a_0, a_1, a_2, a_3\}$ and $\{a_4, a_5, a_6, a_7\}$ are parallel.

Next, we show that the elements in the Galois field $GF(2^{ms})$ actually form an m -dimensional Euclidean geometry $EG(m, 2^s)$. Let α be a primitive element of $GF(2^{ms})$. Then, the 2^{ms} elements in $GF(2^{ms})$ can be expressed as powers of α as follows: $\alpha^\infty = 0, \alpha^0 = 1, \alpha^1, \alpha^2, \dots, \alpha^{2^{ms}-2}$. It is known that $GF(2^{ms})$ contains $GF(2^s)$ as a subfield. Every element α^i in $GF(2^{ms})$ can be expressed as

$$\alpha^i = a_{i0} + a_{i1}\alpha + a_{i2}\alpha^2 + \dots + a_{i,m-1}\alpha^{m-1},$$

where $a_{ij} \in GF(2^s)$ for $0 \leq j < m$. There is a *one-to-one correspondence* between the element α^i and the m -tuple $(a_{i0}, a_{i1}, \dots, a_{i,m-1})$ over $GF(2^s)$. Therefore, the 2^{ms} elements in $GF(2^{ms})$ may be regarded as the 2^{ms} points in $EG(m, 2^s)$, and $GF(2^{ms})$ as the geometry $EG(m, 2^s)$. In this case, a μ -flat passing through the point α^{l_0} consists of the following $2^{\mu s}$ points:

$$\alpha^{l_0} + \beta_1\alpha^{l_1} + \dots + \beta_\mu\alpha^{l_\mu},$$

where $\alpha^{l_0}, \alpha^{l_1}, \dots, \alpha^{l_\mu}$ are $\mu + 1$ linearly independent elements in $GF(2^{ms})$, and $\beta_i \in GF(2^s)$.

EXAMPLE 8.15

Consider the Galois field $GF(2^4)$ given by Table 2.8. Let $m = 2$. Let α be a primitive element whose minimal polynomial is $\phi(X) = 1 + X + X^4$. Let $\beta = \alpha^5$. We see that $\beta^0 = 1, \beta^1 = \alpha^5, \beta^2 = \alpha^{10}$, and $\beta^3 = \alpha^{15} = 1$. Therefore, the order of β is 3. We can readily check that the elements

$$0, 1, \beta, \beta^2$$

TABLE 8.5: Elements in $GF(2^4)^*$.

2-tuples over $GF(2^2)$	
$0 = 0$	(0, 0)
$1 = 1$	(1, 0)
$\alpha = \alpha$	(0, 1)
$\alpha^2 = \beta + \alpha$	(β , 1)
$\alpha^3 = \beta + \beta^2\alpha$	(β , β^2)
$\alpha^4 = 1 + \alpha$	(1, 1)
$\alpha^5 = \beta$	(β , 0)
$\alpha^6 = \beta\alpha$	(0, β)
$\alpha^7 = \beta^2 + \beta\alpha$	(β^2 , β)
$\alpha^8 = \beta^2 + \alpha$	(β^2 , 1)
$\alpha^9 = \beta + \beta\alpha$	(β , β)
$\alpha^{10} = \beta^2$	(β^2 , 0)
$\alpha^{11} = \beta^2\alpha$	(0, β^2)
$\alpha^{12} = 1 + \beta^2\alpha$	(1, β^2)
$\alpha^{13} = 1 + \beta\alpha$	(1, β)
$\alpha^{14} = \beta^2 + \beta^2\alpha$	(β^2 , β^2)

*Elements in $GF(2^4)$ are expressed in the form $a_{i0} + a_{i1}\alpha$, where α is a primitive element in $GF(2^4)$, and a_{ij} is an element in $GF(2^2) = \{0, 1, \beta, \beta^2\}$ with $\beta = \alpha^5$.

form a field of four elements, $GF(2^2)$. Therefore, $GF(2^2)$ is a subfield of $GF(2^4)$. Table 8.5 shows that every element α^i in $GF(2^4)$ is expressed in the form

$$\alpha^i = a_{i0} + a_{i1}\alpha,$$

with a_{i0} and a_{i1} in $GF(2^2) = \{0, 1, \beta, \beta^2\}$. We may regard $GF(2^4)$ as the Euclidean geometry $EG(2, 2^2)$ over $GF(2^2)$. Then, the points

$$\begin{array}{ll} \alpha^{14} + 0 \cdot \alpha = \alpha^{14}, & \alpha^{14} + 1 \cdot \alpha = \alpha^7, \\ \alpha^{14} + \beta \cdot \alpha = \alpha^8, & \alpha^{14} + \beta^2 \cdot \alpha = \alpha^{10}, \end{array}$$

form a line passing through the point α^{14} . The other four lines in $EG(2, 2^2)$ passing through α^{14} are

$$\begin{array}{ll} \{\alpha^{14}, \alpha^{13}, \alpha, \alpha^5\}, & \{\alpha^{14}, \alpha^0, \alpha^6, \alpha^2\}, \\ \{\alpha^{14}, \alpha^9, \alpha^4, 0\}, & \{\alpha^{14}, \alpha^{12}, \alpha^{11}, \alpha^3\}. \end{array}$$

The field $GF(2^{ms})$ may be regarded either as an extension field of $GF(2^s)$ or as an extension field of $GF(2^m)$. Therefore, $GF(2^{ms})$ may be regarded either as the m -dimensional Euclidean geometry $EG(m, 2^s)$ over $GF(2^s)$ or as the s -dimensional Euclidean geometry $EG(s, 2^m)$ over $GF(2^m)$.

8.6 EUCLIDEAN GEOMETRY CODES

Let

$$\mathbf{v} = (v_0, v_1, \dots, v_{2^m-2}),$$

be a $(2^m - 1)$ -tuple over the binary field $GF(2)$. Let α be a primitive element of the Galois field $GF(2^m)$. We may number the components of \mathbf{v} with the nonzero elements of $GF(2^m)$ as follows: the component v_i is numbered α^i for $0 \leq i \leq 2^m - 2$. Hence, α^i is the location number of v_i . Now, we regard $GF(2^m)$ as the m -dimensional Euclidean geometry over $GF(2^s)$, $EG(m, 2^s)$. Let F be a μ -flat in $EG(m, 2^s)$ that does not pass through the origin, $\alpha^\infty = 0$. Based on this μ -flat F , we may form a vector over $GF(2)$ as follows:

$$\mathbf{v}_F = (v_0, v_1, \dots, v_{2^m-2}),$$

whose i th component v_i is 1 if its location number α^i is a point in F ; otherwise, v_i is 0. In other words, the location numbers for the nonzero components of \mathbf{v}_F form the points of the μ -flat F . The vector \mathbf{v}_F is called the *incidence vector* of the μ -flat F . The incidence vector \mathbf{v}_F for the μ -flat F simply displays the points contained in F . A very interesting structural property of the incidence vectors of the μ -flats in $EG(m, 2^s)$ not passing through the origin is their cyclic structure: a cyclic shift of the incidence vector of a μ -flat not passing through the origin is the incidence vector of another μ -flat not passing through the origin (see Problem 8.33).

EXAMPLE 8.16

Let $m = 2$ and $s = 2$. Consider the field $GF(2^4)$, which is regarded as the Euclidean geometry over $GF(2^2)$, $EG(2, 2^2)$. From Example 8.15, the four 1-flats (or lines) passing through the point α^{14} but not the origin are

$$L_1 = \{\alpha^{14}, \alpha^7, \alpha^8, \alpha^{10}\}, \quad L_2 = \{\alpha^{14}, \alpha^{13}, \alpha, \alpha^5\},$$

$$L_3 = \{\alpha^{14}, \alpha^0, \alpha^6, \alpha^2\}, \quad L_4 = \{\alpha^{14}, \alpha^{12}, \alpha^{11}, \alpha^3\}.$$

The incidence vectors for these four 1-flats are

Location Numbers

α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
$\mathbf{v}_{L_1} = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)$														
$\mathbf{v}_{L_2} = (0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1)$														
$\mathbf{v}_{L_3} = (1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)$														
$\mathbf{v}_{L_4} = (0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 0)$														

Suppose we cyclically shift the incidence vector \mathbf{v}_{L_2} of line L_2 . We obtain the following vector:

$$(1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1),$$

which is the incidence vector of line $\{\alpha^0, \alpha^2, \alpha^6, \alpha^{14}\}$. If we cyclically shift the incidence vector \mathbf{v}_{L_1} of line L_1 , we obtain the following vector:

$$(1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0),$$

which is the incidence vector of line $\{\alpha^0, \alpha^8, \alpha^9, \alpha^{11}\}$.

DEFINITION 8.3 A (μ, s) th-order binary Euclidean geometry (EG) code of length $2^{ms} - 1$ is the largest cyclic code whose null space contains the incidence vectors of all the $(\mu + 1)$ -flats of $\text{EG}(m, 2^s)$ that do not pass through the origin.

Basically, the (μ, s) th-order EG code of length $2^{ms} - 1$ is the dual code (or null space) of the code (or space) spanned by the incidence vectors of the $(\mu + 1)$ -flats of $\text{EG}(m, 2^s)$ that do not pass through the origin. Due to the cyclic structural property of the incidence vectors of the flats in $\text{EG}(m, 2^s)$ not passing through the origin, the code spanned by the incidence vectors of $(\mu + 1)$ -flats not passing through the origin is cyclic and hence its dual code, the (μ, s) th-order EG code, is also cyclic.

The generator polynomial of a (μ, s) th-order EG code is given in terms of its roots in $GF(2^{ms})$. Let h be a nonnegative integer less than 2^{ms} . Then, we can express h in radix- 2^s form as follows:

$$h = \delta_0 + \delta_1 2^s + \delta_2 2^{2s} + \cdots + \delta_{m-1} 2^{(m-1)s},$$

where $0 \leq \delta_i < 2^s$ for $0 \leq i < m$. The 2^s -weight of h , denoted by $W_{2^s}(h)$, is defined as the real sum of the coefficients in the radix- 2^s expansion of h ; that is,

$$W_{2^s}(h) = \sum_{i=0}^{m-1} \delta_i. \quad (8.28)$$

As an example, let $m = 3$ and $s = 2$. Then, we can expand the integer $h = 45$ in radix- 2^2 form as follows:

$$45 = 1 + 3 \cdot 2^2 + 2 \cdot 2^{2 \times 2},$$

with $\delta_0 = 1$, $\delta_1 = 3$, and $\delta_2 = 2$. The 2^2 -weight of 45 is then

$$W_{2^2}(45) = 1 + 3 + 2 = 6.$$

Consider the difference $h - W_{2^s}(h)$, which we can express as follows:

$$h - W_{2^s}(h) = \delta_1(2^s - 1) + \delta_2(2^{2s} - 1) + \cdots + \delta_{m-1}(2^{(m-1)s} - 1).$$

It is clear from this difference that h is divisible by $2^s - 1$ if and only if its 2^s -weight, $W_{2^s}(h)$, is divisible by $2^s - 1$. Let $h^{(l)}$ be the remainder resulting from dividing $2^l h$ by $2^{ms} - 1$; that is,

$$2^l h = q(2^{ms} - 1) + h^{(l)},$$

with $0 \leq h^{(l)} < 2^{ms} - 1$. Clearly, $h^{(l)}$ is divisible by $2^s - 1$ if and only if h is divisible by $2^s - 1$. Note that $h^{(0)} = h$.

Now, we state a theorem (without proof) that characterizes the roots of the generator polynomial of a (μ, s) th-order EG code. The proof of this theorem can be found in [26, 27], and [33].

THEOREM 8.3 Let α be a primitive element of the Galois field $GF(2^{ms})$. Let h be a nonnegative integer less than $2^{ms} - 1$. The generator polynomial $\mathbf{g}(X)$ of the (μ, s) th-order EG code of length $2^{ms} - 1$ has α^h as a root if and only if

$$0 < \max_{0 \leq l < s} W_{2^s}(h^{(l)}) \leq (m - \mu - 1)(2^s - 1). \quad (8.29)$$

EXAMPLE 8.17

Let $m = 2$, $s = 2$, and $\mu = 0$. Then, the Galois field $GF(2^4)$ may be regarded as the Euclidean geometry $EG(2, 2^2)$ over $GF(2^2)$. Let α be a primitive element in $GF(2^4)$ (use Table 2.8). Let h be a nonnegative integer less than 15. It follows from Theorem 8.3 that the generator polynomial $\mathbf{g}(X)$ of the $(0, 2)$ th-order EG code of length 15 has α^h as a root if and only if

$$0 < \max_{0 \leq l < 2} W_{2^2}(h^{(l)}) \leq 3.$$

The nonnegative integers less than 15 that satisfy this condition are 1, 2, 3, 4, 6, 8, 9, and 12. Therefore, $\mathbf{g}(X)$ has $\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^6, \alpha^8, \alpha^9$, and α^{12} as all its roots. The elements $\alpha, \alpha^2, \alpha^4$, and α^8 have the same minimal polynomial, $\phi_1(X) = 1 + X + X^4$, and the elements $\alpha^3, \alpha^6, \alpha^9$, and α^{12} have the same minimal polynomial, $\phi_1(X) = 1 + X + X^2 + X^3 + X^4$. Thus, the generator polynomial of the $(0, 2)$ th-order EG code of length 15 is

$$\begin{aligned}\mathbf{g}(X) &= (1 + X + X^4)(1 + X + X^2 + X^3 + X^4) \\ &= 1 + X^4 + X^6 + X^7 + X^8.\end{aligned}$$

It is interesting to note that the $(0, 2)$ th-order EG code is the $(15, 7)$ BCH code considered in Example 8.1. It is one-step majority-logic decodable.

EXAMPLE 8.18

Let $m = 3$, $s = 2$, and $\mu = 1$. Then, the Galois field $GF(2^6)$ may be regarded as the Euclidean geometry $EG(3, 2^2)$ over $GF(2^2)$. Let α be a primitive element in $GF(2^6)$ (use Table 6.2). Let h be a nonnegative integer less than 63. It follows from Theorem 8.3 that the generator polynomial $\mathbf{g}(X)$ of the $(1, 2)$ th-order EG code of length 63 has α^h as a root if and only if

$$0 < \max_{0 \leq l < 2} W_{2^2}(h^{(l)}) \leq 3.$$

The nonnegative integers less than 63 that satisfy this condition are

$$1, 2, 3, 4, 6, 8, 9, 12, 16, 18, 24, 32, 33, 48.$$

Thus, $\mathbf{g}(X)$ has the following roots:

$$\alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^6, \alpha^8, \alpha^9, \alpha^{12}, \alpha^{16}, \alpha^{18}, \alpha^{24}, \alpha^{32}, \alpha^{33}, \alpha^{48}.$$

From Table 6.3 we find that

1. $\alpha, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}$, and α^{32} have $\phi_1(X) = 1 + X + X^6$ as their minimal polynomial.
2. $\alpha^3, \alpha^6, \alpha^{12}, \alpha^{24}, \alpha^{33}$, and α^{48} have $\phi_3(X) = 1 + X + X^2 + X^4 + X^6$ as their minimal polynomial.
3. α^9, α^{18} , and α^{36} have the same minimal polynomial, $\phi_9(X) = 1 + X^2 + X^3$.

Therefore, the generator polynomial of the (1, 2)th-order EG code of length 63 is

$$\begin{aligned}\mathbf{g}(X) &= (1 + X + X^6)(1 + X + X^2 + X^4 + X^6)(1 + X^2 + X^3) \\ &= 1 + X^2 + X^4 + X^{11} + X^{13} + X^{14} + X^{15}.\end{aligned}$$

Hence, the (1, 2)th-order EG code of length 63 is a (63, 48) cyclic code. Later we will show that this code is two-step orthogonalizable and is capable of correcting any combination of two or fewer errors.

Decoding of the (μ, s) th-order EG code of length $2^{ms} - 1$ is based on the structural properties of the Euclidean geometry $\text{EG}(m, 2^s)$. From Definition 8.3 we know that the null space of the code contains the incidence vectors of all the $(\mu + 1)$ -flats of $\text{EG}(m, 2^s)$ that do not pass through the origin. Let $F^{(\mu)}$ be a μ -flat passing through the point $\alpha^{2^{ms}-2}$. From (8.27) we know that there are

$$J = \frac{2^{(m-\mu)s} - 1}{2^s - 1} - 1 \quad (8.30)$$

$(\mu + 1)$ -flats not passing through the origin that intersect on $F^{(\mu)}$. The incidence vectors of these J $(\mu + 1)$ -flats are orthogonal on the digits at the locations that correspond to the points in $F^{(\mu)}$. Therefore, the parity-check sums formed from these J incidence vectors are orthogonal on the error digits at the locations corresponding to the points in $F^{(\mu)}$. If there are $\lfloor J/2 \rfloor$ or fewer errors in the received vector, the sum of errors at the locations corresponding to the points in $F^{(\mu)}$ can be determined correctly. Let us denote this error sum with $S(F^{(\mu)})$. In this manner the error sum $S(F^{(\mu)})$ can be determined for every μ -flat $F^{(\mu)}$ passing through the point $\alpha^{2^{ms}-2}$. This forms the first step of orthogonalization.

We then use the error sums $S(F^{(\mu)})$'s corresponding to all the μ -flats $F^{(\mu)}$ that pass through the point $\alpha^{2^{ms}-2}$ for the second step of orthogonalization. Let $F^{(\mu-1)}$ be a $(\mu - 1)$ -flat passing through the point $\alpha^{2^{ms}-2}$. From (8.27) we see that there are

$$J_1 = \frac{2^{(m-\mu+1)s} - 1}{2^s - 1} - 1 > J$$

μ -flats not passing through the origin that intersect on $F^{(\mu-1)}$. The error sums corresponding to these J_1 μ -flats are orthogonal on the error digits at the locations corresponding to the points in $F^{(\mu-1)}$. Let $S(F^{(\mu-1)})$ denote the sum of error digits at the locations corresponding to the points in $F^{(\mu-1)}$. Then, $S(F^{(\mu-1)})$ can be determined from the J_1 error sums $S(F^{(\mu)})$'s that are orthogonal on $S(F^{(\mu-1)})$. Since $J_1 > J$, if there are no more than $\lfloor J/2 \rfloor$ errors in the received vector, the error sum $S(F^{(\mu-1)})$ can be determined correctly. In this manner the error sum $S(F^{(\mu-1)})$ can be determined for every $(\mu - 1)$ -flat $F^{(\mu-1)}$ passing through the point $\alpha^{2^{ms}-2}$ but not the origin. This completes the second step of orthogonalization.

The error sums $S(F^{(\mu-1)})$'s now are used for the third step of orthogonalization. Let $F^{(\mu-2)}$ be a $(\mu - 2)$ -flat passing through the point $\alpha^{2^{ms}-2}$ but not the origin. From (8.27) we see that there are

$$J_2 = \frac{2^{(m-\mu+2)s} - 1}{2^s - 1} - 1 > J_1 > J$$

error sums $S(F^{(\mu-1)})$'s orthogonal on the error sum $S(F^{(\mu-2)})$. Hence, $S(F^{(\mu-2)})$ can be determined correctly. The error sums $S(F^{(\mu-2)})$'s are then used for the next step of orthogonalization. This process continues until the error sums corresponding to all the 1-flats (lines) passing through the point $\alpha^{2^{ms}-2}$ but not the origin are determined. There are

$$J_\mu = \frac{2^{ms} - 1}{2^s - 1} > J_{\mu-1} > \cdots > J_1 > J$$

such error sums orthogonal on the error digit $e_{2^{ms}-2}$ at the location $\alpha^{2^{ms}-2}$. Thus, $e_{2^{ms}-2}$ can be determined correctly from these orthogonal error sums provided that there are no more than $\lfloor J/2 \rfloor$ errors in the received vector. Because the code is cyclic, other error digits can successively be decoded in the same manner.

Because the decoding of each error digit requires $\mu + 1$ steps of orthogonalization, the (μ, s) th-order EG code of length $2^{ms} - 1$ is therefore $(\mu + 1)$ -step majority-logic decodable. The code is capable of correcting

$$t_{ML} = \left\lfloor \frac{2^{(m-\mu)s} - 1}{2(2^s - 1)} - \frac{1}{2} \right\rfloor \quad (8.31)$$

or fewer errors. Therefore, its minimum distance is at least

$$2t_{ML} + 1 = 2^s(2^{(m-\mu-2)s} + \cdots + 2^s + 1). \quad (8.32)$$

Note that at each step of orthogonalization we need only J orthogonal error sums to determine an error sum for the next step. For $\mu = 0$, a $(0, s)$ th-order EG code is one-step majority-logic decodable.

EXAMPLE 8.19

Let $m = 2$, $s = 2$ and $\mu = 0$. Consider the $(0, 2)$ th-order EG code of length 15. From Example 8.17 we know that this code is the $(15, 7)$ BCH code (also a type-1 DTI code). The null space of this code contains the incidence vectors of all the 1-flats (lines) in $\text{EG}(2, 2^2)$ that do not pass through the origin. To decode e_{14} , we need to determine the incidence vectors of the 1-flat passing through the point α^{14} , where α is a primitive element in $GF(2^4)$. There are

$$J = \frac{2^{2 \cdot 2} - 1}{2^2 - 1} - 1 = 4$$

such incidence vectors, which are given in Example 8.16. These four vectors are orthogonal on the digit position α^{14} . In fact, these are exactly the four orthogonal vectors $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3$, and \mathbf{w}_4 given in Example 8.1.

EXAMPLE 8.20

Let $m = 4$, $s = 1$, and $\mu = 1$. Consider the $(1, 1)$ th-order EG code of length $2^4 - 1 = 15$. Let α be a primitive element of $GF(2^4)$ given by Table 2.8. Let h be

a nonnegative integer less than 15. It follows from Theorem 8.3 that the generator polynomial $\mathbf{g}(X)$ of this code has α^h as a root if and only if

$$0 < W_2(h^{(0)}) \leq 2.$$

Note that $h^{(0)} = h$. From the preceding condition we find that $\mathbf{g}(X)$ has the following roots: $\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^8, \alpha^9, \alpha^{10}$, and α^{12} . From Table 2.9 we find that

$$\begin{aligned}\mathbf{g}(X) &= (1 + X + X^4)(1 + X + X^2 + X^3 + X^4)(1 + X + X^2) \\ &= 1 + X + X^2 + X^4 + X^5 + X^8 + X^{10}.\end{aligned}$$

It is interesting to note that this EG code is actually the $(15, 5)$ BCH code studied in Example 8.12.

The null space of this code contains the incidence vectors of all the 2-flats of the EG(4, 2) that do not pass through the origin. Now, we will show how to form orthogonal check-sums based on the structure of EG(4, 2). First, we treat $GF(2^4)$ as the geometry EG(4, 2). A 1-flat passing through the point α^{14} consists of the points of the form $\alpha^{14} + a\alpha^i$ with $a \in GF(2)$. There are thirteen 1-flats passing through α^{14} but not the origin, $\alpha^\infty = 0$; they are

$$\begin{aligned}&\{\alpha^{13}, \alpha^{14}\}, \{\alpha^{12}, \alpha^{14}\}, \{\alpha^{11}, \alpha^{14}\}, \{\alpha^{10}, \alpha^{14}\}, \{\alpha^9, \alpha^{14}\}, \{\alpha^8, \alpha^{14}\}, \\ &\{\alpha^7, \alpha^{14}\}, \{\alpha^6, \alpha^{14}\}, \{\alpha^5, \alpha^{14}\}, \{\alpha^4, \alpha^{14}\}, \{\alpha^3, \alpha^{14}\}, \{\alpha^2, \alpha^{14}\}, \{\alpha, \alpha^{14}\}.\end{aligned}$$

For each of these 1-flats, there are

$$J = \frac{2^{(4-1)\cdot 1} - 1}{2^1 - 1} - 1 = 6$$

2-flats not passing through the origin that intersect on it. Each of these 2-flats consists of the points of the form $\alpha^{14} + a\alpha^i + b\alpha^j$, with a and b in $GF(2)$. The six 2-flats that intersect on the 1-flat $\{\alpha^{13}, \alpha^{14}\}$ are

$$\begin{aligned}&\{\alpha^4, \alpha^{10}, \alpha^{13}, \alpha^{14}\}, \quad \{\alpha^7, \alpha^{12}, \alpha^{13}, \alpha^{14}\}, \quad \{\alpha^9, \alpha^{11}, \alpha^{13}, \alpha^{14}\}, \\ &\{\alpha^0, \alpha^8, \alpha^{13}, \alpha^{14}\}, \quad \{\alpha^1, \alpha^5, \alpha^{13}, \alpha^{14}\}, \quad \{\alpha^3, \alpha^6, \alpha^{13}, \alpha^{14}\}.\end{aligned}$$

The incidence vectors of these six 2-flats are

$$\begin{array}{cccccccccccccccccc} & \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\ \mathbf{w}_{11} & = & (0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1) \\ \mathbf{w}_{12} & = & (0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1) \\ \mathbf{w}_{13} & = & (0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1) \\ \mathbf{w}_{14} & = & (1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1) \\ \mathbf{w}_{15} & = & (0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1) \\ \mathbf{w}_{16} & = & (0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1). \end{array}$$

Clearly, these six vectors are orthogonal on digits at locations α^{13} and α^{14} . Let \mathbf{r} be the received vector. The parity-check sums formed from these six

orthogonal vectors are

$$\begin{aligned} A_{11} &= \mathbf{w}_{11} \cdot r = e_4 + e_{10} + e_{13} + e_{14} \\ A_{12} &= \mathbf{w}_{12} \cdot r = e_7 + e_{12} + e_{13} + e_{14} \\ A_{13} &= \mathbf{w}_{13} \cdot r = e_9 + e_{11} + e_{13} + e_{14} \\ A_{14} &= \mathbf{w}_{14} \cdot r = e_0 + e_8 + e_{13} + e_{14} \\ A_{15} &= \mathbf{w}_{15} \cdot r = e_1 + e_5 + e_{13} + e_{14} \\ A_{16} &= \mathbf{w}_{16} \cdot r = e_3 + e_6 + e_{13} + e_{14}. \end{aligned}$$

We see that these six check-sums orthogonal on $\{e_{13}, e_{14}\}$ are exactly the same check sums given in Example 8.12. Thus, we can determine the error sum $e_{13} + e_{14}$ corresponding to the 1-flat $\{\alpha^{13}, \alpha^{14}\}$ from these six check-sums.

In the same manner we can determine the error sums corresponding to the other twelve 1-flats passing through α^{14} . Because $J = 6$, we need to determine only six error sums corresponding to any six 1-flats passing through α^{14} . We then use these error sums to determine e_{14} . Thus, the $(1, 1)$ th-order EG code of length 15 is a two-step majority-logic decodable code.

Except for certain special cases, there is no simple formula for enumerating the number of parity-check digits of EG codes. Complicated combinatorial expressions for the number of parity-check digits of EG codes can be found in [17] and [18]. One special case is $\mu = m - 2$. The number of parity-check digits for a $(m - 2, s)$ th-order EG code of length $2^{ms} - 1$ is

$$n - k = \binom{m+1}{m}^s - 1. \quad (8.33)$$

This result was obtained independently by Smith [19] and by MacWilliams and Mann [20].

For $s = 1$, we obtain another special subclass of EG codes, which happens to be the class of RM codes of length $2^m - 1$ in cyclic form [11, 21–29]. A μ th-order cyclic RM is simply a $(\mu, 1)$ th-order EG code. If we add an overall parity bit to each codeword of this code, we obtain the μ -th order RM code of length 2^m presented in Section 4.3. Let α be a primitive element of the Galois field $GF(2^m)$. Let h be a nonnegative integer less than 2^m . It follows from Theorem 8.3 that the generator polynomial $\mathbf{g}(X)$ of the μ th-order cyclic RM code of length $2^m - 1$ has α^h as a root if and only if

$$0 < W_2(h) \leq m - \mu - 1. \quad (8.34)$$

The μ th-order cyclic RM code of length $2^m - 1$ has the following parameters:

$$k = \sum_{i=0}^{\mu} \binom{m}{i},$$

$$d_{min} = 2^{m-\mu} - 1,$$

$$J = 2^{m-\mu} - 2.$$

Because $J = d_{min} - 1$, cyclic RM codes are completely orthogonalizable. The cyclic structure of RM codes was proved independently by Kasami et al. [21, 22] and Kolesnik and Mironchikov [23].

Except for RM codes and other special cases, EG codes in general are not completely orthogonalizable. For moderate-length n , the error-correcting capability of an EG code is slightly inferior to that of a comparable BCH code; however, the majority-logic decoding for EG codes is more simply implemented than the decoding for BCH codes. Thus, for moderate n , EG codes provide rather effective error control. For large-length n , EG codes become much inferior to the comparable BCH codes, and the number of majority-logic gates required for decoding becomes prohibitively large. In this case, BCH codes are definitely superior to the EG codes in error-correcting capability and decoding complexity. A list of EG codes with $n \leq 1023$ is given in Table 8.6. See [24] for a more extensive list.

TABLE 8.6: A list of EG codes.

m	s	μ	n	k	J	t_{ML}
3	1	1	7	4	2	1
4	1	2	15	11	2	1
4	1	1	15	5	6	3
2	2	0	15	7	4	2
5	1	3	31	26	2	1
5	1	2	31	16	6	3
5	1	1	31	6	14	7
6	1	4	63	57	2	1
6	1	3	63	42	6	3
6	1	2	63	22	14	7
6	1	1	63	7	31	15
3	2	1	63	48	4	2
3	2	0	63	13	20	10
2	3	0	63	37	8	4
7	1	5	127	120	2	1
7	1	4	127	99	6	3
7	1	3	127	64	14	7
7	1	2	127	29	30	15
7	1	1	127	8	62	31
8	1	6	255	247	2	1
8	1	5	255	219	6	3
8	1	4	255	163	14	7
8	1	3	255	93	30	15
8	1	2	255	37	62	31
8	1	1	255	9	126	63
4	2	2	255	231	4	2
4	2	1	255	127	20	10
4	2	0	255	21	84	42
2	4	0	255	175	16	8
9	1	7	511	502	2	1
9	1	6	511	466	6	3
9	1	5	511	382	14	7

TABLE 8.6: (*continued*)

<i>m</i>	<i>s</i>	<i>μ</i>	<i>n</i>	<i>k</i>	<i>J</i>	<i>t_{ML}</i>
9	1	4	511	256	30	15
9	1	3	511	130	62	31
9	1	2	511	46	126	63
9	1	1	511	10	254	127
3	3	1	511	448	8	4
3	3	0	511	139	72	36
10	1	8	1023	1013	2	1
10	1	7	1023	968	6	3
10	1	6	1023	848	14	7
10	1	5	1023	638	30	15
10	1	4	1023	386	62	31
10	1	3	1023	176	126	63
10	1	2	1023	56	254	127
10	1	1	1023	11	510	255
5	2	3	1023	988	4	2
5	2	2	1023	748	20	10
5	2	1	1023	288	84	42
5	2	0	1023	31	340	170
2	5	0	1023	781	32	16

A very special subclass of EG codes is the subclass of codes with $m = 2$ and $\mu = 0$. A code in this subclass is a $(0, s)$ th-order EG code of length $n = 2^{2s} - 1$. The null space of this code contains the incidence vectors of all the lines in $\text{EG}(2, 2^s)$ not passing through the origin. It follows from (8.30) that 2^s check-sums orthogonal on any code digit can be formed. Therefore, the minimum distance of the code is at least $2^s + 1$. It follows from (8.29) that the generator polynomial $\mathbf{g}(X)$ of this code has $\alpha^1, \alpha^2, \dots, \alpha^{2^s}$ and their conjugates as roots. The polynomial $X^{2^{2s}-1} + 1$ can be factored as follows:

$$X^{2^{2s}-1} + 1 = (X^{2^s-1} + 1)(X^{2^s(2^s-1)} + \dots + X^{2^s-1} + 1).$$

The first factor, $X^{2^s-1} + 1$, has $\alpha^0 = 1, \alpha^{2^s+1}, \alpha^{2(2^s+1)}, \dots, \alpha^{(2^s-2)(2^s+1)}$ as all its roots. Then, the second factor, $\mathbf{v}(X) = 1 + X^{2^s-1} + X^{2(2^s-1)} + \dots + X^{2^s(2^s-1)}$, has $\alpha^1, \alpha^2, \dots, \alpha^{2^s}$ as roots. Therefore, $\mathbf{v}(X)$ must be a multiple of $\mathbf{g}(x)$ (or divisible by $\mathbf{g}(X)$) and hence it is a code polynomial of the $(0, s)$ th-order EG code of length $n = 2^{2s} - 1$. This code polynomial $\mathbf{v}(x)$ has a weight of exactly $2^s + 1$. This weight together with the bound that the minimum distance of the code is at least $2^s + 1$ imply that the minimum distance of the $(0, s)$ th-order EG code of length $n = 2^{2s} - 1$ is exactly $2^s + 1$. Therefore, the code is one-step completely orthogonalizable. It follows from (8.33) that the number of parity-check digits of the $(0, s)$ th-order EG code constructed based on the two-dimensional Euclidean geometry $\text{EG}(2, 2^s)$ is

$$n - k = 3^s - 1. \quad (8.35)$$

It is interesting to note that a $(0, s)$ th-order EG code of length $n = 2^{2s} - 1$ is also a type-I DTI code given in Section 8.2.

EXAMPLE 8.21

Consider the $(0, 6)$ th-order EG code of length $n = 2^{2 \times 6} - 1 = 4095$ constructed based on the lines in $\text{EG}(2, 2^6)$ not passing through the origin. This code is a $(4095, 3367)$ cyclic code with a minimum distance of 65 and a rate of 0.83. Let α be a primitive element of $GF(2^{12})$. The generator polynomial of this code has $\alpha^1, \alpha^2, \dots, \alpha^{64}$ as roots. It is one-step completely orthogonalizable and can correct up to 32 errors with one-step majority-logic decoding, either type I or type II. The error performance of this code on an AWGN channel with BPSK signaling and one-step majority-logic decoding is shown in Figure 8.10. At the BER of 10^{-5} , it achieves a 4-dB coding gain over the uncoded BPSK. The majority-logic decoder for this code can easily be implemented in hardware with a feedback shift register of 728 flip-flops and a majority-logic circuit with 64 inputs. This hardware implementation can achieve very high decoding speed and is quite suitable for high-speed optical networks operating at 10 Gbits or for high-speed satellite communications. Consider the NASA standard $(255, 223, 33)$ RS code over $GF(2^8)$. For binary transmission,

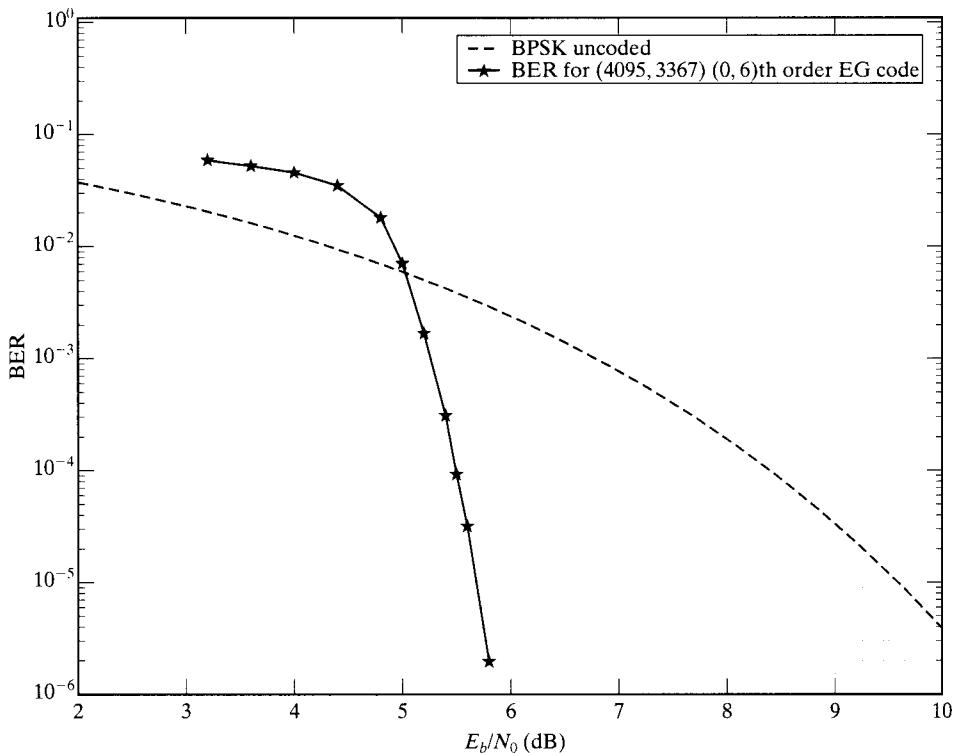


FIGURE 8.10: Bit-error performance of the $(4095, 3367)$ $(0, 6)$ th-order EG code with majority-logic decoding.

each code symbol is expanded into an 8-bit byte. This symbol-to-binary expansion results in a (2040, 1784) binary code. At the receiving end, the received digits are grouped back into symbols in $GF(2^8)$ for decoding. From Figures 7.3 and 8.10 we see that the (4095, 3367) (0, 6)th-order EG code outperforms the (255, 223, 33) RS code by more than 0.5 dB at the BER of 10^{-5} . Even though the (4095, 3367) EG code is twice as long as the (255, 223, 33) RS code in binary form, its decoding complexity is much simpler, because majority-logic decoding requires only simple binary logic operations, whereas decoding of the (255, 223, 33) RS code with algebraic decoding algorithms requires computations in $GF(2^8)$ to find the error-location polynomial and the error-value enumerator. The (4095, 3367) EG code can also be decoded with several other hard- or soft-decision decoding methods to achieve better error performance at the expense of increasing decoding complexity. This topic will be discussed in Chapter 17.

EG codes were first studied by Rudolph [3]. Rudolph's work was later extended and generalized by other coding theorists [24–30]. Improvements for decoding EG codes were suggested by Weldon [31] and by Chen [32]. Chen proved that any EG code can be decoded in *no more* than three steps. Chen's decoding algorithm is based on further structure of the Euclidean geometry, which is not covered in this introductory book.

There are several classes of generalized EG codes [24, 28–30, 34] that all contain EG codes as subclasses. We will not cover these generalizations here; however, we present a simple generalization using parallel flats next.

8.7 TWOFOLD EG CODES

Let F and F_1 be any two parallel μ -flats in $EG(m, 2^s)$. We say that F and F_1 form a $(\mu, 2)$ -frame in $EG(m, 2^s)$, denoted by $\{F, F_1\}$. Because F and F_1 do not have any point in common, the $(\mu, 2)$ -frame $\{F, F_1\}$ consists of $2^{\mu s+1}$ points. Let F_2 be another μ -flat parallel to F and F_1 . Then, the two $(\mu, 2)$ -frames $\{F, F_1\}$ and $\{F, F_2\}$ intersect on F . Let L be a $(\mu+1)$ -flat that contains the μ -flat F . Then, L contains $2^s - 1$ other μ -flats that are parallel to F . Each of these $2^s - 1$ μ -flats together with F forms a $(\mu, 2)$ -frame. There are $2^s - 1$ such $(\mu, 2)$ -frames that intersect on F . Clearly, these $2^s - 1$ $(\mu, 2)$ -frames are all contained in the $(\mu+1)$ -flat L . Any point in L but outside F is in *on one and only one of these $2^s - 1$ $(\mu, 2)$ -frames*. Because there are

$$\frac{2^{(m-\mu)s} - 1}{2^s - 1}$$

$(\mu+1)$ -flats that intersect on F , there are

$$(2^s - 1) \cdot \frac{2^{(m-\mu)s} - 1}{2^s - 1} = 2^{(m-\mu)s} - 1 \quad (8.36)$$

$(\mu, 2)$ -frames that intersect on F . Any point outside F is in on one and only one of these $(\mu, 2)$ -frames. We say that these $(\mu, 2)$ -frames are orthogonal on the μ -flat F . If F does not pass through the origin, there are

$$2^{(m-\mu)s} - 2 \quad (8.37)$$

$(\mu, 2)$ -frames that are orthogonal on F and do not pass through the origin.

Again, we regard the Galois field $GF(2^{ms})$ as the geometry $EG(m, 2^s)$. Let α be a primitive element of $GF(2^{ms})$. For any $(2^{ms} - 1)$ -tuple

$$\mathbf{v} = (v_0, v_1, \dots, v_{2^{ms}-2})$$

over $GF(2)$, we again number its components with the nonzero elements of $GF(2^{ms})$ as usual (i.e., v_i is numbered with α^i for $0 \leq i < 2^{ms} - 1$). For each $(\mu, 2)$ -frame Q in $EG(m, 2^s)$, we define its incidence vector as follows:

$$\mathbf{v}_Q = (v_0, v_1, \dots, v_{2^{ms}-2}),$$

where the i th component is

$$v_i = \begin{cases} 1 & \text{if } \alpha^i \text{ is a point in } Q, \\ 0 & \text{otherwise.} \end{cases}$$

DEFINITION 8.4 A (μ, s) th-order twofold EG code of length $2^{ms} - 1$ is the largest cyclic code whose null space contains the incidence vectors of all the $(\mu, 2)$ -frames in $EG(m, 2^s)$ that do not pass through the origin.

We now state a theorem (without proof) [34] that characterizes the roots of the generator polynomial of a (μ, s) th-order twofold EG code.

THEOREM 8.4 Let α be a primitive element of the Galois field $GF(2^{ms})$. Let h be a nonnegative integer less than $2^{ms} - 1$. The generator polynomial $\mathbf{g}(X)$ of the (μ, s) th-order twofold EG code of length $2^{ms} - 1$ has α^h as a root if and only if

$$0 < \max_{0 \leq l < s} W_{2^s}(h^{(l)}) < (m - \mu)(2^s - 1). \quad (8.38)$$

EXAMPLE 8.22

Let $m = 2$, $s = 3$, and $\mu = 1$. Consider the $(1, 3)$ th-order twofold EG code of length 63. Let α be a primitive element of $GF(2^6)$ given by Table 6.2. Let h be a nonnegative integer less than 63. It follows from (8.38) that the generator polynomial $\mathbf{g}(X)$ of the $(1, 3)$ th-order twofold EG code of length 63 has α^h as a root if and only if

$$0 < \max_{0 \leq l < 3} W_{2^3}(h^{(l)}) < 7.$$

The nonnegative integers less than 63 that satisfy this condition are

$$1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 16, 17, 20, 24, 32, 33, 34, 40, 48.$$

Thus, the generator polynomial $\mathbf{g}(X)$ has the following roots:

$$\begin{aligned} &\alpha_1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{12}, \\ &\alpha^{16}, \alpha^{17}, \alpha^{20}, \alpha^{24}, \alpha^{32}, \alpha^{33}, \alpha^{34}, \alpha^{40}, \alpha^{48}. \end{aligned}$$

From Table 6.3 we find that:

1. The roots $\alpha, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}$, and α^{32} have the same minimal polynomial, $\phi_1(X) = 1 + X + X^6$.

2. The roots $\alpha^3, \alpha^6, \alpha^{12}, \alpha^{24}, \alpha^{48}$ and α^{33} have the same minimal polynomial, $\phi_3(X) = 1 + X + X^2 + X^4 + X^6$.
3. The roots $\alpha^5, \alpha^{10}, \alpha^{20}, \alpha^{40}, \alpha^{17}$, and α^{34} have the same minimal polynomial, $\phi_5(X) = 1 + X + X^2 + X^5 + X^6$.

Therefore,

$$\begin{aligned} \mathbf{g}(X) &= \phi_1(X) \cdot \phi_3(X) \cdot \phi_5(X) \\ &= 1 + X + X^2 + X^3 + X^6 + X^7 + X^9 + X^{15} + X^{16} + X^{17} + X^{18}. \end{aligned}$$

Therefore, the $(1, 3)$ th-order twofold EG code of length 63 with $m = 2$ is a $(63, 45)$ cyclic code. In fact, it is the $(63, 45)$ BCH code with a minimum distance equal to 7.

To decode the (μ, s) th-order twofold EG code of length $2^{ms} - 1$, we first form the parity-check sums from the incidence vectors of all the $(\mu, 2)$ -frames in $\text{EG}(m, 2^s)$ that do not pass through the origin (note that these incidence vectors are in the null space of the code). Let $F^{(\mu)}$ be a μ -flat that passes through the point $\alpha^{2^{ms}-2}$. From (8.37) we see that there are

$$J = 2^{(m-\mu)s} - 2 \quad (8.39)$$

$(\mu, 2)$ -frames not passing through the origin that are orthogonal on $F^{(\mu)}$. The incidence vectors of these $(\mu, 2)$ -frames are orthogonal on the digits at the locations that correspond to the points in $F^{(\mu)}$. Therefore, the parity-check sums formed from these J incidence vectors are orthogonal on the error digits at the locations that correspond to the points in $F^{(\mu)}$. Let $S(F^{(\mu)})$ denote the sum of error digits at the locations corresponding to the points in $F^{(\mu)}$. Then, we can correctly determine this error sum, $S(F^{(\mu)})$, from the J check-sums orthogonal on it provided that there are no more than

$$\left\lfloor \frac{J}{2} \right\rfloor = 2^{(m-\mu)s-1} - 1$$

errors in the received vector. In this manner we can determine the error sums, $S(F^{(\mu)})$'s that correspond to all the μ -flats passing through the point $\alpha^{2^{ms}-2}$. This completes the first step of orthogonalization. After this step, the rest of orthogonalization steps are the same as those for a (μ, s) th-order EG code. Therefore, a total of $\mu + 1$ steps of orthogonalization are needed to decode a (μ, s) th-order twofold EG code.

We can easily check that at each decoding step there are at least $J = 2^{(m-\mu)s} - 2$ error sums orthogonal on an error sum for the next step. Thus, the (μ, s) th-order twofold EG code of length $2^{ms} - 1$ is capable of correcting.

$$t_{ML} = \left\lfloor \frac{J}{2} \right\rfloor = 2^{(m-\mu)s-1} - 1 \quad (8.40)$$

or fewer errors with majority-logic decoding. It has been proved [34] that the minimum distance of the (μ, s) th-order twofold EG code of length $2^{ms} - 1$ is exactly $2^{(m-\mu)s} - 1$. Therefore, the class of twofold EG codes is completely orthogonalizable.

EXAMPLE 8.23

Consider the decoding of the $(1, 3)$ th-order twofold EG of length 63 with $m = 2$ and $s = 3$. In Example 8.22 we showed that this code is a $(63, 45)$ cyclic code (also a BCH code). The null space of this code contains the incidence vectors of all the $(1, 2)$ -frames in $\text{EG}(2, 2^3)$ that do not pass through the origin. Regard $GF(2^6)$ as the geometry $\text{EG}(2, 2^3)$. Let α be a primitive element of $GF(2^6)$ (use Table 6.2). From (8.26) we see that there are nine lines in $\text{EG}(2, 2^3)$ that intersect at the point α^{62} . Eight of these lines do not pass through the origin. From (8.37) we see that for each of these eight lines there are six $(1, 2)$ -frames intersecting on it. The incidence vectors of these six $(1, 2)$ -frames are in the null space of the code, and they will be used to form parity-check sums for decoding the error digit e_{62} at location α^{62} . Because $J = 6$, we need to find only six lines in $\text{EG}(2, 2^3)$ that intersect at the point α^{62} and do not pass through the origin.

Let $\beta = \alpha^9$. Then, $0, 1, \beta, \beta^2, \beta^3, \beta^4, \beta^5$, and $\beta^6 (\beta^7 = 1)$ form a subfield $GF(2^3)$ of the field $GF(2^6)$ (use Table 6.2). Then, each line in $\text{EG}(2, 2^3)$ that passes through α^{62} consists of the following points:

$$\alpha^{62} + \eta\alpha^j$$

where $\eta \in \{0, 1, \beta, \beta^2, \beta^3, \beta^4, \beta^5, \beta^6\}$. Six lines passing through the point α^{62} are as follows:

$$\begin{aligned} L_1 &= \{\alpha^{11}, \alpha^{16}, \alpha^{18}, \alpha^{24}, \alpha^{48}, \alpha^{58}, \alpha^{59}, \alpha^{62}\}, \\ L_2 &= \{\alpha^1, \alpha^7, \alpha^{31}, \alpha^{41}, \alpha^{42}, \alpha^{45}, \alpha^{57}, \alpha^{62}\}, \\ L_3 &= \{\alpha^{23}, \alpha^{33}, \alpha^{34}, \alpha^{37}, \alpha^{49}, \alpha^{54}, \alpha^{56}, \alpha^{62}\}, \\ L_4 &= \{\alpha^2, \alpha^{12}, \alpha^{19}, \alpha^{21}, \alpha^{27}, \alpha^{51}, \alpha^{61}, \alpha^{62}\}, \\ L_5 &= \{\alpha^0, \alpha^3, \alpha^{15}, \alpha^{20}, \alpha^{22}, \alpha^{28}, \alpha^{52}, \alpha^{62}\}, \\ L_6 &= \{\alpha^9, \alpha^{10}, \alpha^{13}, \alpha^{25}, \alpha^{30}, \alpha^{32}, \alpha^{38}, \alpha^{62}\}. \end{aligned}$$

For each of these lines we form six $(1, 2)$ -frames intersecting on it. A $(1, 2)$ -frame that contains the line $\{\alpha^{62} + \eta\alpha^j\}$ is of the form

$$(\{\alpha^{62} + \eta\alpha^j\}, \{\alpha^{62} + \alpha^i + \eta\alpha^j\}),$$

where α^i is not in $\{\alpha^{62} + \eta\alpha^j\}$. Line L_1 consists of the points $\{\alpha^{62} + \eta\alpha\}$. The point α^9 is not in L_1 . Then, the line $\{\alpha^{62} + \alpha^9 + \eta\alpha\}$ is parallel to $\{\alpha^{62} + \eta\alpha\}$. Thus,

$$(\{\alpha^{62} + \eta\alpha\}, \{\alpha^{62} + \alpha^9 + \eta\alpha\})$$

forms a $(1, 2)$ -frame containing the line L_1 . This $(1, 2)$ -frame consists of the following points:

$$\{\alpha^{11}, \alpha^{16}, \alpha^{18}, \alpha^{21}, \alpha^{24}, \alpha^{31}, \alpha^{32}, \alpha^{35}, \alpha^{47}, \alpha^{48}, \alpha^{52}, \alpha^{54}, \alpha^{58}, \alpha^{59}, \alpha^{60}, \alpha^{62}\}.$$

In this manner, for each line L_i , we can form six $(1, 2)$ -frames orthogonal on it. The incidence vectors of these 36 $(1, 2)$ -frames are given in Tables 8.7A through 8.7F.

TABLE 8.7A: Polynomials orthogonal on $\{e_{11}, e_{16}, e_{18}, e_{24}, e_{48}, e_{58}, e_{59}, e_{62}\}$.

$w_{11}(X)^* = (11, 16, 18, 21, 24, 31, 32, 35, 47, 48, 52, 54, 58, 59, 60, 62)$
$w_{12}(X) = (11, 12, 16, 18, 22, 23, 24, 26, 38, 43, 45, 48, 51, 58, 59, 62)$
$w_{13}(X) = (0, 6, 11, 16, 18, 24, 30, 40, 41, 44, 48, 56, 58, 59, 61, 62)$
$w_{14}(X) = (4, 5, 8, 11, 16, 18, 20, 24, 25, 27, 33, 48, 57, 58, 59, 62)$
$w_{15}(X) = (3, 11, 13, 14, 16, 17, 18, 24, 29, 34, 36, 42, 48, 58, 59, 62)$
$w_{16}(X) = (2, 7, 9, 11, 15, 16, 18, 24, 39, 48, 49, 50, 53, 58, 59, 62)$

*In Tables 8.7A through 8.7F, the integers inside the parentheses are powers of X .

TABLE 8.7B: Polynomials orthogonal on $\{e_1, e_7, e_{31}, e_{41}, e_{42}, e_{45}, e_{57}, e_{62}\}$.

$w_{21}(X) = (1, 7, 13, 23, 24, 27, 31, 39, 41, 42, 44, 45, 46, 52, 57, 62)$
$w_{22}(X) = (1, 7, 22, 31, 32, 33, 36, 41, 42, 45, 48, 53, 55, 57, 61, 62)$
$w_{23}(X) = (0, 1, 7, 12, 17, 19, 25, 31, 41, 42, 45, 49, 57, 59, 60, 62)$
$w_{24}(X) = (1, 5, 6, 7, 9, 21, 26, 28, 31, 34, 41, 42, 45, 57, 58, 62)$
$w_{25}(X) = (1, 3, 7, 8, 10, 16, 31, 40, 41, 42, 45, 50, 51, 54, 57, 62)$
$w_{26}(X) = (1, 4, 7, 14, 15, 18, 30, 31, 35, 37, 41, 42, 43, 45, 57, 62)$

TABLE 8.7C: Polynomials orthogonal on $\{e_{23}, e_{33}, e_{34}, e_{37}, e_{49}, e_{54}, e_{56}, e_{62}\}$.

$w_{31}(X) = (4, 9, 11, 17, 23, 33, 34, 37, 41, 49, 51, 52, 54, 55, 56, 62)$
$w_{32}(X) = (5, 15, 16, 19, 23, 31, 33, 34, 36, 37, 38, 44, 49, 54, 56, 62)$
$w_{33}(X) = (1, 13, 18, 20, 23, 26, 33, 34, 37, 42, 43, 46, 49, 54, 56, 58, 62)$
$w_{34}(X) = (0, 2, 8, 23, 32, 33, 34, 37, 42, 43, 46, 49, 54, 56, 58, 62)$
$w_{35}(X) = (14, 23, 24, 25, 28, 33, 34, 37, 40, 45, 47, 49, 53, 54, 56, 62)$
$w_{36}(X) = (6, 7, 10, 22, 23, 27, 29, 33, 34, 35, 37, 49, 54, 56, 59, 62)$

TABLE 8.7D: Polynomials orthogonal on $\{e_2, e_{14}, e_{19}, e_{21}, e_{27}, e_{51}, e_{61}, e_{62}\}$.

$w_{41}(X) = (2, 7, 8, 11, 14, 19, 21, 23, 27, 28, 30, 36, 51, 60, 61, 62)$
$w_{42}(X) = (0, 2, 14, 19, 21, 24, 27, 34, 35, 38, 50, 51, 55, 57, 61, 62)$
$w_{43}(X) = (2, 14, 15, 19, 21, 25, 26, 27, 29, 41, 46, 48, 51, 54, 61, 62)$
$w_{44}(X) = (1, 2, 3, 9, 14, 19, 21, 27, 33, 43, 44, 47, 51, 59, 61, 62)$
$w_{45}(X) = (2, 6, 14, 16, 17, 19, 20, 21, 27, 32, 37, 39, 45, 51, 61, 62)$
$w_{46}(X) = (2, 5, 10, 12, 14, 18, 19, 21, 27, 42, 51, 52, 53, 56, 61, 62)$

To decode the code, the incidence vectors of the 36 (1, 2)-frames given in Tables 8.7A through 8.7F are used to form parity-check sums. Let $S(L_i)$ denote the sum of error digits at the locations corresponding to the points on line L_i for $1 \leq i \leq 6$. Then, for each error sum $S(L_i)$, there are six parity-check sums orthogonal

TABLE 8.7E: Polynomials orthogonal on $\{e_0, e_3, e_{15}, e_{20}, e_{22}, e_{28}, e_{52}, e_{62}\}$.

$w_{51}(X) = (0, 3, 6, 11, 13, 15, 19, 20, 22, 28, 43, 52, 53, 54, 57, 62)$
$w_{52}(X) = (0, 3, 8, 9, 12, 15, 20, 22, 24, 28, 29, 31, 37, 52, 61, 62)$
$w_{53}(X) = (0, 1, 3, 15, 20, 22, 25, 28, 35, 36, 39, 51, 52, 56, 58, 62)$
$w_{54}(X) = (0, 3, 15, 16, 20, 22, 26, 27, 28, 30, 42, 47, 49, 52, 55, 62)$
$w_{55}(X) = (0, 2, 3, 4, 10, 15, 20, 22, 28, 34, 44, 45, 48, 52, 60, 62)$
$w_{56}(X) = (0, 3, 7, 15, 17, 18, 20, 21, 22, 28, 33, 38, 40, 46, 52, 62)$

TABLE 8.7F: Polynomials orthogonal on $\{e_9, e_{10}, e_{13}, e_{25}, e_{30}, e_{32}, e_{38}, e_{62}\}$.

$w_{61}(X) = (3, 5, 9, 10, 11, 13, 25, 30, 32, 35, 38, 45, 46, 49, 61, 62)$
$w_{62}(X) = (9, 10, 13, 17, 25, 27, 28, 30, 31, 32, 38, 43, 48, 50, 56, 62)$
$w_{63}(X) = (0, 1, 4, 9, 10, 13, 16, 21, 23, 25, 29, 30, 32, 38, 53, 62)$
$w_{64}(X) = (8, 9, 10, 13, 18, 19, 22, 25, 30, 32, 34, 38, 39, 41, 47, 62)$
$w_{65}(X) = (7, 9, 10, 12, 13, 14, 20, 25, 30, 32, 38, 44, 54, 55, 58, 62)$
$w_{66}(X) = (2, 9, 10, 13, 25, 26, 30, 32, 36, 37, 38, 40, 52, 57, 59, 62)$

on it. Thus, $S(L_i)$ can be determined correctly provided that there are three or fewer errors in the error vector.

The error sums $S(L_1), S(L_2), S(L_3), S(L_4), S(L_5)$, and $S(L_6)$ are orthogonal on e_{62} . Consequently, e_{62} can be determined from these error sums. Thus, the $(1, 3)$ th-order twofold (63, 45) EG code is two-step majority-logic decodable. Because its minimum distance $d_{min} = 7$ and $J = 6$, it is completely orthogonalizable.

There is no simple formula for enumerating the number of parity-check digits for a general twofold EG code; however, for $\mu = m - 1$, the number of parity-check digits for the $(m - 1, s)$ th-order twofold EG code of length $2^{ms} - 1$ is [34]

$$n - k = \binom{m+1}{m}^s - \binom{m}{m-1}^s. \quad (8.41)$$

A list of twofold EG codes is given in Table 8.8. We see that the twofold EG codes are more efficient than their corresponding RM codes and are comparable to their corresponding BCH codes. For example, for error-correcting capability $t = 7$, there is a two-step majority-logic decodable (255, 191) twofold EG code; the corresponding RM code is a (255, 163) code that is five-step majority-logic decodable (using Chen's decoding algorithm [32], it may be decoded in two steps); the corresponding BCH code is a (255, 199) code. Twofold EG codes form a special subclass of multifold EG codes presented in [30] and [34].

TABLE 8.8: Twofold EG codes*.

<i>m</i>	<i>s</i>	μ	<i>n</i>	<i>k</i>	<i>J</i>	t_{ML}
3	2	1	63	24	14	7
2	3	1	63	45	6	3
4	2	1	255	45	62	31
4	2	2	255	171	14	7
2	4	1	255	191	14	7
3	3	1	511	184	62	31
3	3	2	511	475	6	3
5	2	1	1023	76	254	127
5	2	2	1023	438	62	31
5	2	3	1023	868	14	7
2	5	1	1023	813	30	15

*The (63, 24) and (63, 45) codes are BCH codes.

8.8 PROJECTIVE GEOMETRY AND PROJECTIVE GEOMETRY CODES

Like Euclidean geometry, a projective geometry may be constructed from the elements of a Galois field. Consider the Galois field $GF(2^{(m+1)s})$ that contains $GF(2^s)$ as a subfield. Let α be a primitive element in $GF(2^{(m+1)s})$. Then, the powers of α , $\alpha^0, \alpha^1, \dots, \alpha^{2^{(m+1)s}-2}$ form all the nonzero elements of $GF(2^{(m+1)s})$. Let

$$n = \frac{2^{(m+1)s} - 1}{2^s - 1} = 2^{ms} + 2^{(m-1)s} + \dots + 2^s + 1. \quad (8.42)$$

Then, the order of $\beta = \alpha^n$ is $2^s - 1$. The 2^s elements $0, 1, \beta, \beta^2, \dots, \beta^{2^s-2}$ form the Galois field $GF(2^s)$.

Consider the first n powers of α :

$$\Gamma = \{\alpha^0, \alpha^1, \alpha^2, \dots, \alpha^{n-1}\}.$$

No element α^i in Γ can be a product of an element in $GF(2^s)$ and another element α^j in Γ [i.e., $\alpha^i \neq \eta \cdot \alpha^j$ for $\eta \in GF(2^s)$]. Suppose that $\alpha^i = \eta\alpha^j$. Then, $\alpha^{i-j} = \eta$. Because $\eta^{2^s-1} = 1$, we obtain $\alpha^{(i-j)(2^s-1)} = 1$. This is impossible, since $(i-j)(2^s-1) < 2^{(m+1)s} - 1$, and the order of α is $2^{(m+1)s} - 1$. Therefore, we conclude that for α^i and α^j in Γ , $\alpha^i \neq \eta\alpha^j$ for any $\eta \in GF(2^s)$. Now, we partition the nonzero elements of $GF(2^{(m+1)s})$ into n disjoint subsets as follows:

$$\begin{aligned} &\{\alpha^0, \beta\alpha^0, \beta^2\alpha^0, \dots, \beta^{2^s-2}\alpha^0\}, \\ &\{\alpha^1, \beta\alpha^1, \beta^2\alpha^1, \dots, \beta^{2^s-2}\alpha^1\}, \\ &\{\alpha^2, \beta\alpha^2, \beta^2\alpha^2, \dots, \beta^{2^s-2}\alpha^2\}, \\ &\vdots \\ &\{\alpha^{n-1}, \beta\alpha^{n-1}, \beta^2\alpha^{n-1}, \dots, \beta^{2^s-2}\alpha^{n-1}\}, \end{aligned}$$

where $\beta = \alpha^n$, a primitive element in $GF(2^s)$. Each set consists of $2^s - 1$ elements, and each element is a multiple of the first element in the set. No element in one set can be a product of an element of $GF(2^s)$ and an element from a different set. Now, we represent each set by its first element as follows:

$$(\alpha^i) \triangleq \{\alpha^i, \beta\alpha^i, \dots, \beta^{2^s-2}\alpha^i\},$$

with $0 \leq i < n$. For any α^j in $GF(2^{(m+1)s})$, if $\alpha^j = \beta^l \cdot \alpha^i$ with $0 \leq i < n$, then α^j is represented by (α^i) . If each element in $GF(2^{(m+1)s})$ is represented as an $(m+1)$ -tuple over $GF(2^s)$, then (α^i) consists of $2^s - 1$ $(m+1)$ -tuples over $GF(2^s)$. The $(m+1)$ -tuple for α^i represents the $2^s - 1$ $(m+1)$ -tuples in (α^i) . All the $(m+1)$ -tuples representing the elements in (α^i) are multiples of the $(m+1)$ -tuple representing α^i . The $(m+1)$ -tuple over $GF(2^s)$ that represents (α^i) may be regarded as a point in a finite geometry over $GF(2^s)$. Then, the points

$$(\alpha^0), (\alpha^1), (\alpha^2), \dots, (\alpha^{n-1})$$

are said to form an m -dimensional projective geometry over $GF(2^s)$, denoted by $PG(m, 2^s)$ [15, 16]. In this geometry, the $2^s - 1$ elements in $\{\alpha^i, \beta\alpha^i, \dots, \beta^{2^s-2}\alpha^i\}$ are considered to be the same point in $PG(m, 2^s)$. This is a major difference between a projective geometry and a Euclidean geometry. A projective geometry does not have an origin.

Let (α^i) and (α^j) be any two *distinct* points in $PG(m, 2^s)$. Then, the *line* (1-flat) passing through (or connecting) (α^i) and (α^j) consists of points of the following form:

$$(\eta_1\alpha^i + \eta_2\alpha^j), \quad (8.43)$$

where η_1 and η_2 are from $GF(2^s)$ and are not both equal to zero. There are $(2^s)^2 - 1$ possible choices of η_1 and η_2 from $GF(2^s)$ (excluding $\eta_1 = \eta_2 = 0$); however, there are always $2^s - 1$ choices of η_1 and η_2 that result in the same point. For example,

$$\eta_1\alpha^i + \eta_2\alpha^j, \beta\eta_1\alpha^i + \beta\eta_2\alpha^j, \dots, \beta^{2^s-2}\eta_1\alpha^i + \beta^{2^s-2}\eta_2\alpha^j$$

represent the same point in $PG(m, 2^s)$. Therefore, a line in $PG(m, 2^s)$ consists of

$$\frac{(2^s)^2 - 1}{2^s - 1} = 2^s + 1$$

points. To generate the $2^s + 1$ distinct points on the line $\{(\eta_1\alpha^i + \eta_2\alpha^j)\}$, we simply choose η_1 and η_2 such that no choice (η_1, η_2) is a multiple of another choice (η'_1, η'_2) [i.e., $(\eta_1, \eta_2) \neq (\delta\eta'_1, \delta\eta'_2)$ for any $\delta \in GF(2^s)$].

EXAMPLE 8.24

Let $m = 2$ and $s = 2$. Consider the projective geometry $PG(2, 2^2)$. This geometry can be constructed from the field $GF(2^6)$, which contains $GF(2^2)$ as a subfield. Let

$$n = \frac{2^6 - 1}{2^2 - 1} = 2^{2 \cdot 2} + 2^2 + 1 = 21.$$

Let α be a primitive of $GF(2^6)$ (use Table 6.2). Let $\beta = \alpha^{21}$. Then $0, 1, \beta$, and β^2 form the field $GF(2^2)$. The geometry $PG(2, 2^2)$ consists of the following 21 points:

$$\begin{aligned} &(\alpha^0), (\alpha^1), (\alpha^2), (\alpha^3), (\alpha^4), (\alpha^5), (\alpha^6), \\ &(\alpha^7), (\alpha^8), (\alpha^9), (\alpha^{10}), (\alpha^{11}), (\alpha^{12}), (\alpha^{13}), \\ &(\alpha^{14}), (\alpha^{15}), (\alpha^{16}), (\alpha^{17}), (\alpha^{18}), (\alpha^{19}), (\alpha^{20}). \end{aligned}$$

Consider the line passing through the point (α) and (α^{20}) that consists of five points of the form $(\eta_1\alpha + \eta_2\alpha^{20})$, with η_1 and η_2 from $GF(2^2) = \{0, 1, \beta, \beta^2\}$. The five distinct points are

$$\begin{aligned} &(\alpha), \\ &(\alpha^{20}), \\ &(\alpha + \alpha^{20}) = (\alpha^{57}) = (\beta^2\alpha^{15}) = (\alpha^{15}), \\ &(\alpha + \beta\alpha^{20}) = (\alpha + \alpha^{41}) = (\alpha^{56}) = (\beta^2\alpha^{14}) = (\alpha^{14}), \\ &(\alpha + \beta^2\alpha^{20}) = (\alpha + \alpha^{62}) = (\alpha^{11}). \end{aligned}$$

Thus, $\{(\alpha), (\alpha^{11}), (\alpha^{14}), (\alpha^{15}), (\alpha^{20})\}$ is the line in $PG(2, 2^s)$ that passes through the points (α) and (α^{20}) .

Let (α^l) be a point not on the line $\{(\eta_1\alpha^i + \eta_2\alpha^j)\}$. Then, the line $\{(\eta_1\alpha^i + \eta_2\alpha^j)\}$ and the line $\{(\eta_1\alpha^l + \eta_2\alpha^j)\}$ have (α^j) as a common point (the only common point). We say that they intersect at (α^j) . The number of lines in $PG(m, 2^s)$ that intersect at a given point is

$$\frac{2^{ms} - 1}{2^s - 1} = 1 + 2^s + \cdots + 2^{(m-1)s}. \quad (8.44)$$

Let $(\alpha^{l_1}), (\alpha^{l_2}), \dots, (\alpha^{l_{\mu+1}})$ be $\mu + 1$ linearly independent points (i.e., $\eta_1\alpha^{l_1} + \eta_2\alpha^{l_2} + \cdots + \eta_{\mu+1}\alpha^{l_{\mu+1}} = 0$ if and only if $\eta_1 = \eta_2 = \cdots = \eta_{\mu+1} = 0$). Then, a μ -flat in $PG(m, 2^s)$ consists of points of the form

$$(\eta_1\alpha^{l_1} + \eta_2\alpha^{l_2} + \cdots + \eta_{\mu+1}\alpha^{l_{\mu+1}}), \quad (8.45)$$

where $\eta_i \in GF(2^s)$, and not all $\eta_1, \eta_2, \dots, \eta_{\mu+1}$ are zero. There are $2^{(\mu+1)s} - 1$ choices for $\eta_1, \eta_2, \dots, \eta_{\mu+1}$ ($\eta_1 = \eta_2 = \cdots = \eta_{\mu+1} = 0$ is not allowed). Because there are always $2^s - 1$ choices of η_1 to $\eta_{\mu+1}$ resulting in the same point in $PG(m, 2^s)$, there are

$$\frac{2^{(\mu+1)s} - 1}{2^s - 1} = 1 + 2^2 + \cdots + 2^{\mu s} \quad (8.46)$$

points in a μ -flat in $PG(m, 2^s)$. Let $\alpha^{l'_{\mu+1}}$ be a point not in the μ -flat:

$$\{(\eta_1\alpha^{l_1} + \eta_2\alpha^{l_2} + \cdots + \eta_{\mu+1}\alpha^{l_{\mu+1}})\}.$$

Then, the μ -flat $\{(\eta_1\alpha^{l_1} + \eta_2\alpha^{l_2} + \cdots + \eta_{\mu}\alpha^{l_{\mu}} + \eta_{\mu+1}\alpha^{l'_{\mu+1}})\}$ and the μ -flat $\{(\eta_1\alpha^{l_1} + \eta_2\alpha^{l_2} + \cdots + \eta_{\mu}\alpha^{l_{\mu}} + \eta_{\mu+1}\alpha^{l'_{\mu+1}})\}$ intersect on the $(\mu - 1)$ -flat $\{(\eta_1\alpha^{l_1} + \eta_2\alpha^{l_2} + \cdots +$

$\eta_\mu \alpha^{\mu})\}. The number of μ -flats in $\text{PG}(m, 2^s)$ that intersect on a given $(\mu - 1)$ -flat in $\text{PG}(m, 2^s)$ is$

$$\frac{2^{(m-\mu+1)s} - 1}{2^s - 1} = 1 + 2^s + \cdots + 2^{(m-\mu)s}. \quad (8.47)$$

Every point outside a $(\mu - 1)$ -flat F is in one and only one of the μ -flats intersecting on F .

Let $\mathbf{v} = (v_0, v_1, \dots, v_{n-1})$ be an n -tuple over $GF(2)$, where

$$n = \frac{2^{(m+1)s} - 1}{2^s - 1} = 1 + 2^s + \cdots + 2^{ms}.$$

Let α be a primitive element in $GF(2^{(m+1)s})$. We may number the components of \mathbf{v} with the first n powers of α as follows: v_i is numbered α^i for $0 \leq i < n$. As usual, α^i is called the location number of v_i . Let F be a μ -flat in $\text{PG}(m, 2^s)$. The incidence vector for F is an n -tuple over $GF(2)$,

$$\mathbf{v}_F = (v_0, v_1, \dots, v_{n-1}),$$

whose i th component

$$v_i = \begin{cases} 1 & \text{if } (\alpha^i) \text{ is a point in } F, \\ 0 & \text{otherwise.} \end{cases}$$

DEFINITION 8.5 A (μ, s) th-order binary projective geometry (PG) code of length $n = (2^{(m+1)s} - 1)/(2^s - 1)$ is defined as the largest cyclic code whose null space contains the incidence vectors of all the μ -flats in $\text{PG}(m, 2^s)$.

Let h be a nonnegative integer less than $2^{(m+1)s} - 1$ and $h^{(l)}$ be the remainder resulting from dividing $2^l h$ by $2^{(m+1)s} - 1$. Clearly, $h^{(0)} = h$. The 2^s -weight of h , $W_{2^s}(h)$, is defined by (8.28). The following theorem characterizes the roots of the generator polynomial of a (μ, s) th-order PG code (the proof is omitted). The proof of this theorem can be found in [27, 35], and [36].

THEOREM 8.5 Let α be a primitive element $GF(\alpha^{(m+1)s})$. Let h be a nonnegative integer less than $2^{(m+1)s} - 1$. Then, the generator polynomial $\mathbf{g}(X)$ of a (μ, s) th-order PG code of length $n = (2^{(m+1)s} - 1)/(2^s - 1)$ has α^h as a root if and only if h is divisible by $2^s - 1$, and

$$0 \leq \max_{0 \leq l < s} W_{2^s}(h^{(l)}) = j(2^s - 1), \quad (8.48)$$

with $0 \leq j \leq m - \mu$.

EXAMPLE 8.25

Let $m = 2$, $s = 2$, and $\mu = 1$. Consider the $(1, 2)$ th-order PG code of length

$$n = \frac{2^{(2+1)\cdot 2} - 1}{2^2 - 1} = 21.$$

Let α be a primitive element of $GF(2^6)$. Let h be a nonnegative integer less than 63. It follows from Theorem 8.5 that the generator polynomial $\mathbf{g}(X)$ of the $(1, 2)$ th-order PG code of length 21 has α^h as a root if and only if h is divisible by 3, and

$$0 \leq \max_{0 \leq l < 2} W_{2^2}(h^{(l)}) = 3j,$$

with $0 \leq j \leq 1$. The integers that are divisible by 3 and satisfy the preceding condition are 0, 3, 6, 9, 12, 18, 24, 33, 36, and 48. Thus, $\mathbf{g}(X)$ has $\alpha^0 = 1, \alpha^3, \alpha^6, \alpha^9, \alpha^{12}, \alpha^{18}, \alpha^{24}, \alpha^{33}, \alpha^{36}$, and α^{48} as roots. From Table 6.3 we find that (1) $\alpha^3, \alpha^6, \alpha^{12}, \alpha^{24}, \alpha^{33}$, and α^{48} have the same minimal polynomial, $\phi_3(X) = 1 + X + X^2 + X^4 + X^6$; and (2) α^9, α^{18} , and α^{36} have $\phi_9(X) = 1 + X^2 + X^3$ as their minimal polynomial. Thus,

$$\begin{aligned}\mathbf{g}(X) &= (1 + X)\phi_3(X)\phi_9(X) \\ &= 1 + X^2 + X^4 + X^6 + X^7 + X^{10}.\end{aligned}$$

Hence, the $(1, 2)$ th-order PG code of length 21 is a $(21, 11)$ cyclic code. It is interesting to note that this code is the $(21, 11)$ difference-set code considered in Example 8.9.

Decoding PG codes is similar to decoding EG codes. Consider the decoding of a (μ, s) th-order PG code of length $n = (2^{(m+1)s} - 1)/(2^s - 1)$. The null space of this code contains the incidence vectors of all the μ -flats of $PG(m, 2^s)$. Let $F^{(\mu-1)}$ be a $(\mu - 1)$ -flat in $PG(m, 2^s)$ that contains the point (α^{n-1}) . From (8.47) we see that there are

$$J = \frac{2^{(m-\mu+1)s} - 1}{2^s - 1}$$

μ -flats intersecting on the $(\mu - 1)$ -flat $F^{(\mu-1)}$. The incidence vectors of these J μ -flats are orthogonal on the digits at the locations corresponding to the points in $F^{(\mu-1)}$. Therefore, the parity-check sums formed from these J incidence vectors are orthogonal on the error digits at the locations corresponding to the points in $F^{(\mu-1)}$. Let $S(F^{(\mu-1)})$ denote the sum of error digits at the locations corresponding to the points in $F^{(\mu-1)}$. Then, we can correctly determine this error sum, $S(F^{(\mu-1)})$, from the J check-sums orthogonal on it provided that there are no more than

$$\left\lfloor \frac{J}{2} \right\rfloor = \left\lfloor \frac{2^{(m-\mu+1)s} - 1}{2(2^s - 1)} \right\rfloor$$

errors in the received vector. In this manner we can determine the error sums, $S(F^{(\mu-1)})$'s, corresponding to all the $(\mu - 1)$ -flats that contain the point (α^{n-1}) . We then use these error sums to determine the error sums, $S(F^{(\mu-2)})$'s, corresponding to all the $(\mu - 2)$ -flats that contain (α^{n-1}) . We continue this process until the error sums, $S(F^{(1)})$'s, corresponding to all the 1-flats that intersect on (α^{n-1}) are formed. These error sums, $S(F^{(1)})$'s, are orthogonal on the error digit e_{n-1} at the location α^{n-1} . Thus, we can determine the value of e_{n-1} . A total of μ steps of orthogonalization are required to decode e_{n-1} . Because the code is cyclic, we can decode other error

digits in the same manner. Thus, the code is μ -step decodable. At the r th step of orthogonalization with $1 \leq r \leq \mu$, the number of error sums, $S(F^{(\mu-r+1)})$'s, that are orthogonal in the error sum corresponding to a given $(\mu-r)$ -flat $F^{(\mu-r)}$ is

$$J_{\mu-r+1} = \frac{2^{(m-\mu+r)s} - 1}{2^s - 1} \geq J.$$

Therefore, at each step of orthogonalization, we can always correctly determine the error sums needed for the next step provided that there are no more than $\lfloor J/2 \rfloor$ errors in the received vector. Thus, the μ th-order PG code of length $n = (2^{(m+1)s} - 1)/(2^s - 1)$ is capable of correcting

$$t_{ML} = \left\lfloor \frac{J}{2} \right\rfloor = \left\lfloor \frac{2^{(m-\mu+1)s} - 1}{2(2^s - 1)} \right\rfloor \quad (8.49)$$

or fewer errors with majority-logic decoding. Its minimum distance is at least

$$2t_{ML} + 1 = 2^{(m-\mu)s} + \dots + 2^s + 2. \quad (8.50)$$

EXAMPLE 8.26

Consider the decoding of the $(1, 2)$ th-order $(21, 11)$ PG code with $m = 2$ and $s = 2$. The null space of this code contains the incidence vectors of all the 1-flats (lines) in $\text{PG}(2, 2^2)$. Let α be a primitive element in $GF(2^6)$. The geometry $\text{PG}(2, 2^2)$ consists of 21 points, (α^0) to (α^{20}) , as given in Example 8.24. Let $\beta = \alpha^{21}$. Then, $0, 1, \beta$, and β^2 form the field $GF(2^2)$.

There are $2^2 + 1 = 5$ lines passing through the point (α^{20}) , namely,

$$\begin{aligned} \{(\eta_1\alpha^0 + \eta_2\alpha^{20})\} &= \{(\alpha^0), (\alpha^5), (\alpha^7), (\alpha^{17}), (\alpha^{20})\}, \\ \{(\eta_1\alpha^1 + \eta_2\alpha^{20})\} &= \{(\alpha^1), (\alpha^{11}), (\alpha^{14}), (\alpha^{15}), (\alpha^{20})\}, \\ \{(\eta_1\alpha^2 + \eta_2\alpha^{20})\} &= \{(\alpha^2), (\alpha^3), (\alpha^8), (\alpha^{10}), (\alpha^{20})\}, \\ \{(\eta_1\alpha^4 + \eta_2\alpha^{20})\} &= \{(\alpha^4), (\alpha^6), (\alpha^{16}), (\alpha^{19}), (\alpha^{20})\}, \\ \{(\eta_1\alpha^9 + \eta_2\alpha^{20})\} &= \{(\alpha^9), (\alpha^{12}), (\alpha^{13}), (\alpha^{18}), (\alpha^{20})\}. \end{aligned}$$

The incidence vectors of these lines (in polynomial form) are

$$\begin{aligned} \mathbf{w}_1(X) &= 1 + X^5 + X^7 + X^{17} + X^{20}, \\ \mathbf{w}_2(X) &= X + X^{11} + X^{14} + X^{15} + X^{20}, \\ \mathbf{w}_3(X) &= X^2 + X^3 + X^8 + X^{10} + X^{20}, \\ \mathbf{w}_4(X) &= X^4 + X^6 + X^{16} + X^{19} + X^{20}, \\ \mathbf{w}_5(X) &= X^9 + X^{12} + X^{13} + X^{18} + X^{20}. \end{aligned}$$

These vectors are orthogonal on digit position 20. They are exactly the orthogonal vectors for the $(21, 11)$ difference-set code given in Examples 8.9 and 8.10.

TABLE 8.9: PG codes.

m	s	μ	n	k	J	t_{ML}
2	2	1	21	11	5	2
2	3	1	73	45	9	4
3	2	2	85	68	5	2
3	2	1	85	24	21	10
2	4	1	273	191	17	8
4	2	3	341	315	5	2
4	2	2	341	195	21	10
4	2	1	341	45	85	42
3	3	2	585	520	9	4
3	3	1	585	184	73	36
2	5	1	1057	813	33	16
5	2	4	1365	1328	5	2
5	2	3	1365	1063	21	10
5	2	2	1365	483	85	21
5	2	1	1365	76	341	170
2	6	1	4161	3431	65	32
6	2	5	5461	5411	5	2
6	2	4	5461	4900	21	10
6	2	3	5461	3185	85	42
6	2	2	5461	1064	341	170
6	2	1	5461	119	1365	682

For $\mu = 1$, we obtain a class of one-step majority-logic decodable PG codes. For $m = 2$, a $(1, s)$ th-order PG code becomes a difference-set code. Thus, the difference-set codes form a subclass of the class of $(1, s)$ th-order PG codes. For $s = 1$, a $(1, 1)$ th-order PG code becomes a maximum-length code.

There is no simple formula for enumerating the number of parity-check digits for a general (μ, s) th-order PG code. Rather complicated combinatorial expressions for the number of parity-check digits of PG codes can be found in [17] and [18]; however for $\mu = m - 1$, the number of parity-check digits for the $(m - 1, s)$ th-order of PG code of length $n = (2^{(m+1)s} - 1)/(2^s - 1)$ is

$$n - k = 1 + \binom{m+1}{m}^s. \quad (8.51)$$

This expression was obtained independently by Goethals Delsarte [35], Smith [19], and MacWilliams and Mann [20]. A list of PG codes is given in Table 8.9.

PG codes were first studied by Rudolph [3] and were later extended and generalized by many others [19, 24, 25, 27, 35, 36].

8.9 REMARKS

In this chapter we considered only finite geometries over Galois field $GF(2^s)$ and the construction of majority-logic decodable codes based on these geometries. Finite geometries over Galois field $GF(p^s)$, where p is a prime, can be constructed in

exactly the same way, simply by replacing 2 with p and $GF(2^s)$ with $GF(p^s)$. Construction of codes based on the flats and points in these finite geometries results in a much larger class of majority-logic decodable codes. Construction of finite geometries over $GF(p^s)$ and their application to the construction of low-density parity-check codes will be discussed in Chapter 17.

PROBLEMS

- 8.1** Consider the $(31, 5)$ maximum-length code whose parity-check polynomial is $\mathbf{p}(X) = 1 + X^2 + X^5$. Find all the polynomials orthogonal on the digit position X^{30} . Devise both type-I and type-II majority-logic decoders for this code.
- 8.2** $P = \{0, 2, 3\}$ is a perfect simple difference set. Construct a difference-set code based on this set.
- What is the length n of this code?
 - Determine its generator polynomial.
 - Find all the polynomials orthogonal on the highest-order digit position X^{n-1} .
 - Construct a type-I majority-logic decoder for this code.
- 8.3** Example 8.1 shows that the $(15, 7)$ BCH code is one-step majority-logic decodable and is capable of correcting any combination of two or fewer errors. Show that the code is also capable of correcting some error patterns of three errors and some error patterns of four errors. List some of these error patterns.
- 8.4** Consider an $(11, 6)$ linear code whose parity-check matrix is

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

(This code is not cyclic.)

- Show that the minimum distance of this code is exactly 4.
 - Let $\mathbf{e} = (e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10})$ be an error vector. Find the syndrome bits in terms of error digits.
 - Construct all possible parity-check sums orthogonal on each message error digit e_i for $i = 5, 6, 7, 8, 9, 10$.
 - Is this code completely orthogonalizable in one step?
- 8.5** Let $m = 6$. Express the integer 43 in radix-2 form. Find all the nonzero proper descendants of 43.
- 8.6** Let α be a primitive element of $GF(2^4)$ given by Table 2.8. Apply the affine permutation $Z = \alpha^3Y + \alpha^{11}$ to the following vector of 16 components:

Location Numbers

$$\begin{array}{cccccccccccccccc} \alpha^\infty & \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\ \hline \mathbf{u} = (1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1) \end{array}$$

What is the resultant vector?

- 8.7** Let $m = 6$. Then, $2^6 - 1$ can be factored as follows: $2^6 - 1 = 7 \times 9$. Let $J = 9$ and $L = 7$. Find the generator polynomial of the type-I DTI code of length 63 and $J = 9$ (use Table 6.2). Find all the polynomials (or vectors) orthogonal on the digit position X^{62} (or α^{62}).

- 8.8** Find the generator polynomial of the type-I DTI code of length 63 and $J = 7$. Find all the polynomials orthogonal on the digit position X^{62} .
- 8.9** Show that the all-one vector is not a code vector in a maximum-length code.
- 8.10** Let $\mathbf{v}(X) = v_0 + v_1X + \cdots + v_{2^m-2}X^{2^m-2}$ be a nonzero code polynomial in the $(2^m - 1, m)$ maximum-length code whose parity-check polynomial is $\mathbf{p}(X)$. Show that the other $2^m - 2$ nonzero code polynomials are cyclic shifts of $\mathbf{v}(X)$. (*Hint:* Let $\mathbf{v}^{(i)}(X)$ and $\mathbf{v}^{(j)}(X)$ be the i th and j th cyclic shifts of $\mathbf{v}(X)$, respectively, with $0 \leq i < j < 2^m - 2$. Show that $\mathbf{v}^{(i)}(X) \neq \mathbf{v}^{(j)}(X)$.)
- 8.11** Arrange the 2^m code vectors of a maximum-length code as rows of a $2^m \times (2^m - 1)$ array.
- Show that each column of this array has 2^{m-1} ones and 2^{m-1} zeros.
 - Show that the weight of each nonzero code vector is exactly 2^{m-1} .
- 8.12** Example 8.12 shows that the $(15, 5)$ BCH code is two-step majority-logic decodable and is capable of correcting any combination of three or fewer errors. Devise a type-I majority-logic decoder for this code.
- 8.13** Show that the extended cyclic Hamming code is invariant under the affine permutations.
- 8.14** Show that the extended primitive BCH code is invariant under the affine permutations.
- 8.15** Let $P = \{l_0, l_1, l_2, \dots, l_{2^s}\}$ be a perfect simple difference set of order 2^s such that

$$0 \leq l_0 < l_1 < l_2 < \cdots < l_{2^s} \leq 2^s(2^s + 1).$$

Construct a vector of $n = 2^{2s} + 2^s + 1$ components,

$$\mathbf{v} = (v_0, v_1, \dots, v_{n-1}),$$

whose nonzero components are $v_{l_0}, v_{l_1}, \dots, v_{l_{2^s}}$; that is,

$$v_{l_0} = v_{l_1} = \cdots = v_{l_{2^s}} = 1.$$

Consider the following $n \times 2n$ matrix:

$$\mathbf{G} = [\mathbf{Q} \ \mathbf{I}_n],$$

where (1) \mathbf{I}_n is an $n \times n$ identity matrix, and (2) \mathbf{Q} is an $n \times n$ matrix whose n rows are \mathbf{v} and $n - 1$ cyclic shifts of \mathbf{v} . The code generated by \mathbf{G} is a $(2n, n)$ linear (not cyclic) code whose parity-check matrix is

$$\mathbf{H} = [\mathbf{I}_n \ \mathbf{Q}^T].$$

- Show that $J = 2^s + 1$ parity-check sums orthogonal on any message error digit can be formed.
 - Show that the minimum distance of this code is $d = J + 1 = 2^s + 2$. (This code is a half-rate *quasi-cyclic code* [20].)
- 8.16** Prove that if J parity-check sums orthogonal on any digit position can be formed for a linear code (cyclic or noncyclic), the minimum distance of the code is at least $J + 1$.
- 8.17** Consider the Galois field $GF(2^4)$ given by Table 2.8. Let $\beta = \alpha^5$. Then, $\{0, 1, \beta, \beta^2\}$ form the subfield $GF(2^2)$ of $GF(2^4)$. Regard $GF(2^4)$ as the two-dimensional Euclidean geometry over $GF(2^2)$, $EG(2, 2^2)$. Find all the 1-flats that pass through the point α^7 .

- 8.18** Consider the Galois field $GF(2^6)$ given by Table 6.2. Let $\beta = \alpha^{21}$. Then, $\{0, 1, \beta, \beta^2\}$ form the subfield $GF(2^2)$ of $GF(2^6)$. Regard $GF(2^6)$ as the three-dimensional Euclidean geometry $EG(3, 2^2)$.
- Find all the 1-flats that pass through the point α^{63} .
 - Find all the 2-flats that intersect on the 1-flat, $\{\alpha^{63} + \eta\alpha\}$, where $\eta \in GF(2^2)$.
- 8.19** Regard $GF(2^6)$ as the two-dimensional Euclidean geometry $EG(2, 2^3)$. Let $\beta = \alpha^9$. Then, $\{0, 1, \beta, \beta^2, \beta^3, \beta^4, \beta^5, \beta^6\}$ form the subfield $GF(2^3)$ of $GF(2^6)$. Determine all the 1-flats that pass through the point α^{21} .
- 8.20** Let $m = 2$ and $s = 3$.
- Determine the 2^3 -weight of 47.
 - Determine $\max_{0 \leq l \leq 3} W_{2^3}(47^{(l)})$.
 - Determine all the positive integers h less than 63 such that
- $$0 < \max_{0 \leq l \leq 3} W_{2^3}(h^{(l)}) \leq 2^3 - 1.$$
- 8.21** Find the generator polynomial of the first-order cyclic RM code of length $2^5 - 1$. Describe how to decode this code.
- 8.22** Find the generator polynomial of the third-order cyclic RM code of length $2^6 - 1$. Describe how to decode this code.
- 8.23** Let $m = 2$ and $s = 3$. Find the generator polynomial of the $(0, 3)$ th-order EG code of length $2^{2 \times 3} - 1$. This code is one-step majority-logic decodable. Find all the polynomials orthogonal on the digit location α^{62} where α is a primitive element in $GF(2^{2 \times 3})$. Design a type-I majority-logic decoder for this code.
- 8.24** Let $m = 3$ and $s = 2$. Find the generator polynomial of the $(1, 2)$ th-order twofold EG code of length $2^{3 \times 2} - 1$. Describe how to decode this code.
- 8.25** Prove that the $(m - 2)$ th-order cyclic RM code of length $2^m - 1$ is a Hamming code. (*Hint:* Show that its generator polynomial is a primitive polynomial of degree m .)
- 8.26** Prove that the even-weight codewords of the first-order cyclic RM code of length $2^m - 1$ form the maximum-length code of length $2^m - 1$.
- 8.27** Let $0 < \mu < m - 1$. Prove that the even-weight codewords of the $(m - \mu - 1)$ th-order cyclic RM code of length $2^m - 1$ form the dual of the μ th-order RM code of length $2^m - 1$. (*Hint:* Let $\mathbf{g}(X)$ be the generator polynomial of the $(m - \mu - 1)$ th-order cyclic RM code C . Show that the set of even-weight codewords of C is a cyclic code generated by $(X + 1)\mathbf{g}(X)$. Show that the dual of the μ th-order cyclic RM code is also generated by $(X + 1)\mathbf{g}(X)$.)
- 8.28** The μ th-order cyclic RM code of length $2^m - 1$ has a minimum distance of $d_{\min} = 2^{m-\mu} - 1$. Prove that this RM code is a subcode of the primitive BCH code of length $2^m - 1$ and designed distance $2^{m-\mu} - 1$. (*Hint:* Let $\mathbf{g}(X)_{RM}$ be the generator polynomial of the RM code and let $\mathbf{g}(X)_{BCH}$ be the generator polynomial of the BCH code. Prove that $\mathbf{g}(X)_{BCH}$ is a factor of $\mathbf{g}(X)_{RM}$.)
- 8.29** Show that extended RM codes are invariant under the affine permutations.
- 8.30** Let $m = 3, s = 2$ and $\mu = 2$. Find the generator polynomial of the $(2, 2)$ th-order PG code constructed based on the projective geometry $PG(3, 2^2)$. This code is two-step majority-logic decodable. Find all the orthogonal polynomials at each step of orthogonalization.
- 8.31** Let \mathcal{L} be a line in the two-dimensional Euclidean geometry $EG(2, 2^s)$ that does not pass through the origin. Let $\mathbf{v}_{\mathcal{L}}$ be the incidence vector of \mathcal{L} . For $0 < i \leq 2^{2s} - 2$, let $\mathbf{v}_{\mathcal{L}}^{(i)}$ be the i th cyclic shift of $\mathbf{v}_{\mathcal{L}}$. Prove that

$$\mathbf{v}_{\mathcal{L}}^{(i)} \neq \mathbf{v}_{\mathcal{L}}.$$

- 8.32** Let \mathcal{L} be a line in the two-dimensional projective geometry $\text{PG}(2, 2^s)$. Let $\mathbf{v}_{\mathcal{L}}$ be the incidence vector of \mathcal{L} . For $0 < i \leq 2^{2s} + 2^s$, let $\mathbf{v}_{\mathcal{L}}^{(i)}$ be the i th cyclic shift of $\mathbf{v}_{\mathcal{L}}$. Prove that

$$\mathbf{v}_{\mathcal{L}}^{(i)} \neq \mathbf{v}_{\mathcal{L}}.$$

- 8.33** Prove that a cyclic shift of the incidence vector of a μ -flat in $\text{EG}(m, 2^s)$ not passing through the origin is the incidence vector of another μ -flat in $\text{EG}(m, 2^s)$ not passing through the origin.
- 8.34** Consider the cyclic product code whose component codes are the $(3, 2)$ cyclic code generated by $\mathbf{g}_1(X) = 1 + X$ and the $(7, 4)$ Hamming code generated by $\mathbf{g}_2(X) = 1 + X + X^3$. The component code C_1 is completely orthogonalizable in one step, and the component code C_2 is completely orthogonalizable in two steps. Show that the product code is completely orthogonalizable in two steps. (In general, if one component code is completely orthogonalizable in one step, and the other component code is completely orthogonalizable in L steps, the product code is completely orthogonalizable in L steps [37].)

BIBLIOGRAPHY

1. I. S. Reed, "A Class of Multiple-Error-Correcting Codes and the Decoding Scheme," *IRE Trans.*, IT-4: 38–49, September 1954.
2. J. L. Massey, *Threshold Decoding*, MIT Press, Cambridge, 1963.
3. L. D. Rudolph, "A Class of Majority Logic Decodable Codes," *IEEE Trans. Inform. Theory*, IT-13: 305–7, April 1967.
4. T. Kasami, L. Lin, and W. W. Peterson, "Some Results on Cyclic Codes Which Are Invariant under the Affine Group and Their Applications," *Inform. Control*, 2(5 and 6): 475–96, November 1968.
5. W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2d ed. MIT Press, Cambridge, 1972.
6. S. Lin and G. Markowsky, "On a Class of One-Step Majority-Logic Decodable Cyclic Codes," *IBM J. Res. Dev.*, January 1980.
7. R. B. Yale, "Error-Correcting Codes and Linear Recurring Sequences," *Lincoln Laboratory Report*, pp. 33–77, Lincoln Labs., MIT, Cambridge, 1958.
8. N. Zierler, "On a Variation of the First-Order Reed-Muller Codes," *Lincoln Laboratory Report*, pp. 38–80, Lincoln Labs., MIT, Cambridge, 1958.
9. J. Singer, "A Theorem in Finite Projective Geometry and Some Applications to Number Theory," *AMS Trans.*, 43: 377–85, 1938.
10. T. A. Evans and H. B. Mann, "On Simple Difference Sets," *Sankhya*, 11: 464–81, 1955.
11. L. D. Rudolph, "Geometric Configuration and Majority Logic Decodable Codes," *M.E.E. thesis*, University of Oklahoma, Norman, 1964.

12. E. J. Weldon, Jr., "Difference-Set Cyclic Codes," *Bell Syst. Tech. J.*, 45: 1045–55, September 1966.
13. F. L. Graham and J. MacWilliams, "On the Number of Parity Checks in Difference-Set Cyclic Codes," *Bell Syst. Tech. J.*, 45: 1046–70, September 1966.
14. R. D. Carmichael, *Introduction to the Theory of Groups of Finite Order*, Dover, New York, 1956.
15. H. B. Mann, *Analysis and Design of Experiments*, Dover, New York, 1949.
16. A. P. Street and D. J. Street, *Combinatorics of Experimental Design*, Oxford Science Publications, Inc., Clarendon Press, Oxford, 1987.
17. N. Hamada, "On the p -rank of the Incidence Matrix of a Balanced or Partially Balanced Incomplete Block Design and Its Applications to Error-Correcting Codes," *Hiroshima Math. J.*, 3: 153–226, 1973.
18. S. Lin, "On the Number of Information Symbols in Polynomial Codes," *IEEE Trans. Inform. Theory*, IT-18: 785–94, November 1972.
19. K. J. C. Smith, "Majority Decodable Codes Derived from Finite Geometries," *Institute of Statistics Mimeo Series*, No. 561, University of North Carolina, Chapel Hill, 1967.
20. F. J. MacWilliams and H. B. Mann, "On the p -rank of the Design Matrix of a Different Set," *Inform. Control*, 12: 474–88, 1968.
21. T. Kasami, S. Lin, and W. W. Peterson, "Linear Codes Which Are Invariant under the Affine Group and Some Results on Minimum Weights in BCH Codes," *Electron. Commun. Jap.*, 50(9): 100–106, September 1967.
22. T. Kasami, S. Lin, and W. W. Peterson, "New Generalizations of the Reed–Muller Codes, Pt. I: Primitive Codes," *IEEE Trans. Inform. Theory*, IT-14: 189–99, March 1968.
23. V. D. Kolesnik and E. T. Mironchikov, "Cyclic Reed–Muller Codes and Their Decoding," *Probl. Inform. Transm.*, no. 4: 15–19, 1968.
24. C. R. P. Hartmann, J. B. Ducey, and L. D. Rudolph, "On the Structure of Generalized Finite Geometry Codes," *IEEE Trans. Inform. Theory*, IT-20(2): 240–52, March 1974.
25. D. K. Chow, "A Geometric Approach to Coding Theory with Applications to Information Retrieval," *CSL Report No. R-368*, University of Illinois, Urbana, 1967.
26. E. J. Weldon, Jr., "Euclidean Geometry Cyclic Codes," *Proc. Symp. Combinatorial Math.*, University of North Carolina, Chapel Hill, April 1967.
27. T. Kasami, S. Lin, and W. W. Peterson, "Polynomial Codes," *IEEE Trans. Inform. Theory*, 14: 807–14, 1968.

28. P. Delsarte, "A Geometric Approach to a Class of Cyclic Codes," *J. Combinatorial Theory*, 6: 340–58, 1969.
29. P. Delsarte, J. M. Goethals, and J. MacWilliams, "On GRM and Related Codes," *Inform. Control*, 16: 403–42, July 1970.
30. S. Lin and K. P. Yiu, "An Improvement to Multifold Euclidean Geometry Codes," *Inform. Control*, 28(3): 221–265 July 1975.
31. E. J. Weldon, Jr., "Some Results on Majority-Logic Decoding," Chap. 8 in *Error-Correcting Codes*, H. Mann, ed., John Wiley, New York, 1968.
32. C. L. Chen, "On Majority-Logic Decoding of Finite Geometry Codes," *IEEE Trans. Inform. Theory*, IT-17(3): 332–36, May 1971.
33. T. Kasami and S. Lin, "On Majority-Logic Decoding for Duals of Primitive Polynomial Codes," *IEEE Trans. Inform. Theory*, IT-17(3): 322–31, May 1971.
34. S. Lin, "Multifold Euclidean Geometry Codes," *IEEE Trans. Inform. Theory*, IT-19(4): 537–48, July 1973.
35. J. M. Goethals and P. Delsarte, "On a Class of Majority-Logic Decodable Codes," *IEEE Trans. Inform. Theory*, IT-14: 182–89, March 1968.
36. E. J. Weldon, Jr., "New Generations of the Reed–Muller Codes, Part II: Nonprimitive Codes," *IEEE Trans. Inform. Theory*, IT-14: 199–205, March 1968.
37. S. Lin and E. J. Weldon, Jr. "Further Results on Cyclic Product Codes," *IEEE Trans. Inform. Theory*, IT-16: 452–59, July 1970.