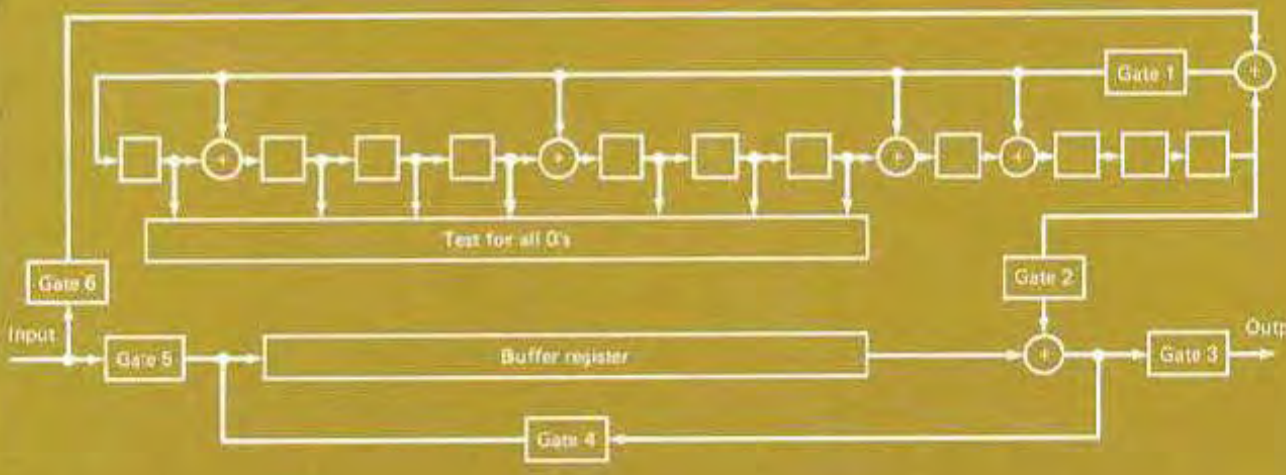
SHU LIN / DANIEL J. COSTELLO, Jr.

Error Control Coding:

Fundamentals and Applications



PRENTICE-HALL SERIES IN COMPUTER APPLICATIONS IN ELECTRICAL ENGINEERING
Franklin F. Kuo, *Editor*

IN/COSTELLO
Error Control Coding:
Fundamentals and Applications

QA
268
.L55
C.1

ERROR CONTROL CODING

Fundamentals and Applications

PRENTICE-HALL COMPUTER APPLICATIONS IN ELECTRICAL ENGINEERING SERIES

FRANKLIN F. KUO, editor

ABRAMSON and KUO, Computer-Communication Networks
BOWERS and SEDORE, Sceptre: A Computer Program for Circuit and Systems Anal;
CADZOW, Discrete Time Systems: An Introduction with Interdisciplinary Applications
CADZOW and MARTENS, Discrete-Time and Computer Control Systems
DAVIS, Computer Data Displays
FRIEDMAN and MENON, Fault Detection in Digital Circuits
HUELSMAN, Basic Circuit Theory
JENSEN and LIEBERMAN, IBM Circuit Analysis Program: Techniques and Applications
JENSEN and WATKINS, Network Analysis: Theory and Computer Methods
KLINE, Digital Computer Design
KOCHENBURGER, Computer Simulation of Dynamic Systems
KUO, (ed.) Protocols and Techniques for Data Communication Networks
KUO and MAGNUSON, Computer Oriented Circuit Design
LIN, An Introduction to Error-Correcting Codes
LIN and COSTELLO, Error Control Coding: Fundamentals and Applications
NAGLE, CARROLL, and IRWIN, An Introduction to Computer Logic
RHYNE, Fundamentals of Digitals Systems Design
SIFFERLEN and VARTANIAN, Digital Electronics with Engineering Applications
STAUDHAMMER, Circuit Analysis by Digital Computer
STOUTEMYER, PL/1 Programming for Engineering and Science

ERROR CONTROL CODING

Fundamentals and Applications

SHU LIN
University of Hawaii
Texas A&M University

DANIEL J. COSTELLO, JR.
Illinois Institute of Technology

Prentice-Hall, Inc. Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

LIN, SHU.
Error control coding.

(Prentice-Hall computer applications in electrical engineering series)
Includes bibliographical references and index.
1. Error-correcting codes (Information theory)
I. Costello, Daniel J. . II. Title.
III. Series.
QA268.L55 001.53'9 82-5255
ISBN 0-13-283796-X AACR2



Editorial/production supervision and interior design by Anne Simpson
Cover design by Marvin Warshaw
Manufacturing buyer: Joyce Levatino

© 1983 by Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632

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.

Printed in the United States of America

10 9 8 7

ISBN 0-13-283796-X

PRENTICE-HALL INTERNATIONAL, INC., *London*
PRENTICE-HALL OF AUSTRALIA PTY. LIMITED, *Sydney*
EDITORA PRENTICE-HALL DO BRAZIL, LTDA, *Rio de Janeiro*
PRENTICE-HALL CANADA INC., *Toronto*
PRENTICE-HALL OF INDIA PRIVATE LIMITED, *New Delhi*
PRENTICE-HALL OF JAPAN, INC., *Tokyo*
PRENTICE-HALL OF SOUTHEAST ASIA PTE. LTD., *Singapore*
WHITEHALL BOOKS LIMITED, *Wellington, New Zealand*

With Love and Affection for
Ivy,
Julian, Patrick, and Michelle Lin
and
Lucretia,
Kevin, Nick, Daniel, and Anthony Costello

Contents

PREFACE xiii

CHAPTER 1 CODING FOR RELIABLE DIGITAL TRANSMISSION AND STORAGE 1

1.1 Introduction 1
1.2 Types of Codes 3
1.3 Modulation and Demodulation 5
1.4 Maximum Likelihood Decoding 8
1.5 Types of Errors 11
1.6 Error Control Strategies 12
References 14

CHAPTER 2 INTRODUCTION TO ALGEBRA 15

2.1 Groups 15
2.2 Fields 19
2.3 Binary Field Arithmetic 24
2.4 Construction of Galois Field GF(2^m) 29
2.5 Basic Properties of Galois Field GF(2^m) 34
2.6 Computations Using Galois Field GF(2^m) Arithmetic 39
2.7 Vector Spaces 40
2.8 Matrices 46
Problems 48
References 50

Ex.1020 / Page 8 of 305
Sandisk Technologies, Inc. et al

CHAPTER 3 LINEAR BLOCK CODES 51

3.1 Introduction to Linear Block Codes 51
3.2 Syndrome and Error Detection 58
3.3 Minimum Distance of a Block Code 63
3.4 Error-Detecting and Error-Correcting Capabilities of a Block Code 65
3.5 Standard Array and Syndrome Decoding 68
3.6 Probability of an Undetected Error for Linear Codes over a BSC 76
3.7 Hamming Codes 79
Problems 82
References 84

CHAPTER 4 CYCLIC CODES 85

4.1 Description of Cyclic Codes 85
4.2 Generator and Parity-Check Matrices of Cyclic Codes 92
4.3 Encoding of Cyclic Codes 95
4.4 Syndrome Computation and Error Detection 98
4.5 Decoding of Cyclic Codes 103
4.6 Cyclic Hamming Codes 111
4.7 Shortened Cyclic Codes 116
Problems 121
References 123

CHAPTER 5 ERROR-TRAPPING DECODING FOR CYCLIC CODES 125

5.1 Error-Trapping Decoding 125
5.2 Improved Error-Trapping Decoding 131
5.3 The Golay Code 134
Problems 139
References 139

CHAPTER 6 BCH CODES 141

6.1 Description of the Codes 142
6.2 Decoding of the BCH Codes 151
6.3 Implementation of Galois Field Arithmetic 161
6.4 Implementation of Error Correction 167
6.5 Nonbinary BCH Codes and Reed–Solomon Codes 170
6.6 Weight Distribution and Error Detection of Binary BCH Codes 177
Problems 180
References 182

Ex.1020 / Page 9 of 305
Sandisk Technologies, Inc. et al

CHAPTER 7 MAJORITY-LOGIC DECODING FOR CYCLIC CODES 184

7.1 One-Step Majority-Logic Decoding 184
7.2 Class of One-Step Majority-Logic Decodable Codes 194
7.3 Other One-Step Majority-Logic Decodable Codes 201
7.4 Multiple-Step Majority-Logic Decoding 209
Problems 219
References 221

CHAPTER 8 FINITE GEOMETRY CODES 223

8.1 Euclidean Geometry 223
8.2 Majority-Logic Decodable Cyclic Codes Based on Euclidean Geometry 227
8.3 Projective Geometry and Projective Geometry Codes 240
8.4 Modifications of Majority-Logic Decoding 245
Problems 253
References 255

CHAPTER 9 BURST-ERROR-CORRECTING CODES 257

9.1 Introduction 257
9.2 Decoding of Single-Burst-Error-Correcting Cyclic Codes 259
9.3 Single-Burst-Error-Correcting Codes 261
9.4 Interleaved Codes 271
9.5 Phased-Burst-Error-Correcting Codes 272
9.6 Burst-and-Random-Error-Correcting Codes 274
9.7 Modified Fire Codes for Simultaneous Correction of Burst and Random Errors 280
Problems 282
References 284

CHAPTER 10 CONVOLUTIONAL CODES 287

10.1 Encoding of Convolutional Codes 288
10.2 Structural Properties of Convolutional Codes 295
10.3 Distance Properties of Convolutional Codes 308
Problems 312
References 313

CHAPTER 11 MAXIMUM LIKELIHOOD DECODING OF CONVOLUTIONAL CODES 315

11.1 The Viterbi Algorithm 315
11.2 Performance Bounds for Convolutional Codes 322

Ex.1020 / Page 10 of 305
Sandisk Technologies, Inc. et al

11.3 Construction of Good Convolutional Codes 329
11.4 Implementation of the Viterbi Algorithm 337
11.5 Modifications of the Viterbi Algorithm 345
Problems 346
References 348

CHAPTER 12 SEQUENTIAL DECODING OF CONVOLUTIONAL CODES 350

12.1 The Stack Algorithm 351
12.2 The Fano Algorithm 360
12.3 Performance Characteristics of Sequential Decoding 364
12.4 Code Construction for Sequential Decoding 374
12.5 Other Approaches to Sequential Decoding 380
Problems 384
References 386

CHAPTER 13 MAJORITY-LOGIC DECODING OF CONVOLUTIONAL CODES 388

13.1 Feedback Decoding 389
13.2 Error Propagation and Definite Decoding 406
13.3 Distance Properties and Code Performance 408
13.4 Code Construction for Majority-Logic Decoding 414
13.5 Comparison with Probabilistic Decoding 424
Problems 426
References 428

CHAPTER 14 BURST-ERROR-CORRECTING CONVOLUTIONAL CODES 429

14.1 Bounds on Burst-Error-Correcting Capability 430
14.2 Burst-Error-Correcting Convolutional Codes 430
14.3 Interleaved Convolutional Codes 441
14.4 Burst-and-Random-Error-Correcting Convolutional Codes 442
Problems 455
References 456

CHAPTER 15 AUTOMATIC-REPEAT-REQUEST STRATEGIES 458

15.1 Basic ARQ Schemes 459
15.2 Selective-Repeat ARQ System with Finite Receiver Buffer 465
15.3 ARQ Schemes with Mixed Modes of Retransmission 474
15.4 Hybrid ARQ Schemes 477
15.5 Class of Half-Rate Invertible Codes 481

Ex.1020 / Page 11 of 305
Sandisk Technologies, Inc. et al

15.6 Type II Hybrid Selective-Repeat ARQ with Finite Receiver Buffer 483
Problems 494
References 495

CHAPTER 16 APPLICATIONS OF BLOCK CODES FOR ERROR CONTROL IN DATA STORAGE SYSTEMS 498

16.1 Error Control for Computer Main Processor and Control Storages 498
16.2 Error Control for Magnetic Tapes 503
16.3 Error Control in IBM 3850 Mass Storage System 516
16.4 Error Control for Magnetic Disks 525
16.5 Error Control in Other Data Storage Systems 531
Problems 532
References 532

CHAPTER 17 PRACTICAL APPLICATIONS OF CONVOLUTIONAL CODES 533

17.1 Applications of Viterbi Decoding 533
17.2 Applications of Sequential Decoding 539
17.3 Applications of Majority-Logic Decoding 543
17.4 Applications to Burst-Error Correction 547
17.5 Applications of Convolutional Codes in ARQ Systems 551
Problems 556
References 557

Appendix A Tables of Galois Fields 561

Appendix B Minimal Polynomials of Elements in GF(2^m) 579

Appendix C Generator Polynomials of Binary Primitive BCH Codes of Length up to $2^{10} - 1$ 583

INDEX 599

Ex.1020 / Page 12 of 305
Sandisk Technologies, Inc. et al

Preface

This book owes its beginnings to the pioneering work of Claude Shannon in 1948 on achieving reliable communication over a noisy transmission channel. 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, Slepian, and others in the early 1950s, has occupied the energies of many researchers since then. Much of this work is highly mathematical in nature, and requires an extensive background in modern algebra and probability theory to understand. This has acted as an impediment to many practicing engineers and computer scientists, who are interested in applying these techniques to real 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.

Work on coding 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. The present book takes the same approach to covering the fundamentals of coding. However, the entire manuscript has been rewritten and much new material has been added. In particular, during the 1970s the emphasis in coding research shifted from theory to practical applications. Consequently, three completely new chapters on the applications of coding to digital transmission and storage systems have been added. Other major additions include a comprehensive treatment of the error-detecting capabilities of block codes, and an emphasis on probabilistic decoding methods for convolutional codes. A brief description of each chapter follows.

Chapter 1 presents an overview of coding for error control in data transmission

Ex.1020 / Page 14 of 305
Sandisk Technologies, Inc. et al

and storage systems. A brief discussion of modulation and demodulation serves to place coding in the context of a complete system. Chapter 2 develops those concepts from modern algebra that are necessary to an understanding of 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 8 cover in detail block codes for random-error correction. The fundamentals of linear codes are presented in Chapter 3. Also included is an extensive section on error detection with linear codes, an important topic which is discussed only briefly in most other books on coding. Most linear codes used in practice are cyclic codes. The basic structure and properties of cyclic codes are presented in Chapter 4. A simple way of decoding some cyclic codes, known as error-trapping decoding, is covered in Chapter 5. The important class of BCH codes for multiple-error correction 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. Chapters 7 and 8 provide detailed coverage of majority-logic decoding and majority-logic decodable codes. The material on fundamentals of block codes concludes with Chapter 9 on burst-error correction. This discussion includes codes for correcting a combination of burst and random errors.

Chapters 10 through 14 are devoted to the presentation of the fundamentals of convolutional codes. Convolutional codes are introduced in Chapter 10, with the encoder state diagram serving as the basis for studying code structure and distance properties. The Viterbi decoding algorithm for both hard and soft demodulator decisions is covered in Chapter 11. A detailed performance analysis based on code distance properties is also included. Chapter 12 presents the basics of sequential decoding using both the stack and Fano algorithms. The difficult problem of the computational performance of sequential decoding is discussed without including detailed proofs. Chapter 13 covers majority-logic decoding of convolutional codes. The chapter concludes with a comparison of the three primary decoding methods for convolutional codes. Burst-error-correcting convolutional codes are presented in Chapter 14. A section is included on convolutional codes that correct a combination of burst and random errors. Burst-trapping codes, which embed a block code in a convolutional code, are also covered here.

Chapters 15 through 17 cover a variety of applications of coding to modern day data communication and storage systems. Although they are not intended to be comprehensive, they are representative of the many different ways in which coding is used as a method of error control. This emphasis on practical applications makes the book unique in the coding literature. Chapter 15 is devoted to automatic-repeat-request (ARQ) error control schemes used for data communications. Both pure ARQ (error detection with retransmission) and hybrid ARQ (a combination of error correction and error detection with retransmission) are discussed. Chapter 16 covers the application of block codes for error control in data storage systems. Coding techniques for computer memories, magnetic tape, magnetic disk, and optical storage systems are included. Finally, Chapter 17 presents a wide range of applications of convolutional codes to digital communication systems. Codes actually used on many space and satellite systems are included, as well as a section on using convolutional codes in a hybrid ARQ system.

Ex.1020 / Page 15 of 305
Sandisk Technologies, Inc. et al

Several additional features are included to 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 are given which include details of algebraic structure used in the construction of block codes. Many tables of the best known codes for a given decoding structure are presented throughout the book. These should prove valuable to designers looking for the best code for a particular application. A set of problems is given at the end of each chapter. Most of the problems are relatively straightforward applications of material covered in the text, although some more advanced problems are also included. There are a total of over 250 problems. A solutions manual will be made available to instructors using the text. Over 300 references are also included. 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 error-correcting codes and their applications at the senior or beginning graduate level. It can also 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 and applications, with Chapters 1 through 9 on block codes covered in one semester and Chapters 10 through 17 on convolutional codes and applications in a second semester. Alternatively, portions of the book can be covered in a one-semester course. One possibility is to cover Chapters 1 through 6 and 10 through 12, which include the basic fundamentals of both block and convolutional codes. A course on block codes and applications can be comprised of Chapters 1 through 6, 9, 15, and 16, whereas Chapters 1 through 3, 10 through 14, and 17 include convolutional codes and applications as well as the rudiments of block codes. Preliminary versions of the notes on which the book is based have been classroom tested by both authors for university courses and for short courses in industry, with very gratifying results.

It is difficult to identify the many individuals who have influenced this work over the years. Naturally, we both owe a great deal of thanks to our thesis advisors, Professors Paul E. Pfeiffer and James L. Massey. Without their stimulating our interest in this exciting field and their constant encouragement and guidance through the early years of our research, this book would not have been possible.

Much of the material in the first half of the book on block codes owes a great deal to Professors W. Wesley Peterson and Tadao Kasami. Their pioneering work in algebraic coding and their valuable discussions and suggestions had a major impact on the writing of this material. The second half of the book on convolutional codes was greatly influenced by Professor James L. Massey. His style of clarifying the basic elements in highly complex subject matter was instrumental throughout the preparation of this material. In particular, most of Chapter 14 was based on a set of notes that he prepared.

We are grateful to the National Science Foundation, and to Mr. Elias Schutzman, for their continuing support of our research in the coding field. Without this assistance, our interest in coding could never have developed to the point of writing

Ex.1020 / Page 16 of 305
Sandisk Technologies, Inc. et al

this book. We thank the University of Hawaii and Illinois Institute of Technology for their support of our efforts in writing this book and for providing facilities. We also owe thanks to Professor Franklin F. Kuo for suggesting that we write this book, and for his constant encouragement and guidance during the preparation of the manuscript. Another major source of stimulation for this effort came from our graduate students, who have provided a continuing stream of new ideas and insights. Those who have made contributions directly reflected in this book include Drs. Pierre Chevillat, Farhad Hemmati, Alexander Drukarev, and Michael J. Miller.

We would like to express our special appreciation to Professors Tadao Kasami, Michael J. Miller, and Yu-ming Wang, who read the first draft very carefully and made numerous corrections and suggestions for improvements. We also wish to thank our secretaries for their dedication and patience in typing this manuscript. Deborah Waddy and Michelle Masumoto deserve much credit for their perseverence in preparing drafts and redrafts of this work.

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

Shu Lin
Daniel J. Costello, Jr.

Ex.1020 / Page 17 of 305
Sandisk Technologies, Inc. et al

ERROR CONTROL CODING

Fundamentals and Applications

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 military, governmental, and private spheres. A merging of communications and computer technology is required in the design of these systems. A major concern of the designer is the control of errors so that reliable reproduction of data can be obtained.

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. Since Shannon's work, a great deal of 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 systems and digital computers.

The transmission and storage of digital information have much in common. They both 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 (e.g., a digital computer). The source output, which is to be communicated to the destination, can be either a continuous waveform or a sequence of discrete symbols.

Ex.1020 / Page 20 of 305
Sandisk Technologies, Inc. et al

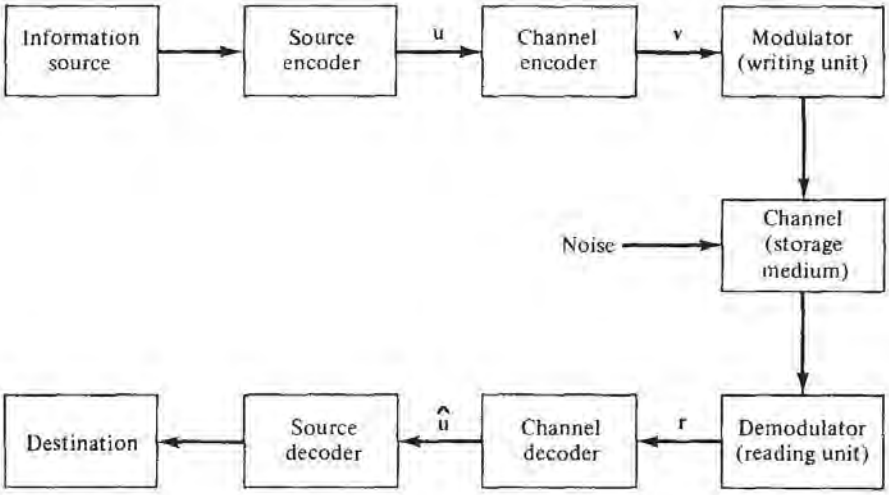


Figure 1.1 Block diagram of a typical data transmission or storage system.

The *source encoder* transforms the source output into a sequence of binary digits (bits) called the *information sequence* **u**. In the case of a continuous source, this 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 reconstructed from the information sequence **u** without ambiguity. The subject of source coding is not discussed in this book. For a thorough treatment of this important topic, see References 2 and 3.

The *channel encoder* transforms the information sequence **u** into a discrete *encoded sequence* **v** called a *code word*. In most instances **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 code words must be transmitted or stored is one of the major topics of this book.

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 which 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, high-frequency radio links, telemetry links, microwave links, satellite links, and so on. Typical storage media include core and semiconductor memories, magnetic tapes, drums, disk files, 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, crosstalk from other lines, or lightning. On magnetic tape, surface defects are regarded as a noise disturbance. The *demodulator* (or *reading unit*) processes each received waveform of duration T and produces an output that may be discrete (quantized) or continuous (unquantized). 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 $\hat{\mathbf{u}}$ called the *estimated sequence*. The decoding strategy is based on the rules of channel encoding and the noise characteristics of the channel (or storage medium). Ideally, $\hat{\mathbf{u}}$

Ex.1020 / Page 21 of 305
Sandisk Technologies, Inc. et al

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 sequence* $\hat{\mathbf{u}}$ into an *estimate* of the source output and delivers this estimate to the *destination*. When the source is continuous, this 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 are combined into a *digital source* with output **u**; (2) the modulator (or writing unit), the channel (or storage medium), and the demodulator (or reading unit) are combined into a *coding channel* with input **v** and output **r**; and (3) the source decoder and destination are combined into a *digital sink* with input $\hat{\mathbf{u}}$. A simplified block diagram is shown in Figure 1.2.

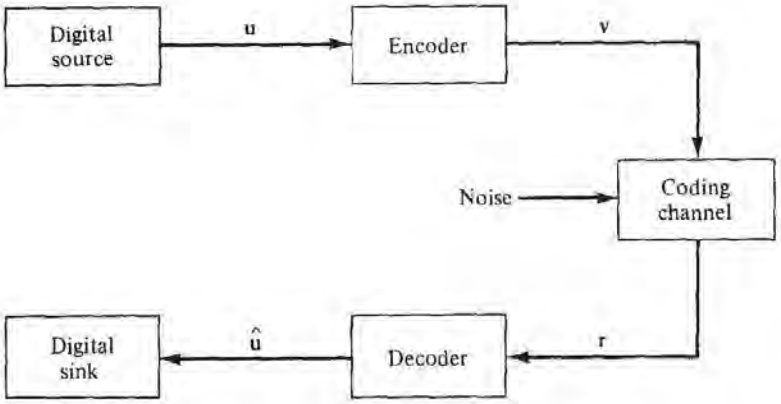


Figure 1.2 Simplified model of a coded system.

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 as possible, (2) reliable reproduction of the information can be obtained 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

There are two different types of codes 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 each. A message block is represented by the binary k-tuple $\mathbf{u} = (u_1, u_2, \ldots, u_k)$ called a *message*. (In block coding, the symbol **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 **u** independently into an n-tuple $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ of discrete symbols called a *code word*. (In block coding, the symbol **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 code words at the encoder

Ex.1020 / Page 22 of 305
Sandisk Technologies, Inc. et al

output. This set of 2^k code words of length n is called an (n, k) *block code*. The ratio $R = k/n$ is called the *code rate*, and can be interpreted as the number of information bits entering the encoder per transmitted symbol. Since the n-symbol output code word depends only on the corresponding k-bit input message, the encoder is memoryless, and can be implemented with a combinational logic circuit.

In a binary code, each code word $\mathbf{v}$ is also binary. Hence, for a binary code to be useful (i.e., to have a different code word assigned to each message), $k \leq n$ or $R \leq 1$. When $k < n$, $n - k$ redundant bits can be added to each message to form a code word. 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 block length n of the code while holding the ratio k/n constant. How to choose these redundant bits to achieve reliable transmission 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 9 are devoted to the analysis and design of block codes for controlling errors in a noisy environment.

TABLE 1.1 BINARY BLOCK CODE WITH $k = 4$ AND $n = 7$

Messages	Code words
(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)

The encoder for a convolutional code also accepts k-bit blocks of the information sequence $\mathbf{u}$ and produces an encoded sequence (code word) $\mathbf{v}$ 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 encoded sequences produced by a k-input, n-output encoder of memory order m is called an (n, k, m) *convolutional code*. The ratio $R = k/n$ is called the *code rate*. Since 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

Ex.1020 / Page 23 of 305
Sandisk Technologies, Inc. et al

can be 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. An example of a binary convolutional encoder with $k = 1$, $n = 2$, and $m = 2$ is shown in Figure 1.3. As an illustration of how code words are generated, consider the information sequence $\mathbf{u} = (1\ 1\ 0\ 1\ 0\ 0\ 0\ \ldots)$, where the leftmost bit is assumed to enter the encoder first. Using the rules of exclusive-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 $\mathbf{v} = (1\ 1,\ 1\ 0,\ 1\ 0,\ 0\ 0,\ 0\ 1,\ 1\ 1,\ 0\ 0,\ 0\ 0,\ 0\ 0,\ \ldots)$. Chapters 10 through 14 are devoted to the analysis and design of convolutional codes for controlling errors in a noisy environment.

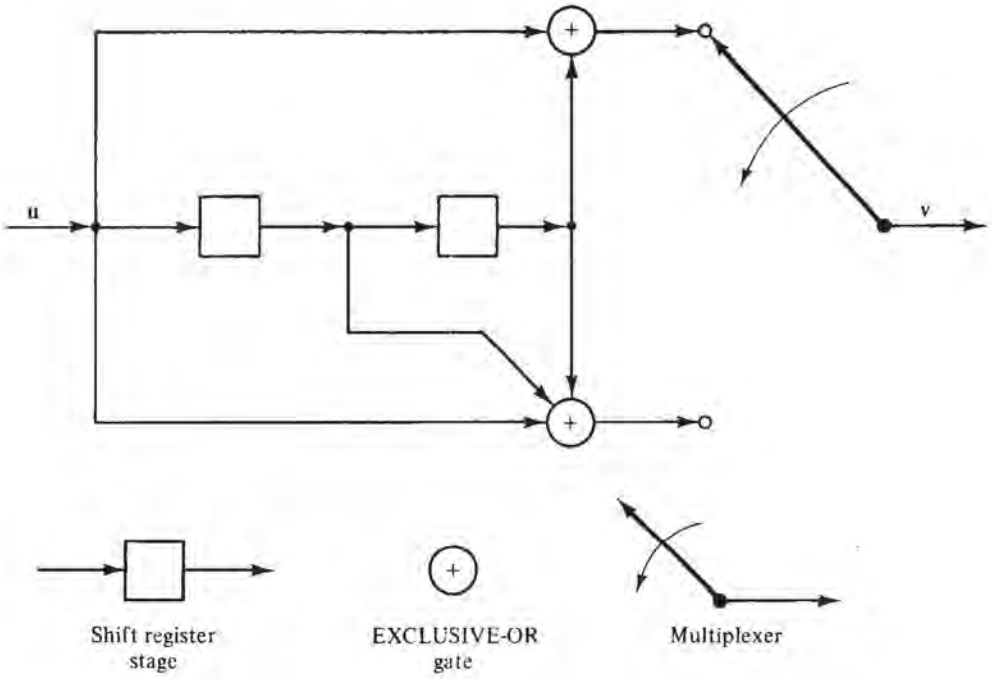


Figure 1.3 Binary convultional encoder with $k = 1$, $n = 2$, and $m = 2$.

1.3 MODULATION AND DEMODULATION

The modulator in a communication system must select a waveform of duration T seconds, which is suitable for transmission, for each encoder output symbol. In the case of a binary code, the modulator must generate one of two signals, $s_0(t)$ for an encoded "0" or $s_1(t)$ for an encoded "1." For a wideband channel, the optimum choice of signals is

$$s_0(t) = \sqrt{\frac{2E}{T}} \sin\left(2\pi f_0 t + \frac{\pi}{2}\right), \qquad 0 \le t \le T$$
$$s_1(t) = \sqrt{\frac{2E}{T}} \sin\left(2\pi f_0 t - \frac{\pi}{2}\right), \qquad 0 \le t \le T, \tag{1.1}$$

Ex.1020 / Page 24 of 305
Sandisk Technologies, Inc. et al

where f_0 is a multiple of $1/T$ and E is the energy of each signal. This is called *binary-phase-shift-keyed* (BPSK) modulation, since the transmitted signal is a sine-wave pulse whose phase is either $+\pi/2$ or $-\pi/2$, depending on the encoder output. The BPSK modulated waveform corresponding to the code word $\mathbf{v} = (1\ 1\ 0\ 1\ 0\ 0\ 0)$ in the code of Table 1.1 is shown in Figure 1.4.

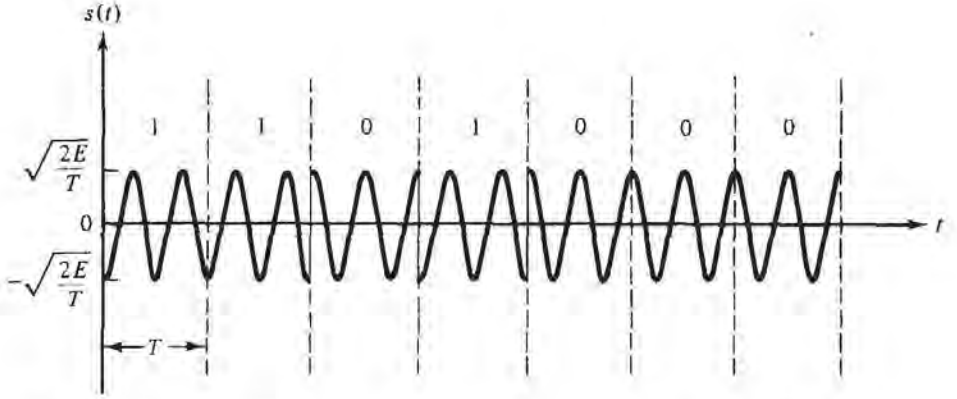


Figure 1.4 BPSK modulated waveform corresponding to the code word $\mathbf{v} = (1\ 1\ 0\ 1\ 0\ 0\ 0)$.

A common form of noise disturbance present in any communication system is *additive white Gaussian noise* (AWGN). If the transmitted signal is $s(t)$ [$= s_0(t)$ or $s_1(t)$], the received signal is

$$r(t) = s(t) + n(t), \tag{1.2}$$

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. 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 component 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 sampling switch. For BPSK modulation with coherent detection the sampled output is a real number,

$$\rho = \int_0^T r(t)\sqrt{\frac{2E}{T}} \sin\left(2\pi f_0 t + \frac{\pi}{2}\right) dt. \tag{1.3}$$

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 analog inputs; that is, it must be an *analog decoder*. A much more common approach to decoding is to quantize the continuous detector output ρ into one of a finite number Q of discrete output symbols. In this case, the channel decoder has discrete inputs; that is, it must be a *digital decoder*. Almost all coded communication systems use some form of digital decoding.

If the detector output in a given interval depends only on the transmitted signal

Ex.1020 / Page 25 of 305
Sandisk Technologies, Inc. et al

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* $P(j|i)$, $0 \le i \le M-1$, $0 \le j \le Q-1$, 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 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 1.5(a). Note that the transition probability p completely describes the channel.

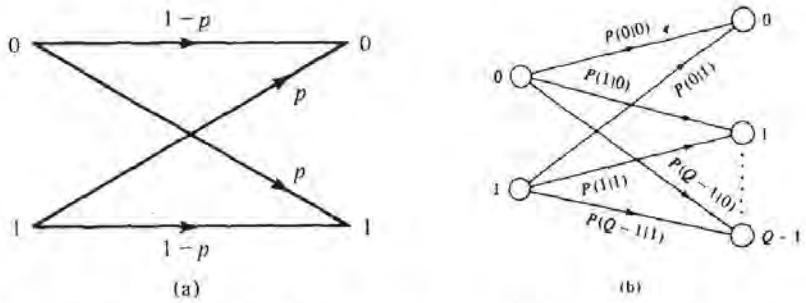


Figure 1.5 Transition probability diagrams: (a) binary symmetric channel; (b) binary-input, Q-ary-output discrete memoryless channel.

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 BPSK bit error probability for equally likely signals given by

$$p = Q\left(\sqrt{\frac{2E}{N_0}}\right), \tag{1.4}$$

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

$$Q(x) \le \tfrac{1}{2} e^{-x^2/2}, \qquad x \ge 0. \tag{1.5}$$

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*. Most coded digital communication systems, whether block or convolutional, use binary coding with hard-decision decoding, owing to the resulting simplicity of

Ex.1020 / Page 26 of 305
Sandisk Technologies, Inc. et al

implementation compared to nonbinary systems. However, some binary coded systems do not use hard decisions at the demodulator output. 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 analog) inputs. Although this makes the decoder more difficult to implement, soft-decision decoding offers significant performance improvement over hard-decision decoding, as discussed in Chapter 11. A transition probability diagram for a soft-decision DMC with $M = 2$ and $Q > 2$ is shown in Figure 1.5(b). 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. For a more thorough treatment of the calculation of DMC transition probabilities, see References 4 and 5.

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 normally done on an ad hoc basis.

Two important and related parameters in any digital communication system are the speed of information transmission and the bandwidth of the channel. Since 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, all communication channels are subject to signal distortion due to bandwidth limitations. To minimize the effect of this distortion on the detection process, the channel should have a *bandwidth* W of roughly $1/2T$ hertz (Hz).[1] In an uncoded system ($R = 1$), the data rate is $1/T = 2W$, and is limited by the channel bandwidth. In a binary-coded system, with a code rate $R < 1$, the data rate is $R/T = 2RW$, and is reduced by the factor R compared to an uncoded system. Hence, to maintain the same data rate as the uncoded system, the coded system requires a *bandwidth expansion* by a factor of $1/R$. This is characteristic of binary-coded systems: they require some bandwidth expansion to maintain a constant data rate. If no additional bandwidth is available without undergoing severe signal distortion, binary coding is not feasible, and other means of reliable communication must be sought.[2]

1.4 MAXIMUM LIKELIHOOD DECODING

A block diagram of a coded system on an AWGN channel with finite output quantization is shown in Figure 1.6. In a block-coded system, the source output $\mathbf{u}$ represents a k-bit message, the encoder output $\mathbf{v}$ represents an n-symbol code word, the demodu-

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

[2]This does not preclude the use of coding, but requires only that a larger set of signals be found. See References 4 to 6.

Ex.1020 / Page 27 of 305
Sandisk Technologies, Inc. et al

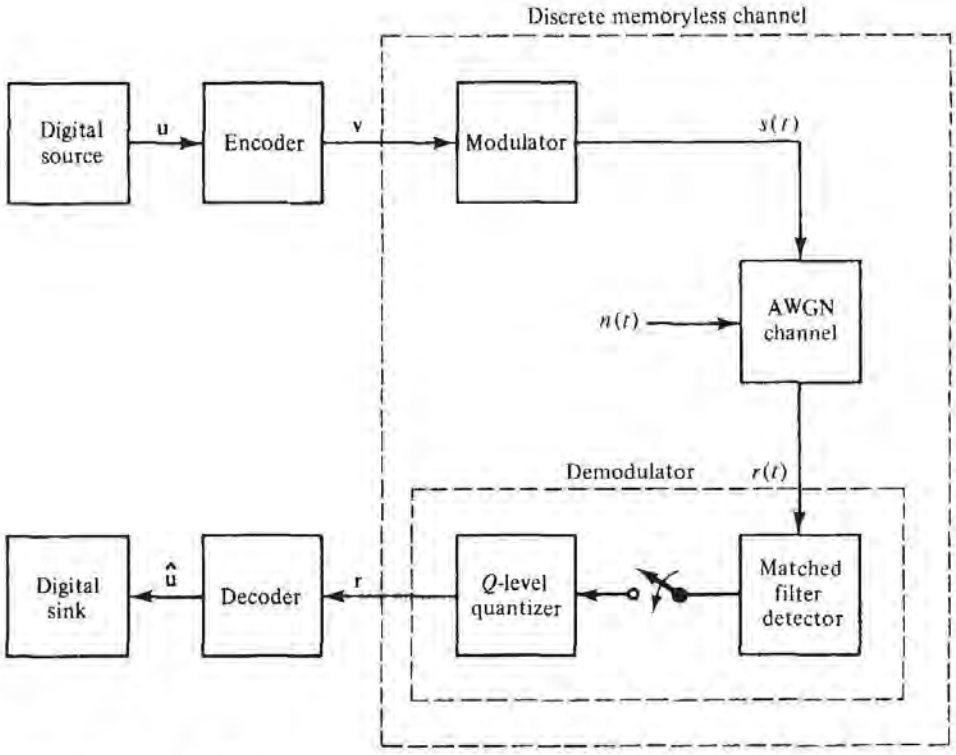


Figure 1.6 Coded system on an additive white Gaussian noise channel.

lator 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 convolutional coded system, $\mathbf{u}$ represents a sequence of kL information bits and $\mathbf{v}$ represents a code word containing $N \triangleq nL + nm = n(L + m)$ symbols, where kL is the length of the information sequence and N is the length of the code word. The additional nm encoded symbols are produced after the last block of information bits has entered the encoder. This is due to the m time unit memory of the encoder, and is discussed more fully in Chapter 10. The demodulator output $\mathbf{r}$ is a Q-ary received N-tuple, and the decoder output $\hat{\mathbf{u}}$ is a kL-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 code word $\mathbf{v}$, the decoder can produce an estimate $\hat{\mathbf{v}}$ of the code word $\mathbf{v}$. 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 code word $\hat{\mathbf{v}}$ for each possible received sequence $\mathbf{r}$. If the code word $\mathbf{v}$ 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

$$P(E|\mathbf{r}) \triangleq P(\hat{\mathbf{v}} \neq \mathbf{v}|\mathbf{r}). \tag{1.6}$$

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

$$P(E) = \sum_{\mathbf{r}} P(E|\mathbf{r})P(\mathbf{r}). \tag{1.7}$$

$P(\mathbf{r})$ is independent of the decoding rule used since $\mathbf{r}$ is produced prior to decoding. Hence, an optimum decoding rule [i.e., one that minimizes $P(E)$] must minimize $P(E|\mathbf{r}) = P(\hat{\mathbf{v}} \neq \mathbf{v}|\mathbf{r})$ for all $\mathbf{r}$. Since minimizing $P(\hat{\mathbf{v}} \neq \mathbf{v}|\mathbf{r})$ is equivalent to maximiz-

Ex.1020 / Page 28 of 305
Sandisk Technologies, Inc. et al

ing $P(\hat{\mathbf{v}} = \mathbf{v}|\mathbf{r})$, $P(E|\mathbf{r})$ is minimized for a given $\mathbf{r}$ by choosing $\hat{\mathbf{v}}$ as the code word $\mathbf{v}$ which maximizes

$$P(\mathbf{v}|\mathbf{r}) = \frac{P(\mathbf{r}|\mathbf{v})P(\mathbf{v})}{P(\mathbf{r})}, \tag{1.8}$$

that is, $\hat{\mathbf{v}}$ is chosen as the most likely code word given that $\mathbf{r}$ is received. If all information sequences, and hence all code words, are equally likely [i.e., $P(\mathbf{v})$ is the same for all $\mathbf{v}$], maximizing (1.8) is equivalent to maximizing $P(\mathbf{r}|\mathbf{v})$. For a DMC

$$P(\mathbf{r}|\mathbf{v}) = \prod_i P(r_i|v_i), \tag{1.9}$$

since for a memoryless channel each received symbol depends only on the corresponding transmitted symbol. A decoder that chooses its estimate to maximize (1.9) is called a *maximum likelihood decoder* (MLD). Since $\log x$ is a monotone increasing function of x, maximizing (1.9) is equivalent to maximizing the *log-likelihood function*

$$\log P(\mathbf{r}|\mathbf{v}) = \sum_i \log P(r_i|v_i). \tag{1.10}$$

An MLD for a DMC then chooses $\hat{\mathbf{v}}$ as the code word $\mathbf{v}$ that maximizes the sum in (1.10). If the code words are not equally likely, an MLD is not necessarily optimum, since the conditional probabilities $P(\mathbf{r}|\mathbf{v})$ must be weighted by the code word probabilities $P(\mathbf{v})$ to determine which code word maximizes $P(\mathbf{v}|\mathbf{r})$. However, in many systems, the code word 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 which may differ from the transmitted code word $\mathbf{v}$ in some positions because of 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 $\mathbf{v}$ (i.e., the number of positions in which $\mathbf{r}$ and $\mathbf{v}$ differ). For a block code of length n, (1.10) becomes

$$\begin{aligned} \log P(\mathbf{r}|\mathbf{v}) &= d(\mathbf{r}, \mathbf{v}) \log p + [n - d(\mathbf{r}, \mathbf{v})] \log (1 - p) \\ &= d(\mathbf{r}, \mathbf{v}) \log \frac{p}{1 - p} + n \log (1 - p). \end{aligned} \tag{1.11}$$

[For a convolutional code, n in (1.11) is replaced by N.] Since $\log [p/(1 - p)] < 0$ 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 code word* $\mathbf{v}$ *which minimizes the distance* $d(\mathbf{r}, \mathbf{v})$ *between* $\mathbf{r}$ *and* $\mathbf{v}$; *that is, it chooses the code word 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*.

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 *channel capacity* C, and that for any rate $R < C$, there exists codes of rate R which, with maximum likelihood decoding, have an arbitrarily small decoding error probability $P(E)$. In particular, for any $R < C$, there exists block codes of length n such that

$$P(E) \leq 2^{-nE_b(R)}, \tag{1.12}$$

and there exists convolutional codes of *memory order* m such that

$$P(E) \leq 2^{-(m+1)nE_c(R)} = 2^{-n_A E_c(R)}, \tag{1.13}$$

Ex.1020 / Page 29 of 305
Sandisk Technologies, Inc. et al

where $n_A \triangleq (m+1)n$ is called the code *constraint length*. $E_b(R)$ and $E_c(R)$ are positive functions of R for $R < C$ and are completely determined by the channel characteristics. The bound of (1.12) 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 (1.13) implies that arbitrarily small error probabilities are achievable with convolutional coding for any fixed $R < C$ by increasing the constraint length n_A (i.e., 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. Since some codes must perform better than the average, the noisy channel coding theorem guarantees the existence of codes satisfying (1.12) and (1.13), but 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 code words $2^k = 2^{nR}$ must be very large. Since a MLD must compute $\log P(\mathbf{r}|\mathbf{v})$ for each code word, and then choose the code word that gives the maximum, the number of computations that must be performed by a MLD becomes excessively large. For convolutional codes, low error probabilities require a large memory order m. As will be seen in Chapter 11, a MLD for convolutional codes requires approximately 2^{km} computations to decode each block of k information bits. This, too, becomes excessively 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 maximum likelihood decoding would satisfy (1.12) and (1.13), 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 maximum likelihood decoding. 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 1.5(a). Each transmitted bit has a probability p of being received incorrectly and a probability $1-p$ of being received correctly, independently of 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 affected primarily by random errors. The codes devised for correcting random errors are called *random-error-correcting codes*. Most of the codes presented in this book are random-error-correcting codes. In particular, Chapters 3 through 8 and 10 through 13 are devoted to codes of this type.

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 1.7. 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

Ex.1020 / Page 30 of 305
Sandisk Technologies, Inc. et al

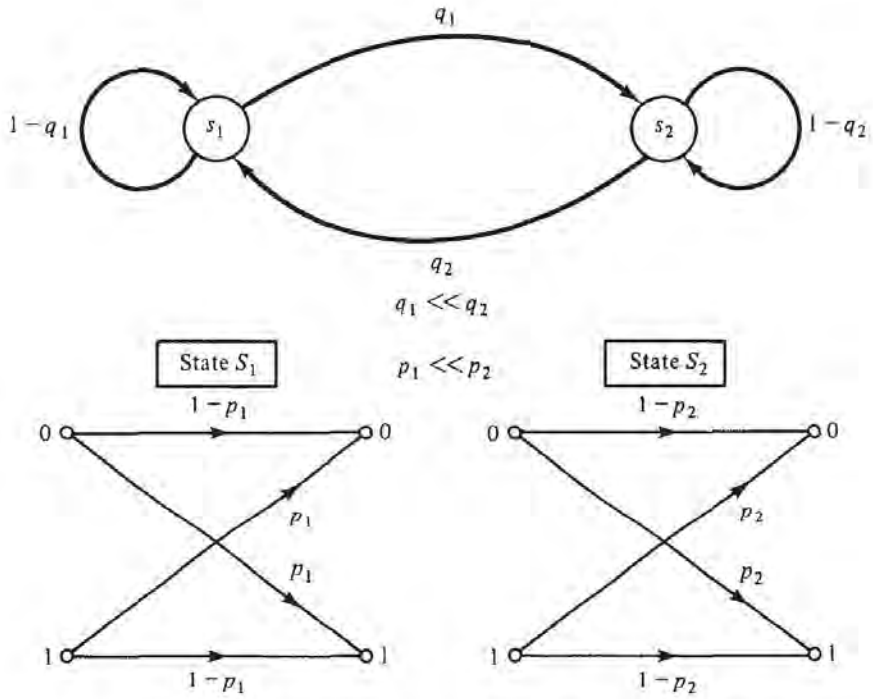


Figure 1.7 Simplified model of a channel with memory.

probable, $p_2 \approx 0.5$. The channel is in the good state most of the time, but on occasion shifts to the bad state due to a change in the transmission characteristic of the channel (e.g., 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 radio channels, where the error bursts are caused by signal fading due to multipath transmission, wire and cable transmission, which is affected by impulsive switching noise and crosstalk, and magnetic recording, which is subject to tape dropouts due to surface defects and dust particles. The codes devised for correcting burst errors are called *burst-error-correcting codes*. Sections 9.1 to 9.5 and 14.1 to 14.3 are devoted to codes of this type.

Finally, some channels contain a combination of both random and burst errors. These are called *compound channels*, and codes devised for correcting errors on these channels are called *burst-and-random-error-correcting codes*. Sections 9.6, 9.7, and 14.4 are devoted to codes of this type.

1.6 ERROR CONTROL STRATEGIES

The block diagram shown in Figure 1.1 represents a one-way system. The transmission (or recording) is strictly in one direction, from transmitter to receiver. Error control for a one-way system must be accomplished using *forward error correction* (FEC), that is, by employing error-correcting codes that automatically correct errors detected at the receiver. Examples are magnetic tape storage systems, in which the

Ex.1020 / Page 31 of 305
Sandisk Technologies, Inc. et al

information recorded on tape 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. Applications of FEC to storage and communication systems are presented in Chapter 16 and Sections 17.1 to 17.4.

In some cases, 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 telephone channels and some satellite communication systems. Error control for a two-way system can be accomplished using 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 this continues until the message is received correctly.

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

With *continuous ARQ*, the transmitter sends code words to the receiver continuously and receives acknowledgments continuously. When a NAK is received, the transmitter begins a retransmission. It may back up to the code word in error and resend that word plus the words that follow it. This is called *go-back-N ARQ*. Alternatively, the transmitter may simply resend only those code words that are acknowledged negatively. 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. 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 code word is long compared to the time taken to receive an acknowledgment. Stop-and-wait ARQ is designed for use on half-duplex channels, whereas continuous ARQ is designed for use on full-duplex channels.

The major advantage of ARQ over FEC is that error detection requires much simpler decoding equipment than does error correction. Also, ARQ is adaptive in the sense that information is retransmitted only when errors occur. On the other hand, 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 combination of FEC for the most frequent error patterns, together with error detection and retransmission for the less likely error patterns, is more efficient than ARQ alone. Although this hybrid ARQ error control strategy has not been implemented in many systems, it clearly carries

Ex.1020 / Page 32 of 305
Sandisk Technologies, Inc. et al

the potential for improving throughput in two-way systems subject to a high channel error rate. Various types of ARQ and hybrid ARQ schemes are discussed in Chapter 15 and Section 17.5.

REFERENCES

1. C. E. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. J.*, 27, pp. 379–423 (Part I), 623–656 (Part II), July 1948.
2. T. Berger, *Rate Distortion Theory*, Prentice-Hall, Englewood Cliffs, N.J., 1971.
3. L. Davisson and R. Gray, eds., *Data Compression*, Dowden, Hutchinson, & Ross, Stroudsburg, Pa., 1976.
4. J. M. Wozencraft and I. M. Jacobs, *Principles of Communication Engineering*, Wiley, New York, 1965.
5. A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*, McGraw-Hill, New York, 1979.
6. R. G. Gallager, *Information Theory and Reliable Communication*, Wiley, New York, 1968.

Ex.1020 / Page 33 of 305
Sandisk Technologies, Inc. et al

2

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. Birkhoff and MacLane [2] is probably the most easily understood text on modern algebra. Fraleigh [4] is also a good and fairly simple text.

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.

Ex.1020 / Page 34 of 305
Sandisk Technologies, Inc. et al

(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 noted above 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:

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

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

Ex.1020 / Page 35 of 305
Sandisk Technologies, Inc. et al

a group of order m under a binary operation which is very similar to real addition. This is shown in the next example.

Example 2.2

Let m be a positive integer. Consider the set of integers $G = \{0, 1, 2, \ldots, m-1\}$. 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 $G = \{0, 1, \ldots, m-1\}$ 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 to each other with respect to $\boxplus$. It is also clear that the inverse of 0 is itself. Since 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. Since 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 $G = \{0, 1, 2, \ldots, m-1\}$ is a group under modulo-

Ex.1020 / Page 36 of 305
Sandisk Technologies, Inc. et al

m addition. We shall call this group an *additive* group. For $m = 2$, we obtain the binary group given in Example 2.1.

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

TABLE 2.1 MODULO-5 ADDITION

$\boxplus$	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

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

Example 2.3

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

$$i \boxdot j = r,$$

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 $\boxdot$, which is referred to as *modulo-p multiplication*. The set $G = \{1, 2, \ldots, p-1\}$ 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. Since 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

$$a \boxdot i = i \boxdot a = 1.$$

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

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

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

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

Therefore, $r \boxdot i = i \boxdot r = 1$ and r is the inverse of i. Hence, any element i in G has an inverse with respect to modulo-p multiplication. The group $G = \{1, 2, \ldots, p-1\}$

Ex.1020 / Page 37 of 305
Sandisk Technologies, Inc. et al

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 $G = \{1, 2, \ldots, p-1\}$ 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.

TABLE 2.2 MODULO-5 MULTIPLICATION

$\boxdot$	1	2	3	4
1	1	2	3	4
2	2	4	1	3
3	3	1	4	2
4	4	3	2	1

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.

2.2 FIELDS

Now, we use the group concepts to introduce another algebraic system, called a *field.* Roughly speaking, a field is a set of elements in which we can do 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 below.

Definition 2.2. Let F be a set of elements on which two binary operations, called addition "+" and multiplication "$\cdot$," are defined. The set F together with the two binary operations + and $\cdot$ 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 $\cdot$. 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

Ex.1020 / Page 38 of 305
Sandisk Technologies, Inc. et al

field. A field with 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 equality above, we have

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

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

$$a^{-1} \cdot (a \cdot b) = a^{-1} \cdot (a \cdot c)$$
$$(a^{-1} \cdot a) \cdot b = (a^{-1} \cdot a) \cdot c$$
$$1 \cdot b = 1 \cdot c.$$

Thus, $b = c$. Q.E.D.

We can verify readily 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

Ex.1020 / Page 39 of 305
Sandisk Technologies, Inc. et al

finite number of elements can be constructed and are illustrated in the next two examples and in Section 2.4.

Example 2.4

Consider the set {0, 1} together with modulo-2 addition and multiplication shown in Tables 2.3 and 2.4. In Example 2.1 we have shown that {0, 1} is a commutative group under modulo-2 addition; and in Example 2.3, we have shown 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.

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

The field given in Example 2.4 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.

Example 2.5

Let p be a prime. We have shown in Example 2.2 that the set of integers $\{0, 1, 2, \ldots, p - 1\}$ is a commutative group under modulo-p addition. We have also shown in Example 2.3 that the nonzero elements, $\{1, 2, \ldots, p - 1\}$ 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 $\{0, 1, 2, \ldots, p - 1\}$ is a field of order p under modulo-p addition and multiplication. Since 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 in a manner similar to ordinary arithmetic, with which we are quite familiar.

In Example 2.5 we have shown that, for any prime p, there exists a finite field

Ex.1020 / Page 40 of 305
Sandisk Technologies, Inc. et al

TABLE 2.5
MODULO-7 ADDITION

+	0	1	2	3	4	5	6
0	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

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 will be mainly descriptive and no attempt is made to be mathematically rigorous. Since finite-field arithmetic is very similar to ordinary arithmetic, most of the rules of ordinary 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):

$$\sum_{i=1}^{1} 1 = 1, \quad \sum_{i=1}^{2} 1 = 1 + 1, \quad \sum_{i=1}^{3} 1 = 1 + 1 + 1, \quad \ldots,$$

$$\sum_{i=1}^{k} 1 = 1 + 1 + \cdots + 1 \ (k \text{ times}), \quad \ldots$$

Since the field is closed under addition, these sums must be elements in the field. Since the field has finite number of elements, these sums cannot be all distinct. Therefore, at some point of 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 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 2.3. 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., $\lambda = km$). Since the field is closed under multiplication,

Ex.1020 / Page 41 of 305
Sandisk Technologies, Inc. et al

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

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

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

Since $\sum_{i=1}^{km} 1 = 0$, 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

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

are λ distinct elements in GF(q). In fact, this set of sums itself is a field of λ elements, GF(λ), under the addition and multiplication of GF(q) (see Problem 2.6). Since GF(λ) is a subset of GF(q), GF(λ) 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,

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

must also be nonzero elements in GF(q). Since GF(q) has only a finite number of elements, the powers of a given above cannot all be distinct. Therefore, at some point of 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 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 $a^1, a^2, a^3, \ldots$ repeats itself after $a^n = 1$. Also, the powers $a^1, a^2, \ldots, a^{n-1}, a^n = 1$ 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 $i + j \leq n$,

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

Ex.1020 / Page 42 of 305
Sandisk Technologies, Inc. et al

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 $a^1, a^2, \ldots, a^{n-1}, a^n = 1$ are closed under the multiplication of GF(q). For $1 \leq i < n$, a^{n-i} is the multiplicative inverse of a^i. Since the powers of a are nonzero elements in GF(q), they satisfy the associative and commutative laws. Therefore, we conclude that $a^n = 1, a^1, a^2, \ldots, a^{n-1}$ form a 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 2.4. Let a be a nonzero element of a finite field GF(q). Then $a^{q-1} = 1$.

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

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

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

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 2.5. 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

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

Since $a^{q-1} = 1$ 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 2.7).

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 code 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

Ex.1020 / Page 43 of 305
Sandisk Technologies, Inc. et al

and storage systems because information in these systems is universally coded in binary form for practical reasons. In this book we are concerned only with binary codes and codes with symbols from the field GF(2^m). Most of the results presented in this book 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 rest 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 the binary arithmetic, we consider the following sets 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 back into the original set of equations and verify that they are correct.

Since we were able to solve the equations shown above, 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 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:

$$X = \frac{\begin{vmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix}} = \frac{0}{1} = 0, \qquad Y = \frac{\begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix}} = \frac{1}{1} = 1,$$

$$Z = \frac{\begin{vmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix}} = \frac{0}{1} = 0.$$

Ex.1020 / Page 44 of 305
Sandisk Technologies, Inc. et al

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:

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

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 polynomial above, 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 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 $m \leq n$):

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

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

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

Ex.1020 / Page 45 of 305
Sandisk Technologies, Inc. et al

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:

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

(iii) Distributive:

$$a(X) \cdot [b(X) + c(X)] = [a(X) \cdot b(X)] + [a(X) \cdot c(X)]. \tag{2.8}$$

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 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

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

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 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 $X + 1$.

$$\begin{array}{r} X^3 + X + 1 \\ \hline X + 1 \;\overline{)}\; X^4 + X^3 + X^2 \qquad + 1 \\ \underline{X^4 + X^3} \\ X^2 \qquad + 1 \\ \underline{X^2 + X} \\ X + 1 \\ \underline{X + 1} \\ 0 \end{array}$$

Ex.1020 / Page 46 of 305
Sandisk Technologies, Inc. et al

For a polynomial $f(X)$ over GF(2), if it 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 $X + 1$. Since it 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 below without a proof.

Theorem 2.6. Any irreducible polynomial over GF(2) of degree m divides $X^{2^m-1} + 1$.

As an example of Theorem 2.6, we can check that $X^3 + X + 1$ divides $X^{2^3-1} + 1 = X^7 + 1$:

$$
\begin{array}{r}
X^4 + X^2 + X + 1 \\
\hline
X^3 + X + 1 \,)\, X^7 \qquad\qquad\qquad\qquad + 1 \\
X^7 \quad + X^5 + X^4 \\
\hline
X^5 + X^4 \qquad\qquad + 1 \\
X^5 \quad + X^3 + X^2 \\
\hline
X^4 + X^3 + X^2 \quad + 1 \\
X^4 \quad + X^2 + X \\
\hline
X^3 \quad + X + 1 \\
X^3 \quad + X + 1 \\
\hline
0.
\end{array}
$$

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 [6,7]. 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.

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

Ex.1020 / Page 47 of 305
Sandisk Technologies, Inc. et al

TABLE 2.7 LIST OF PRIMITIVE POLYNOMIALS

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

$$
\begin{aligned}
f^2(X) &= (f_0 + f_1 X + \cdots + f_n X^n)^2 \\
&= [f_0 + (f_1 X + f_2 X^2 + \cdots + f_n X^n)]^2 \\
&= f_0^2 + f_0 \cdot (f_1 X + f_2 X^2 + \cdots + f_n X^n) \\
&\qquad + f_0 \cdot (f_1 X + f_2 X^2 + \cdots + f_n X^n) + (f_1 X + f_2 X^2 + \cdots + f_n X^n)^2 \\
&= f_0^2 + (f_1 X + f_2 X^2 + \cdots + f_n X^n)^2.
\end{aligned}
$$

Expanding the equation above repeatedly, we eventually obtain

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

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

$$
\begin{aligned}
f^2(X) &= f_0 + f_1 X^2 + f_2 (X^2)^2 + \cdots + f_n (X^2)^n \\
&= f(X^2).
\end{aligned}
\tag{2.9}
$$

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

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

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 "$\cdot$" to introduce a sequence of powers of α as follows:

$$
\begin{aligned}
&0 \cdot 0 = 0, \\
&0 \cdot 1 = 1 \cdot 0 = 0, \\
&1 \cdot 1 = 1, \\
&0 \cdot \alpha = \alpha \cdot 0 = 0, \\
&1 \cdot \alpha = \alpha \cdot 1 = \alpha,
\end{aligned}
\tag{2.11}
$$

Ex.1020 / Page 48 of 305
Sandisk Technologies, Inc. et al

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

It follows from the definition of multiplication above 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 "$\cdot$" is defined:

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

The element 1 is sometimes denoted 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 "$\cdot$" defined by (2.11). Let $p(X)$ be a primitive polynomial of degree m over GF(2). We assume that $p(\alpha) = 0$. Since $p(X)$ divides $X^{2^m-1} + 1$ (Theorem 2.6), we have

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

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

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

Since $p(\alpha) = 0$, 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 $\alpha^{2^m-1} + 1 = 0$ (use modulo-2 addition) results in the following equality:

$$\alpha^{2^m-1} = 1. \tag{2.14}$$

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

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

The nonzero elements of F^* are closed under the multiplication operation "$\cdot$" 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 $0 \leq r < 2^m - 1$.

Ex.1020 / Page 49 of 305
Sandisk Technologies, Inc. et al

Then

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

which is also a nonzero element in F^*. Hence, we conclude that the nonzero elements of F^* are closed under the multiplication "$\cdot$" defined by (2.11). In fact, these nonzero elements form a commutative group under "$\cdot$". First, we see that the element 1 is the unit element. From (2.11) and (2.12) we see readily that the multiplication operation "$\cdot$" 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 what follows that $1, \alpha, \alpha^2, \ldots, \alpha^{2^m-2}$ represent $2^m - 1$ distinct elements. Therefore, the nonzero elements of F^* form a group of order $2^m - 1$ under the multiplication operation "$\cdot$" 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:

$$X^i = q_i(X)p(X) + a_i(X), \tag{2.15}$$

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:

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

Since 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 $i \geq 0$,

$$a_i(X) \neq 0. \tag{2.16}$$

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

$$a_i(X) \neq a_j(X). \tag{2.17}$$

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 $j > i$). Since 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 which 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 $i = 0, 1, 2, \ldots, 2^m - 2$, we obtain $2^m - 1$ distinct nonzero polynomials $a_i(X)$ of degree $m - 1$ or less. Now, replacing X by 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:

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

From (2.16), (2.17), and (2.18), we see that the $2^m - 1$ nonzero elements, $\alpha^0, \alpha^1, \ldots, \alpha^{2^m-2}$ 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*

Ex.1020 / Page 50 of 305
Sandisk Technologies, Inc. et al

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:

$$0 + 0 = 0 \tag{2.19a}$$

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

$$0 + \alpha^i = \alpha^i + 0 = \alpha^i, \tag{2.19b}$$

$$\begin{aligned}\alpha^i + \alpha^j &= (a_{i0} + a_{i1}\alpha + \cdots + a_{i,m-1}\alpha^{m-1}) + (a_{j0} + a_{j1}\alpha + \cdots + a_{j,m-1}\alpha^{m-1}) \\ &= (a_{i0} + a_{j0}) + (a_{i1} + a_{j1})\alpha + \cdots + (a_{i,m-1} + a_{j,m-1})\alpha^{m-1},\end{aligned} \tag{2.19c}$$

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

$$\alpha^i + \alpha^i = 0 \tag{2.20}$$

and for $i \neq j$,

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

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. Using the fact that 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 $F^* = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^m-2}\}$ is a commutative group under an addition operation "+" and the nonzero elements of F^* form a commutative group under a multiplication operation "$\cdot$." 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 $F^* = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^m-2}\}$ is a Galois field of 2^m elements, GF(2^m). We notice that the addition and multiplication defined on $F^* = \text{GF}(2^m)$ 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 2.6

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, 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,

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

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

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

Ex.1020 / Page 51 of 305
Sandisk Technologies, Inc. et al

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)

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 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 $a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{m-1}\alpha^{m-1}$ 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:

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

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, . . . ,0). Let $(b_0, b_1, \ldots, b_{m-1})$ be the m-tuple representation of γ in GF(2^m). Adding β and γ, we simply add the corresponding components of their m-tuple representations as follows:

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

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.

Ex.1020 / Page 52 of 305
Sandisk Technologies, Inc. et al

2.5 BASIC PROPERTIES OF 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 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 α^7, α^{11}, α^{13}, and α^{14} are the roots of $X^4 + X^3 + 1$. We may verify this 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 product above using Table 2.8:

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

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 is answered by the following theorem.

Theorem 2.7. 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 equation above, 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 2.7 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.

Ex.1020 / Page 53 of 305
Sandisk Technologies, Inc. et al

To verify this, we use Table 2.8 and the fact that $\alpha^{15} = 1$,

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

The conjugates of α^4 are

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

[Note that $(\alpha^4)^{2^4} = \alpha^{64} = \alpha^4$.] It follows from Theorem 2.7 that α^8, α, and α^2 must be also 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 $f(X) = 1 + X^3 + X^4 + X^5 + X^6$.

Let β be a nonzero element in the field GF(2^m). It follows from Theorem 2.4 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 $X^{2^m-1} + 1$. Since 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 result above, we obtain Theorem 2.8.

Theorem 2.8. 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 2.8 has the following corollary:

Corollary 2.8.1. The elements of GF(2^m) form all the roots of $X^{2^m} + X$.

Since 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, a number of properties of minimal polynomials are derived.

Theorem 2.9. 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 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 2.10. 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 $\phi(X)$.

Ex.1020 / Page 54 of 305
Sandisk Technologies, Inc. et al

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 equation above 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.

It follows from Corollary 2.8.1 and Theorem 2.10 that we have the following result:

Theorem 2.11. The minimal polynomial $\phi(X)$ of an element β in GF(2^m) divides $X^{2^m} + X$.

Theorem 2.11 says that all the roots of $\phi(X)$ are from GF(2^m). Then, what are the roots of $\phi(X)$? This will be answered by the next two theorems.

Theorem 2.12. 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 2.10 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 2.12 says that if an irreducible polynomial has β as a root, it is the minimal polynomial $\phi(X)$ of β. It follows from Theorem 2.7 that β and its conjugates $\beta^2, \beta^{2^2}, \ldots, \beta^{2^i}, \ldots$ are roots of $\phi(X)$. Let e be the smallest integer such that $\beta^{2^e} = \beta$. Then $\beta^2, \beta^{2^2}, \ldots, \beta^{2^{e-1}}$ are all the distinct conjugates of β (see Problem 2.14). Since $\beta^{2^m} = \beta$, $e \leq m$.

Theorem 2.13. Let β be an element in GF(2^m) and let e be the smallest non-negative 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 $(X + \beta^{2^i})^2 = X^2 + (\beta^{2^i} + \beta^{2^i})X + \beta^{2^{i+1}} = X^2 + \beta^{2^{i+1}}$,

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

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

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

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

Ex.1020 / Page 55 of 305
Sandisk Technologies, Inc. et al

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

$$[f(X)]^2 = (f_0 + f_1X + \cdots + f_eX^e)^2$$

$$= \sum_{i=0}^{e} f_i^2X^{2i} + (1+1)\sum_{i=0}^{e}\sum_{\substack{j=0 \\ i\neq j}}^{e} f_if_jX^{i+j} = \sum_{i=0}^{e} f_i^2X^{2i}. \tag{2.22}$$

From (2.21) and (2.22), we obtain

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

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

$$f_i = f_i^2.$$

This 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 $\beta, \beta^2, \ldots, \beta^{2^{s-1}}$ 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 2.12 and 13 is Theorem 2.14.

Theorem 2.14. 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

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

Example 2.7

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

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

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 equation above with the aid of Table 2.8, we obtain

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

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

Example 2.8

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

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

Ex.1020 / Page 56 of 305
Sandisk Technologies, Inc. et al

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 $\gamma, \gamma^2, \gamma^3$, and γ^4 in the equation above, we obtain the following:

$$a_0 + a_1(1 + \alpha + \alpha^3) + a_2(1 + \alpha^3) + a_3(\alpha^2 + \alpha^3) + (1 + \alpha^2 + \alpha^3) = 0$$

$$(a_0 + a_1 + a_2 + 1) + a_1\alpha + (a_3 + 1)\alpha^2 + (a_1 + a_2 + a_3 + 1)\alpha^3 = 0.$$

For the equality above to be true, we must have the coefficients equal to zero,

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

Solving the linear equations above, 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 2.14 is Theorem 2.15.

Theorem 2.15. 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 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 $p(X)$. Since the powers of α generate all the nonzero elements of GF(2^m), α is a primitive element. In fact,

Ex.1020 / Page 57 of 305
Sandisk Technologies, Inc. et al

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 2.5 that n divides $2^m - 1$,

$$2^m - 1 = k \cdot n. \tag{2.24}$$

Since 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

$$n = q \cdot (2^m - 1). \tag{2.25}$$

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 2.16. If β is a primitive element of GF(2^m), all its conjugates β^2, β^{2^2}, . . . are also primitive elements of GF(2^m).

Example 2.9

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

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

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

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

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

A more general form of Theorem 2.16 is Theorem 2.17.

Theorem 2.17. 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 2.10

Consider the element α^5 in GF(2^4) given by Table 2.8. Since $(\alpha^5)^{2^2} = \alpha^{20} = \alpha^5$, 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):

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

Ex.1020 / Page 58 of 305
Sandisk Technologies, Inc. et al

Multiplying the second equation by α^3 gives

$$X + \alpha^7 Y = \alpha^2,$$
$$X + \alpha^{11} Y = \alpha^7.$$

By adding the two equations above, we get

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

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:

$$X = \frac{\begin{vmatrix} \alpha^2 & \alpha^7 \\ \alpha^4 & \alpha^8 \end{vmatrix}}{\begin{vmatrix} 1 & \alpha^7 \\ \alpha^{12} & \alpha^8 \end{vmatrix}} = \frac{\alpha^{10} + \alpha^{11}}{\alpha^8 + \alpha^{19}} = \frac{1 + \alpha^3}{\alpha + \alpha^2} = \frac{\alpha^{14}}{\alpha^5} = \alpha^9,$$

$$Y = \frac{\begin{vmatrix} 1 & \alpha^2 \\ \alpha^{12} & \alpha^4 \end{vmatrix}}{\begin{vmatrix} 1 & \alpha^7 \\ \alpha^{12} & \alpha^8 \end{vmatrix}} = \frac{\alpha^4 + \alpha^{14}}{\alpha^8 + \alpha^{19}} = \frac{\alpha + \alpha^3}{\alpha + \alpha^2} = \frac{\alpha^9}{\alpha^5} = \alpha^4.$$

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 computations above are typical of those required for decoding a class of block codes, known as BCH 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.

Ex.1020 / Page 59 of 305
Sandisk Technologies, Inc. et al

(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 definition above.

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. Since $1 + 0 = 1$ 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 definition of a vector space given above, 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

$$\mathbf{0} = \mathbf{0} + 0\cdot\mathbf{v}$$
$$\mathbf{0} = 0\cdot\mathbf{v}.$$ Q.E.D.

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) which plays a central role in coding theory. Consider an ordered sequence of n components,

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

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). Since 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. Now, we define an addition $+$ on V_n as the following: For any $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1})$ and $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ in V_n,

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

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

Ex.1020 / Page 60 of 305
Sandisk Technologies, Inc. et al

we see that the all-zero n-tuple $\mathbf{0} = (0, 0, \ldots, 0)$ is the additive identity. For any $\mathbf{v}$ in V_n,

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

Hence, the additive inverse of each n-tuple in V_n is itself. Since 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:

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

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

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

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 2.11

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

$$\begin{array}{llll}
(0\ 0\ 0\ 0\ 0), & (0\ 0\ 0\ 0\ 1), & (0\ 0\ 0\ 1\ 0), & (0\ 0\ 0\ 1\ 1), \\
(0\ 0\ 1\ 0\ 0), & (0\ 0\ 1\ 0\ 1), & (0\ 0\ 1\ 1\ 0), & (0\ 0\ 1\ 1\ 1), \\
(0\ 1\ 0\ 0\ 0), & (0\ 1\ 0\ 0\ 1), & (0\ 1\ 0\ 1\ 0), & (0\ 1\ 0\ 1\ 1), \\
(0\ 1\ 1\ 0\ 0), & (0\ 1\ 1\ 0\ 1), & (0\ 1\ 1\ 1\ 0), & (0\ 1\ 1\ 1\ 1), \\
(1\ 0\ 0\ 0\ 0), & (1\ 0\ 0\ 0\ 1), & (1\ 0\ 0\ 1\ 0), & (1\ 0\ 0\ 1\ 1), \\
(1\ 0\ 1\ 0\ 0), & (1\ 0\ 1\ 0\ 1), & (1\ 0\ 1\ 1\ 0), & (1\ 0\ 1\ 1\ 1), \\
(1\ 1\ 0\ 0\ 0), & (1\ 1\ 0\ 0\ 1), & (1\ 1\ 0\ 1\ 0), & (1\ 1\ 0\ 1\ 1), \\
(1\ 1\ 1\ 0\ 0), & (1\ 1\ 1\ 0\ 1), & (1\ 1\ 1\ 1\ 0), & (1\ 1\ 1\ 1\ 1).
\end{array}$$

The vector sum of (1 0 1 1 1) and (1 1 0 0 1) is

$$(1\ 0\ 1\ 1\ 1) + (1\ 1\ 0\ 0\ 1) = (1 + 1, 0 + 1, 1 + 0, 1 + 0, 1 + 1) = (0\ 1\ 1\ 1\ 0).$$

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

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

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

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

V being 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.

Ex.1020 / Page 61 of 305
Sandisk Technologies, Inc. et al

Theorem 2.18. 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) say simply 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. Since 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 2.12

Consider the vector space V_5 of all 5-tuples over GF(2) given in Example 2.11. The set

$$\{(0\ 0\ 0\ 0\ 0),\ (0\ 0\ 1\ 1\ 1),\ (1\ 1\ 0\ 1\ 0),\ (1\ 1\ 1\ 0\ 1)\}$$

satisfies both conditions of Theorem 2.18, so it is a subspace of V_5.

Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ be k vectors in a vector space V over a field F. Let $a_1, a_2, \ldots, a_k$ 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 $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$. Clearly, the sum of two linear combinations of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$,

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

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

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

is also a linear combination of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$. It follows from Theorem 2.18 that we have the following result.

Theorem 2.19. Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ be k vectors in a vector space V over a field F. The set of all linear combinations of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ forms a subspace of V.

Example 2.13

Consider the vector space V_5 of all 5-tuples over GF(2) given by Example 2.11. The linear combinations of (0 0 1 1 1) and (1 1 1 0 1) are

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

These four vectors form the same subspace given by Example 2.12.

Ex.1020 / Page 62 of 305
Sandisk Technologies, Inc. et al

A set of vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ in a vector space V over a field F is said to be *linearly dependent* if and only if there exist k scalars $a_1, a_2, \ldots, a_k$ 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, $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$, is said to be *linearly independent* if it is not linearly dependent. That is, if $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ 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 2.14

The vectors (1 0 1 1 0), (0 1 0 0 1), and (1 1 1 1 1) are linearly dependent since

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

However, (1 0 1 1 0), (0 1 0 0 1), and (1 1 0 1 1) are linearly independent. All eight linear combinations of these three vectors are given below:

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

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 which 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:

$$\mathbf{e}_0 = (1, 0, 0, 0, \ldots, 0, 0)$$
$$\mathbf{e}_1 = (0, 1, 0, 0, \ldots, 0, 0)$$
$$\vdots$$
$$\mathbf{e}_{n-1} = (0, 0, 0, 0, \ldots, 0, 1),$$

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

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

Ex.1020 / Page 63 of 305
Sandisk Technologies, Inc. et al

Therefore, $\mathbf{e}_0, \mathbf{e}_1, \ldots, \mathbf{e}_{n-1}$ span the vector space V_n of all n-tuples over GF(2). From the equation above, we also see that $\mathbf{e}_0, \mathbf{e}_1, \ldots, \mathbf{e}_{n-1}$ are linearly independent. Hence, they form a basis for V_n and the dimension of V_n is n. If $k < n$ and $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ are k linearly independent vectors in V_n, then all the linear combinations of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ 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. Since each c_i has two possible values, 0 or 1, there are 2^k possible distinct linear combinations of $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$. Thus, S consists of 2^k vectors and is a k-dimensional subspace of V_n.

Let $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1})$ and $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be two n-tuples in V_n. We define the *inner product* (or *dot product*) of $\mathbf{u}$ and $\mathbf{v}$ as

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

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 $\mathbf{0} = (0, 0, \ldots, 0)$, 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 2.18 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 2.20, whose proof is omitted here [2].

Theorem 2.20. Let S be a k-dimensional space 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 2.15

Consider the vector space V_5 of all 5-tuples over GF(2) given by Example 2.11. The following eight vectors form a three-dimensional subspace S of V_5:

(0 0 0 0 0), (1 1 1 0 0), (0 1 0 1 0), (1 0 0 0 1).
(1 0 1 1 0), (0 1 1 0 1), (1 1 0 1 1), (0 0 1 1 1).

Ex.1020 / Page 64 of 305
Sandisk Technologies, Inc. et al

The null space S_d of S consists of the following 4-vectors:

$$(0\ 0\ 0\ 0\ 0),\quad (1\ 0\ 1\ 0\ 1),\quad (0\ 1\ 1\ 1\ 0),\quad (1\ 1\ 0\ 1\ 1).$$

S_d is spanned by (1 0 1 0 1) and (0 1 1 1 0), 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.

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,

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

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 $\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{k-1}$ as follows:

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

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 $\mathbf{G}'$ gave the same row space.

Example 2.16

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

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

Ex.1020 / Page 65 of 305
Sandisk Technologies, Inc. et al

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:

$$(0\ 0\ 0\ 0\ 0\ 0),\quad (1\ 0\ 0\ 1\ 0\ 1),\quad (0\ 1\ 0\ 0\ 1\ 1),\quad (0\ 0\ 1\ 1\ 1\ 0),$$
$$(1\ 1\ 0\ 1\ 1\ 0),\quad (1\ 0\ 1\ 0\ 1\ 1),\quad (0\ 1\ 1\ 1\ 0\ 1),\quad (1\ 1\ 1\ 0\ 0\ 0).$$

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 $\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{k-1}$ are linearly independent. Let S_d be the null space of S. Then the dimension of S_d is $n - k$. Let $\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{n-k-1}$ 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 $\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{n-k-1}$ as rows:

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

The row space of $\mathbf{H}$ is S_d. Since 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 in S_d, the inner product of $\mathbf{g}_i$ and $\mathbf{h}_j$ must be zero (i.e., $\mathbf{g}_i \cdot \mathbf{h}_j = 0$). Since 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 results above, we have:

Theorem 2.21. 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 2.17

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 of

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

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

Ex.1020 / Page 66 of 305
Sandisk Technologies, Inc. et al

Two matrices can be added if they have the same number of rows and the same number of columns. Adding 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 $\mathbf{B} = [b_{ij}]$, the product

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

is a $k \times l$ matrix where the entry c_{ij} is equal to the inner product of the ith row $\mathbf{a}_i$ in $\mathbf{A}$ and the jth 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 $\{1, 2, \ldots, m-1\}$ 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 $\{0, 1, 2, \ldots, m-1\}$ is not a field under modulo-m addition and multiplication.

2.6. Let λ be the characteristic of a Galois field GF(q). Let 1 be the unit element of GF(q). Show that the sums

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

form a subfield of GF(q).

2.7. Prove that every finite field has a primitive element.

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

$$\begin{aligned} X + Y \quad\quad + W &= 1, \\ X \quad\quad + Z + W &= 0, \\ X + Y + Z + W &= 1, \\ Y + Z + W &= 0. \end{aligned}$$

Ex.1020 / Page 67 of 305
Sandisk Technologies, Inc. et al

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

2.10. 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.

2.11. Find all the irreducible polynomials of degree 5 over GF(2).

2.12. 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.

2.13. 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.

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

2.15. Prove Theorem 2.17.

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

2.17. 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:

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

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

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

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

2.21. Prove that GF(2^m) is a vector space over GF(2).

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

2.23. Given the matrices

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

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

2.24. 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.

2.25. Construct the vector space of all 3-tuples over GF(3). Form a two-dimensional subspace and its dual space.

Ex.1020 / Page 68 of 305
Sandisk Technologies, Inc. et al

REFERENCES

1. A. A. Albert, *Modern Higher Algebra*, The University of Chicago Press, Chicago, 1937.
2. G. Birkhoff and S. MacLane, *A Survey of Modern Algebra*, Macmillan, New York, 1953.
3. R. D. Carmichael, *Introduction to the Theory of Group of Finite Order*, Ginn & Company, Boston, 1937.
4. J. B. Fraleigh, *A First Course in Abstract Algebra*, 2nd ed., Addison-Wesley, Reading, Mass., 1976.
5. N. Jacobson, *Lectures in Abstract Algebra*, Van Nostrand, Princeton, N.J., Vol. 1, 1951; Vol. 2, 1953; Vol. 3, 1964.
6. R. W. Marsh, *Table of Irreducible Polynomials over GF(2) through Degree 19*, NSA, Washington, D.C., 1957.
7. W. W. Peterson, *Error-Correcting Codes*, MIT Press, Cambridge, Mass., 1961.
8. B. L. Van der Waerden, *Modern Algebra*, Vols. 1 and 2, Ungar, New York, 1949.

Ex.1020 / Page 69 of 305
Sandisk Technologies, Inc. et al

3

Linear Block Codes

In this chapter basic concepts of block codes are introduced. For ease of code synthesis and implementation, we restrict our attention to a subclass of the class of all block codes, the *linear block codes*. Since in most present digital computers and digital data communication systems, information is coded in binary digits "0" or "1," we discuss only the linear block codes with symbols from the binary field GF(2). The theory developed for the binary codes can be generalized to codes with symbols from a nonbinary field in a straightforward manner.

First, linear block codes are defined and described in terms of *generator* and *parity-check* matrices. The parity-check equations for a *systematic* code are derived. Encoding of linear block codes is discussed. In Section 3.2 the concept of *syndrome* is introduced. The use of syndrome for error detection and correction is discussed. In Sections 3.3 and 3.4 we define the *minimum distance* of a block code and show that the random-error-detecting and random-error-correcting capabilities of a code are determined by its minimum distance. Probabilities of a decoding error are discussed. In Section 3.5 the *standard array* and its application to the decoding of linear block codes are presented. A general decoder based on the *syndrome decoding* scheme is given. Finally, we conclude the chapter by presenting a class of single-error-correcting linear codes.

References 1 to 4 contain excellent treatments of linear block codes.

3.1 INTRODUCTION TO LINEAR BLOCK CODES

We assume that the output of an information source is a sequence of binary digits "0" or "1." In block coding, this binary information sequence is segmented into *message* blocks of fixed length; each message block, denoted by **u**, consists of k

Ex.1020 / Page 70 of 305
Sandisk Technologies, Inc. et al

information digits. There are a total of 2^k distinct messages. The encoder, *according to certain rules*, transforms each input message $\mathbf{u}$ into a binary n-tuple $\mathbf{v}$ with $n > k$. This binary n-tuple $\mathbf{v}$ is referred to as the *code word* (or *code vector*) of the message $\mathbf{u}$. Therefore, corresponding to the 2^k possible messages, there are 2^k code words. This set of 2^k code words is called a *block* code. For a block code to be useful, the 2^k code words must be distinct. Therefore, there should be a one-to-one correspondence between a message $\mathbf{u}$ and its code word $\mathbf{v}$.

For a block code with 2^k code words and length n, unless it has a certain special structure, the encoding apparatus would be prohibitively complex for large k and n since it has to store the 2^k code words of length n in a dictionary. Therefore, we must restrict our attention to block codes that can be mechanized in a practical manner. A desirable structure for a block code to possess is the *linearity*. With this structure in a block code, the encoding complexity will be greatly reduced, as we will see.

Definition 3.1. A block code of length n and 2^k code words is called a *linear* (n, k) code if and only if its 2^k code words form a k-dimensional subspace of the vector space of all the n-tuples over the field GF(2).

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

Since 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 code words,

TABLE 3.1 LINEAR BLOCK CODE WITH $k = 4$ AND $n = 7$

Messages	Code words
(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)

Ex.1020 / Page 71 of 305
Sandisk Technologies, Inc. et al

$\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{k-1}$ in C such that every code word $\mathbf{v}$ in C is a linear combination of these k code words, that is,

$$\mathbf{v} = u_0\mathbf{g}_0 + u_1\mathbf{g}_1 + \cdots + u_{k-1}\mathbf{g}_{k-1}, \tag{3.1}$$

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

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

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

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

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 code words 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 $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$.

Example 3.1

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

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

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

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

Ex.1020 / Page 72 of 305
Sandisk Technologies, Inc. et al

A desirable property for a linear block code to possess is the *systematic structure* of the code words as shown in Figure 3.1, where a code word 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 code, the rightmost four digits of each code word are identical to the corresponding information digits.

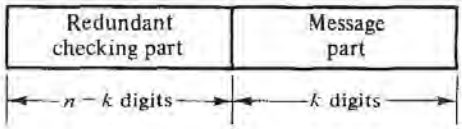


Figure 3.1 Systematic format of a code word.

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

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \mathbf{g}_2 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{g}_{k-1} \end{bmatrix} = \left[\begin{array}{cccc|ccccc} p_{00} & p_{01} & \cdots & p_{0,n-k-1} & 1 & 0 & 0 & \cdots & 0 \\ p_{10} & p_{11} & \cdots & p_{1,n-k-1} & 0 & 1 & 0 & \cdots & 0 \\ p_{20} & p_{21} & \cdots & p_{2,n-k-1} & 0 & 0 & 1 & \cdots & 0 \\ & & & & & & & & \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} & 0 & 0 & 0 & \cdots & 1 \end{array}\right], \tag{3.4}$$

($\mathbf{P}$ matrix; $k \times k$ identity matrix)

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 $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ be the message to be encoded. The corresponding code word is

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

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

$$v_{n-k+i} = u_i \qquad \text{for } 0 \leq i < k \tag{3.6a}$$

and

$$v_j = u_0 p_{0j} + u_1 p_{1j} + \cdots + u_{k-1} p_{k-1,j} \tag{3.6b}$$

for $0 \leq j < n - k$. Equation (3.6a) shows that the rightmost k digits of a code word $\mathbf{v}$ are identical to the information digits $u_0, u_1, \ldots, u_{k-1}$ 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.

Ex.1020 / Page 73 of 305
Sandisk Technologies, Inc. et al

Example 3.2

The matrix **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 code word. Then

$$\mathbf{v} = (u_0, u_1, u_2, u_3) \cdot \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

By matrix multiplication, we obtain the following digits of the code word **v**:

$$\begin{aligned} v_6 &= u_3 \\ v_5 &= u_2 \\ v_4 &= u_1 \\ v_3 &= u_0 \\ v_2 &= u_1 + u_2 + u_3 \\ v_1 &= u_0 + u_1 + u_2 \\ v_0 &= u_0 + u_2 + u_3. \end{aligned}$$

The code word 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 **G** with k linearly independent rows, there exists an $(n - k) \times n$ matrix **H** with $n - k$ linearly independent rows such that any vector in the row space of **G** is orthogonal to the rows of **H** and any vector that is orthogonal to the rows of **H** is in the row space of **G**. Hence, we can describe the (n, k) linear code generated by **G** in an alternate way as follows: *An n-tuple* **v** *is a code word in the code generated by* **G** *if and only if* $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$. This matrix **H** is called a *parity-check matrix* of the code. The 2^{n-k} linear combinations of the rows of matrix **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 **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:

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

Ex.1020 / Page 74 of 305
Sandisk Technologies, Inc. et al

where $\mathbf{P}^T$ is the transpose of the matrix $\mathbf{P}$. Let $\mathbf{h}_j$ be the jth row of $\mathbf{H}$. We can check readily that the inner product of the ith row of $\mathbf{G}$ given by (3.4) and the jth 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) can also be obtained from the parity-check matrix $\mathbf{H}$ of (3.7). Let $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ be the message to be encoded. In systematic form the corresponding code word would be

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

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

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

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 a (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, let us 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 code word 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 be implemented easily. The encoding circuit is shown in Figure 3.2, where $\rightarrow\boxed{\ }\rightarrow$ denotes a shift-register stage (e.g., a flip-flop), $\oplus$ denotes a modulo-2 adder, and $\rightarrow(p_{ij})\rightarrow$ denotes a connection if $p_{ij} = 1$ and no connection if $p_{ij} = 0$. The encoding operation is very simple. The message $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ 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.

Ex.1020 / Page 75 of 305
Sandisk Technologies, Inc. et al

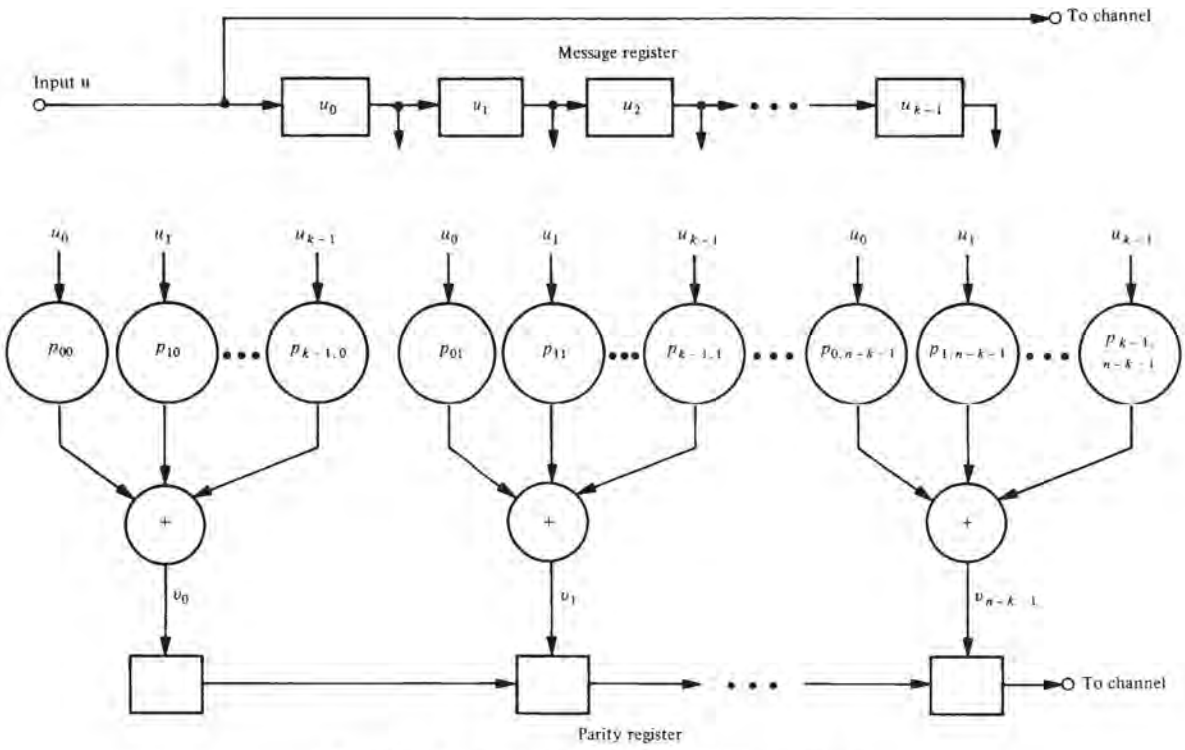


Figure 3.2 Encoding circuit for a linear systematic (n, k) code.

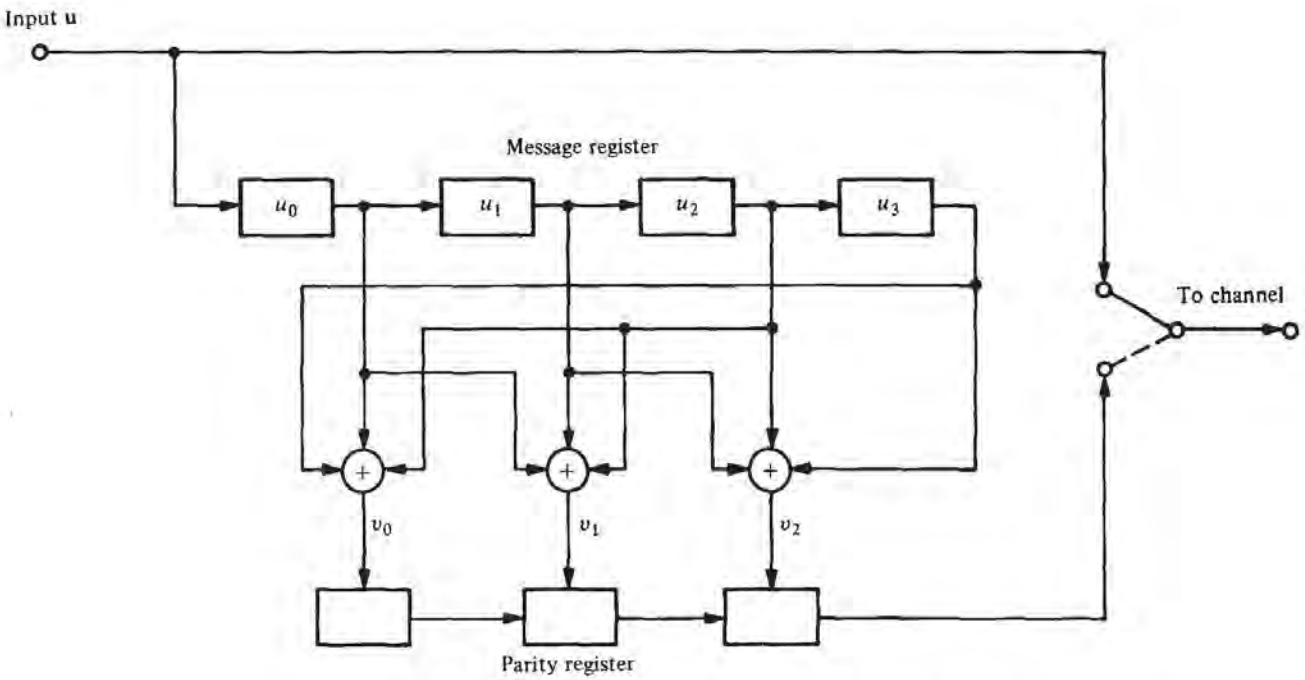


Figure 3.3 Encoding circuit for the (7, 4) systematic code given in Table 3.1.

3.2 SYNDROME AND ERROR DETECTION

Consider an (n, k) linear code with generator matrix $\mathbf{G}$ and parity-check matrix $\mathbf{H}$. Let $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be a code word that was transmitted over a noisy channel. Let $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$ 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

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

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*). 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 code word and the error vector, that is,

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

Of course, the receiver does not know either $\mathbf{v}$ or $\mathbf{e}$. Upon 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 for a retransmission of $\mathbf{v}$(ARQ).

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

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

which is called the *syndrome* of $\mathbf{r}$. Then $\mathbf{s} = \mathbf{0}$ if and only if $\mathbf{r}$ is a code word, and $\mathbf{s} \neq \mathbf{0}$

Ex.1020 / Page 77 of 305
Sandisk Technologies, Inc. et al

if and only if $\mathbf{r}$ is not a code word. Therefore, when $\mathbf{s} \neq \mathbf{0}$, we know that $\mathbf{r}$ is not a code word and the presence of errors has been detected. When $\mathbf{s} = \mathbf{0}$, $\mathbf{r}$ is a code word and the receiver accepts $\mathbf{r}$ as the transmitted code word. 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 code word. In this event, $\mathbf{r}$ is the sum of two code words which is a code word, and consequently $\mathbf{r} \cdot \mathbf{H}^T = \mathbf{0}$. Error patterns of this kind are called *undetectable* error patterns. Since there are $2^k - 1$ nonzero code words, 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 the 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 equations above carefully, we find that the syndrome $\mathbf{s}$ is simply the vector sum of the received parity digits $(r_0, r_1, \ldots, r_{n-k-1})$ and the parity-check digits recomputed from the received information digits $r_{n-k}, r_{n-k+1}, \ldots, r_{n-1}$. 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

$$\begin{aligned} \mathbf{s} &= (s_0, s_1, s_2) \\ &= (r_0, r_1, r_2, r_3, r_4, r_5, r_6) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}. \end{aligned}$$

The syndrome digits are

$$\begin{aligned} s_0 &= r_0 + r_3 + r_5 + r_6 \\ s_1 &= r_1 + r_3 + r_4 + r_5 \\ s_2 &= r_2 + r_4 + r_5 + r_6. \end{aligned}$$

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

Ex.1020 / Page 78 of 305
Sandisk Technologies, Inc. et al

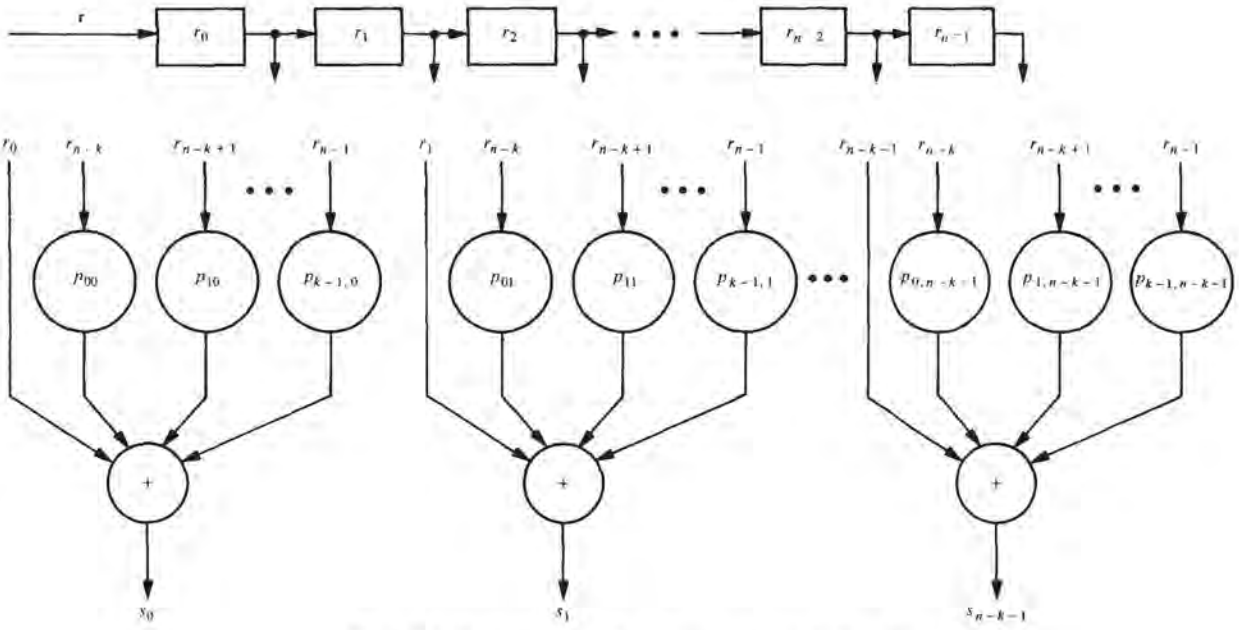


Figure 3.4 Syndrome circuit for a linear systematic (n, k) code.

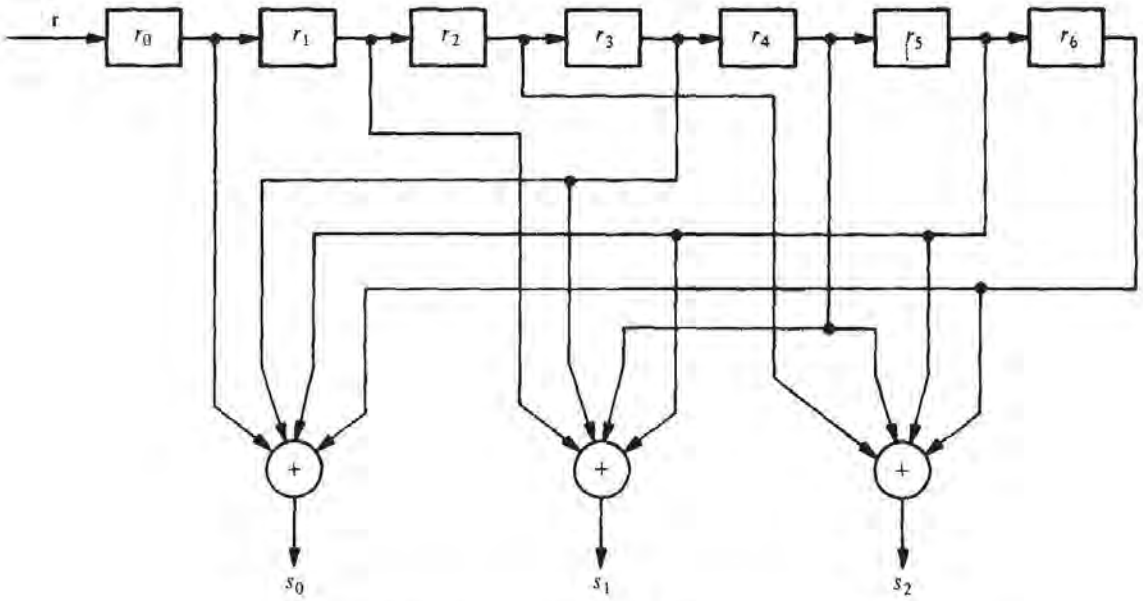


Figure 3.5 Syndrome circuit for the (7, 4) code given in Table 3.1.

The syndrome **s** computed from the received vector **r** actually depends only on the error pattern **e**, and not on the transmitted code word **v**. Since **r** is the vector sum of **v** and **e**, it follows from (3.10) that

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

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

$$\mathbf{s} = \mathbf{e} \cdot \mathbf{H}^T. \tag{3.12}$$

If the parity-check matrix **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:

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

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 feel that any error correction scheme is a method of solving the $n - k$ linear equations of (3.13) for the error digits. Once the error pattern **e** has been found, the vector $\mathbf{r} + \mathbf{e}$ is taken as the actual transmitted code word. Unfortunately, determining the true error vector **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

Ex.1020 / Page 80 of 305
Sandisk Technologies, Inc. et al

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 $\mathbf{v} = (1\ 0\ 0\ 1\ 0\ 1\ 1)$ be the transmitted code word and $\mathbf{r} = (1\ 0\ 0\ 1\ 0\ 0\ 1)$ be the received vector. Upon receiving $\mathbf{r}$, the receiver computes the syndrome:

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

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 syndrome above. 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 equations above. They are

$$(0\ 0\ 0\ 0\ 0\ 1\ 0), \quad (1\ 0\ 1\ 0\ 0\ 1\ 1),$$
$$(1\ 1\ 0\ 1\ 0\ 1\ 0), \quad (0\ 1\ 1\ 1\ 0\ 1\ 1),$$
$$(0\ 1\ 1\ 0\ 1\ 1\ 0), \quad (1\ 1\ 0\ 0\ 1\ 1\ 1),$$
$$(1\ 0\ 1\ 1\ 1\ 1\ 0), \quad (0\ 0\ 0\ 1\ 1\ 1\ 1),$$
$$(1\ 1\ 1\ 0\ 0\ 0\ 0), \quad (0\ 1\ 0\ 0\ 0\ 0\ 1),$$
$$(0\ 0\ 1\ 1\ 0\ 0\ 0), \quad (1\ 0\ 0\ 1\ 0\ 0\ 1),$$
$$(1\ 0\ 0\ 0\ 1\ 0\ 0), \quad (0\ 0\ 1\ 0\ 1\ 0\ 1),$$
$$(0\ 1\ 0\ 1\ 1\ 0\ 0), \quad (1\ 1\ 1\ 1\ 1\ 0\ 1).$$

The error vector $\mathbf{e} = (0\ 0\ 0\ 0\ 0\ 1\ 0)$ has the smallest number of nonzero components. If the channel is a BSC, $\mathbf{e} = (0\ 0\ 0\ 0\ 0\ 1\ 0)$ is the most probable error vector that satisfies the equations above. Taking $\mathbf{e} = (0\ 0\ 0\ 0\ 0\ 1\ 0)$ as the true error vector, the receiver decodes the received vector $\mathbf{r} = (1\ 0\ 0\ 1\ 0\ 0\ 1)$ into the following code word:

$$\begin{aligned}\mathbf{v}^* &= \mathbf{r} + \mathbf{e} \\ &= (1\ 0\ 0\ 1\ 0\ 0\ 1) + (0\ 0\ 0\ 0\ 0\ 1\ 0) \\ &= (1\ 0\ 0\ 1\ 0\ 1\ 1).\end{aligned}$$

We see that $\mathbf{v}^*$ is the actual transmitted code word. Hence, the receiver has made 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 code word 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.

Ex.1020 / Page 81 of 305
Sandisk Technologies, Inc. et al

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 an important parameter of a block code called the *minimum distance* is introduced. This parameter determines the random-error-detecting and random-error-correcting capabilities of a code. Let $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ 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 $\mathbf{v} = (1\ 0\ 0\ 1\ 0\ 1\ 1)$ 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 $\mathbf{v} = (1\ 0\ 0\ 1\ 0\ 1\ 1)$ and $\mathbf{w} = (0\ 1\ 0\ 0\ 0\ 1\ 1)$ 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

$$d(\mathbf{v}, \mathbf{w}) + d(\mathbf{w}, \mathbf{x}) \geq d(\mathbf{v}, \mathbf{x}). \tag{3.14}$$

(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,

$$d(\mathbf{v}, \mathbf{w}) = w(\mathbf{v} + \mathbf{w}). \tag{3.15}$$

For example, the Hamming distance between $\mathbf{v} = (1\ 0\ 0\ 1\ 0\ 1\ 1)$ and $\mathbf{w} = (1\ 1\ 1\ 0\ 0\ 1\ 0)$ is 4 and the weight of $\mathbf{v} + \mathbf{w} = (0\ 1\ 1\ 1\ 0\ 0\ 1)$ is also 4.

Given a block code C, one can compute the Hamming distance between any two distinct code words. The *minimum distance* of C, denoted $d_{\min}$, is defined as

$$d_{\min} = \min \{d(\mathbf{v}, \mathbf{w}) : \mathbf{v}, \mathbf{w} \in C, \mathbf{v} \neq \mathbf{w}\}. \tag{3.16}$$

If C is a linear block code, the sum of two vectors is also a code vector. It follows from (3.15) that the Hamming distance between two code vectors in C is equal to the Hamming weight of a third code vector in C. Then it follows from (3.16) that

$$\begin{aligned} d_{\min} &= \min \{w(\mathbf{v} + \mathbf{w}) : \mathbf{v}, \mathbf{w} \in C, \mathbf{v} \neq \mathbf{w}\} \\ &= \min \{w(\mathbf{x}) : \mathbf{x} \in C, \mathbf{x} \neq \mathbf{0}\} \\ &\triangleq w_{\min}. \end{aligned} \tag{3.17}$$

The parameter $w_{\min} \triangleq \{w(\mathbf{x}) : \mathbf{x} \in C, \mathbf{x} \neq \mathbf{0}\}$ is called the *minimum weight* of the linear code C. Summarizing the result above, 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 code words.

Therefore, for a linear block code, to determine 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

Ex.1020 / Page 82 of 305
Sandisk Technologies, Inc. et al

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 code vector 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 code vector of Hamming weight l in C.

Proof. Let us express the parity-check matrix in the following form:

$$\mathbf{H} = [\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{n-1}],$$

where $\mathbf{h}_i$ represents the ith column of $\mathbf{H}$. Let $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be a code vector of weight l. Then $\mathbf{v}$ has l nonzero components. Let $v_{i_1}, v_{i_2}, \ldots, v_{i_l}$ be the l nonzero components of $\mathbf{v}$, where $0 \leq i_1 < i_2 < \cdots < i_l \leq n-1$. Then $v_{i_1} = v_{i_2} = \cdots = v_{i_l} = 1$. Since $\mathbf{v}$ is a code vector, we must have

$$\begin{aligned}\mathbf{0} &= \mathbf{v} \cdot \mathbf{H}^T \\ &= v_0\mathbf{h}_0 + v_1\mathbf{h}_1 + \cdots + v_{n-1}\mathbf{h}_{n-1} \\ &= v_{i_1}\mathbf{h}_{i_1} + v_{i_2}\mathbf{h}_{i_2} + \cdots + v_{i_l}\mathbf{h}_{i_l} \\ &= \mathbf{h}_{i_1} + \mathbf{h}_{i_2} + \cdots + \mathbf{h}_{i_l}.\end{aligned}$$

This proves the first part of the theorem.

Now suppose that $\mathbf{h}_{i_1}, \mathbf{h}_{i_2}, \ldots, \mathbf{h}_{i_l}$ are l columns of $\mathbf{H}$ such that

$$\mathbf{h}_{i_1} + \mathbf{h}_{i_2} + \cdots + \mathbf{h}_{i_l} = \mathbf{0}. \tag{3.18}$$

Let us form a binary n-tuple $\mathbf{x} = (x_1, x_2, \ldots, x_{n-1})$ whose nonzero components are $x_{i_1}, x_{i_2}, \ldots, x_{i_l}$. The Hamming weight of $\mathbf{x}$ is l. Consider the product

$$\begin{aligned}\mathbf{x} \cdot \mathbf{H}^T &= x_0\mathbf{h}_0 + x_1\mathbf{h}_1 + \cdots + x_{n-1}\mathbf{h}_{n-1} \\ &= x_{i_1}\mathbf{h}_{i_1} + x_{i_2}\mathbf{h}_{i_2} + \cdots + x_{i_l}\mathbf{h}_{i_l} \\ &= \mathbf{h}_{i_1} + \mathbf{h}_{i_2} + \cdots + \mathbf{h}_{i_l}.\end{aligned}$$

It follows from (3.18) that $\mathbf{x} \cdot \mathbf{H}^T = \mathbf{0}$. Thus, $\mathbf{x}$ is a code vector of weight l in C. This proves the second part of the theorem. Q.E.D.

It follows from Theorem 3.2 that we have the following two corollaries.

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

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

Ex.1020 / Page 83 of 305
Sandisk Technologies, Inc. et al

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 code vector $\mathbf{v}$ is transmitted over a noisy channel, an error pattern of l errors will result in a received vector $\mathbf{r}$ which differs from the transmitted vector $\mathbf{v}$ 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 code vectors 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 code vector 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 code word in C. When the receiver detects that the received vector is not a code word 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 code vectors 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 detecting 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 code words. If any of these $2^k - 1$ error patterns occurs, it alters the transmitted code word $\mathbf{v}$ into another code word $\mathbf{w}$. Thus, $\mathbf{w}$ will be received and its syndrome is zero. In this case, the decoder accepts $\mathbf{w}$ as the transmitted code word and thus commits an incorrect decoding. Therefore, there are $2^k - 1$ *undetectable* error patterns. If an error pattern is not identical to a nonzero code word, the received vector $\mathbf{r}$ will not be a code word and the syndrome will not be zero. In this case, error will be detected. There are exactly $2^n - 2^k$ error patterns that are not identical to the code words 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 code vectors of weight i in C. The numbers $A_0, A_1, \ldots, A_n$ are called the *weight distribution* of C. If C is used only for error detection on a BSC, the probability that the decoder fails 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. Since an undetected error occurs only

Ex.1020 / Page 84 of 305
Sandisk Technologies, Inc. et al

when the error pattern is identical to a nonzero code vector of C,

$$P_u(E) = \sum_{i=1}^{n} A_i p^i (1-p)^{n-i}, \tag{3.19}$$

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 code words are transmitted over a BSC with $p = 10^{-2}$, there are on the average seven erroneous code words passing 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 that the code is able to correct. The minimum distance $d_{\min}$ is either odd or even. Let t be a positive integer such that

$$2t + 1 \leq d_{\min} \leq 2t + 2. \tag{3.20}$$

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 code vector and the received vector, respectively. Let $\mathbf{w}$ be any other code vector in C. The Hamming distances among $\mathbf{v}$, $\mathbf{r}$, and $\mathbf{w}$ satisfy the triangle inequality:

$$d(\mathbf{v}, \mathbf{r}) + d(\mathbf{w}, \mathbf{r}) \geq d(\mathbf{v}, \mathbf{w}). \tag{3.21}$$

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 $d(\mathbf{v}, \mathbf{r}) = t'$. Since $\mathbf{v}$ and $\mathbf{w}$ are code vectors in C, we have

$$d(\mathbf{v}, \mathbf{w}) \geq d_{\min} \geq 2t + 1. \tag{3.22}$$

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 $t' \leq t$, then

$$d(\mathbf{w}, \mathbf{r}) > t.$$

The inequality above 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 code vector $\mathbf{v}$ than to any other code vector $\mathbf{w}$ 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 code vector. This results in a correct decoding and thus errors are corrected.

On the other hand, the code is not capable of correcting all the error patterns of l errors with $l > t$, for there is at least one case where an error pattern of l errors results in a received vector which is closer to an incorrect code vector than to the actual transmitted code vector. To show this, let $\mathbf{v}$ and $\mathbf{w}$ be two code vectors in C such that

$$d(\mathbf{v}, \mathbf{w}) = d_{\min}.$$

Ex.1020 / Page 85 of 305
Sandisk Technologies, Inc. et al

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

$$w(\mathbf{e}_1) + w(\mathbf{e}_2) = w(\mathbf{v} + \mathbf{w}) = d(\mathbf{v}, \mathbf{w}) = d_{\min}. \tag{3.23}$$

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

$$d(\mathbf{v}, \mathbf{r}) = w(\mathbf{v} + \mathbf{r}) = w(\mathbf{e}_1). \tag{3.24}$$

The Hamming distance between $\mathbf{w}$ and $\mathbf{r}$ is

$$d(\mathbf{w}, \mathbf{r}) = w(\mathbf{w} + \mathbf{r}) = w(\mathbf{w} + \mathbf{v} + \mathbf{e}_1) = w(\mathbf{e}_2). \tag{3.25}$$

Now suppose that the error pattern $\mathbf{e}_1$ contains more than t errors [i.e., $w(\mathbf{e}_1) > t$]. Since $2t + 1 \leq d_{\min} \leq 2t + 2$, 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 which results in a received vector that is closer to an incorrect code vector than to the transmitted code vector. Based on the maximum likelihood decoding scheme, an incorrect decoding would be committed.

Summarizing the results above, a block code with minimum distance $d_{\min}$ *guarantees* correcting 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 $t = 1$. It 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. For a t-error-correcting (n, k) linear code, it is capable of correcting a total 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

$$P(E) \leq \sum_{i=t+1}^{n} \binom{n}{i} p^i (1 - p)^{n-i}. \tag{3.26}$$

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

Ex.1020 / Page 86 of 305
Sandisk Technologies, Inc. et al

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.

From the discussion above, we see that random-error-detecting and random-error-correcting capabilities of a block code are determined by the code's minimum distance. Clearly, for given n and k, one would like to construct a block code with minimum distance as large as possible, in addition to the implementation considerations.

3.5 STANDARD ARRAY AND SYNDROME DECODING

In this section a scheme for decoding linear block codes is presented. Let C be an (n, k) linear code. Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{2^k}$ be the code vectors of C. No matter which code vector 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 $D_1, D_2, \ldots, D_{2^k}$ such that the code vector $\mathbf{v}_i$ 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 code vector $\mathbf{v}_i$. If the received vector $\mathbf{r}$ is found in the subset D_i, $\mathbf{r}$ is decoded into $\mathbf{v}_i$. Correct decoding is made if and only if the received vector $\mathbf{r}$ is in the subset D_i that corresponds to the actual code vector 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 code vector is described here. The partition is based on the linear structure of the code. First, the 2^k code vectors of C are placed in a row with the all-zero code vector $\mathbf{v}_1 = (0, 0, \ldots, 0)$ as the first (left-most) element. From the remaining $2^n - 2^k$ n-tuples, an n-tuple $\mathbf{e}_2$ is chosen and is placed under the zero vector $\mathbf{v}_1$. Now, we form a second row by adding $\mathbf{e}_2$ to each code vector $\mathbf{v}_i$ in the first row and placing the sum $\mathbf{e}_2 + \mathbf{v}_i$ under $\mathbf{v}_i$. Having completed the second row, an unused n-tuple $\mathbf{e}_3$ is chosen from the remaining n-tuples and is placed under $\mathbf{v}_1$. Then a third row is formed by adding $\mathbf{e}_3$ to each code vector $\mathbf{v}_i$ in the first row and placing $\mathbf{e}_3 + \mathbf{v}_i$ under $\mathbf{v}_i$. We continue this process until all the n-tuples are used. 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 code vector 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 code vectors of C are distinct. Suppose that two n-tuples in the lth rows are identical, say $\mathbf{e}_l + \mathbf{v}_i = \mathbf{e}_l + \mathbf{v}_j$ 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 lth row and the

Ex.1020 / Page 87 of 305
Sandisk Technologies, Inc. et al

$$
\begin{array}{llllll}
\mathbf{v}_1 = 0 & \mathbf{v}_2 & \cdots & \mathbf{v}_i & \cdots & \mathbf{v}_{2^k} \\
\mathbf{e}_2 & \mathbf{e}_2 + \mathbf{v}_2 & \cdots & \mathbf{e}_2 + \mathbf{v}_i & \cdots & \mathbf{e}_2 + \mathbf{v}_{2^k} \\
\mathbf{e}_3 & \mathbf{e}_3 + \mathbf{v}_2 & \cdots & \mathbf{e}_3 + \mathbf{v}_i & \cdots & \mathbf{e}_3 + \mathbf{v}_{2^k} \\
\vdots & & & & & \vdots \\
\mathbf{e}_l & \mathbf{e}_l + \mathbf{v}_2 & \cdots & \mathbf{e}_l + \mathbf{v}_i & \cdots & \mathbf{e}_l + \mathbf{v}_{2^k} \\
\vdots & & & & & \vdots \\
\mathbf{e}_{2^{n-k}} & \mathbf{e}_{2^{n-k}} + \mathbf{v}_2 & \cdots & \mathbf{e}_{2^{n-k}} + \mathbf{v}_i & \cdots & \mathbf{e}_{2^{n-k}} + \mathbf{v}_{2^k}
\end{array}
$$

Figure 3.6 Standard array for an (n, k) linear code.

mth 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 $\mathbf{e}_m = \mathbf{e}_l + (\mathbf{v}_i + \mathbf{v}_j)$. Since $\mathbf{v}_i$ and $\mathbf{v}_j$ are code vectors in C, $\mathbf{v}_i + \mathbf{v}_j$ is also a code vector 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 lth row of the array, which contradicts the construction rule of the array that $\mathbf{e}_m$, the first element of the mth 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*. 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 code vector in C. Let D_j denote the jth column of the standard array. Then

$$
D_j = \{\mathbf{v}_j, \mathbf{e}_2 + \mathbf{v}_j, \mathbf{e}_3 + \mathbf{v}_j, \ldots, \mathbf{e}_{2^{n-k}} + \mathbf{v}_j\}, \tag{3.27}
$$

where $\mathbf{v}_j$ is a code vector of C and $\mathbf{e}_2, \mathbf{e}_3, \ldots, \mathbf{e}_{2^{n-k}}$ are the coset leaders. The 2^k disjoint columns $D_1, D_2, \ldots, D_{2^k}$ can be used for decoding the code C as described earlier in this section. Suppose that the code vector $\mathbf{v}_j$ 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

Ex.1020 / Page 88 of 305
Sandisk Technologies, Inc. et al

Coset leader							
000000	011100	101010	110001	110110	101101	011011	000111
100000	111100	001010	010001	010110	001101	111011	100111
010000	001100	111010	100001	100110	111101	001011	010111
001000	010100	100010	111001	111110	100101	010011	001111
000100	011000	101110	110101	110010	101001	011111	000011
000010	011110	101000	110011	110100	101111	011001	000101
000001	011101	101011	110000	110111	101100	011010	000110
100100	111000	001110	010101	010010	001001	111111	100011

Figure 3.7 Standard array for the (6, 3) code.

correctly into the transmitted code vector $\mathbf{v}_j$. On the other hand, 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 code vector, say in the lth coset and under the code vector $\mathbf{v}_i \neq \mathbf{0}$. 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 code vector. 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 results above, 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. Choosing coset leaders 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 ith column D_i and lth coset of the standard array. Then $\mathbf{r}$ is decoded into the code vector $\mathbf{v}_i$. Since $\mathbf{r} = \mathbf{e}_l + \mathbf{v}_i$, the distance between $\mathbf{r}$ and $\mathbf{v}_i$ is

$$d(\mathbf{r}, \mathbf{v}_i) = w(\mathbf{r} + \mathbf{v}_i) = w(\mathbf{e}_l + \mathbf{v}_i + \mathbf{v}_i) = w(\mathbf{e}_l). \tag{3.28}$$

Now, consider the distance between $\mathbf{r}$ and any other code vector, say $\mathbf{v}_j$,

$$d(\mathbf{r}, \mathbf{v}_j) = w(\mathbf{r} + \mathbf{v}_j) = w(\mathbf{e}_l + \mathbf{v}_i + \mathbf{v}_j).$$

Since $\mathbf{v}_i$ and $\mathbf{v}_j$ are two different code vectors, their vector sum, $\mathbf{v}_i + \mathbf{v}_j$, is a nonzero code vector, say $\mathbf{v}_s$. Thus,

$$d(\mathbf{r}, \mathbf{v}_j) = w(\mathbf{e}_l + \mathbf{v}_s). \tag{3.29}$$

Ex.1020 / Page 89 of 305
Sandisk Technologies, Inc. et al

Since, $\mathbf{e}_l$ 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

$$d(\mathbf{r}, \mathbf{v}_l) \leq d(\mathbf{r}, \mathbf{v}_j).$$

This says that the received vector is decoded into a closest code vector. 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 $\alpha_0, \alpha_1, \ldots, \alpha_n$ are called the *weight distribution* of the coset leaders. Knowing these numbers, we can compute the probability of a decoding error. Since 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

$$P(E) = 1 - \sum_{i=0}^{n} \alpha_i p^i (1-p)^{n-i}. \tag{3.30}$$

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 of $t = \lfloor (d_{\min} - 1)/2 \rfloor$ 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. Since 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 code vector 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 code vector of C [i.e., $w(\mathbf{v}) = d_{\min}$]. Let $\mathbf{x}$ and $\mathbf{y}$ be two n-tuples which 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

$$w(\mathbf{x}) + w(\mathbf{y}) = w(\mathbf{v}) = d_{\min}.$$

Ex.1020 / Page 90 of 305
Sandisk Technologies, Inc. et al

Suppose we choose $\mathbf{y}$ such that $w(\mathbf{y}) = t + 1$. Since $2t + 1 \leq d_{\min} \leq 2t + 2$, 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 code vector $\mathbf{v}_i$ 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 equality above 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 jth and lth cosets, respectively, where $j < l$. Suppose that the syndromes of these two cosets are equal. Then

$$\mathbf{e}_j\mathbf{H}^T = \mathbf{e}_l\mathbf{H}^T,$$

$$(\mathbf{e}_j + \mathbf{e}_l)\mathbf{H}^T = \mathbf{0}.$$

This implies that $\mathbf{e}_j + \mathbf{e}_l$ is a code vector 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 jth 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:

Step 1. Compute the syndrome of $\mathbf{r}$, $\mathbf{r} \cdot \mathbf{H}^T$.

Step 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.

Step 3. Decode the received vector $\mathbf{r}$ into the code vector $\mathbf{v} = \mathbf{r} + \mathbf{e}_l$.

The decoding scheme described above 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,

Ex.1020 / Page 91 of 305
Sandisk Technologies, Inc. et al

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 which are variations of table-lookup decoding are discussed in subsequent chapters. Each of these decoding schemes requires additional properties in 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). Since 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.

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)

Suppose that the code vector $\mathbf{v} = (1\ 0\ 0\ 1\ 0\ 1\ 1)$ is transmitted and $\mathbf{r} = (1\ 0\ 0\ 1\ 1\ 1\ 1)$ is received. For decoding $\mathbf{r}$, we compute the syndrome of $\mathbf{r}$,

$$\mathbf{s} = (1\ 0\ 0\ 1\ 1\ 1\ 1) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = (0\ 1\ 1).$$

From Table 3.2 we find that (0 1 1) is the syndrome of the coset leader $\mathbf{e} = (0\ 0\ 0\ 0\ 1\ 0\ 0)$. Thus, (0 0 0 0 1 0 0) is assumed to be the error pattern caused by the channel, and $\mathbf{r}$ is decoded into

Ex.1020 / Page 92 of 305
Sandisk Technologies, Inc. et al

$$\begin{aligned}\mathbf{v}^* &= \mathbf{r} + \mathbf{e}\\ &= (1\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1) + (0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0)\\ &= (1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1),\end{aligned}$$

which is the actual code vector transmitted. The decoding is correct since the error pattern caused by the channel is a coset leader.

Now suppose that $\mathbf{v} = (0\ 0\ 0\ 0\ 0\ 0\ 0)$ is transmitted and $\mathbf{r} = (1\ 0\ 0\ 0\ 1\ 0\ 0)$ 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 $\mathbf{e} = (0\ 0\ 0\ 0\ 0\ 1\ 0)$ corresponds to the syndrome $\mathbf{s} = (1\ 1\ 1)$. As a result, $\mathbf{r}$ is decoded into the code vector

$$\begin{aligned}\mathbf{v}^* &= \mathbf{r} + \mathbf{e}\\ &= (1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0) + (0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0)\\ &= (1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0).\end{aligned}$$

Since $\mathbf{v}^*$ is not the actual code vector transmitted, a decoding error is committed.

Using Table 3.2, 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:

$$\begin{aligned}e_0 &= f_0(s_0, s_1, \ldots, s_{n-k-1}),\\ e_1 &= f_1(s_0, s_1, \ldots, s_{n-k-1}),\\ &\vdots\\ e_{n-1} &= f_{n-1}(s_0, s_1, \ldots, s_{n-k-1}),\end{aligned}$$

where $s_0, s_1, \ldots, s_{n-k-1}$ are the syndrome digits, which are regarded as switching variables, and $e_0, e_1, \ldots, e_{n-1}$ 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 has been discussed in Section 3.2. The general decoder for an (n, k) linear code based on the table-lookup scheme is shown 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

$$\begin{aligned}e_0 &= s_0 \Lambda s_1' \Lambda s_2', & e_1 &= s_0' \Lambda s_1 \Lambda s_2',\\ e_2 &= s_0' \Lambda s_1' \Lambda s_2, & e_3 &= s_0 \Lambda s_1 \Lambda s_2',\\ e_4 &= s_0' \Lambda s_1 \Lambda s_2, & e_5 &= s_0 \Lambda s_1 \Lambda s_2,\\ e_6 &= s_0 \Lambda s_1' \Lambda s_2,\end{aligned}$$

Ex.1020 / Page 93 of 305
Sandisk Technologies, Inc. et al

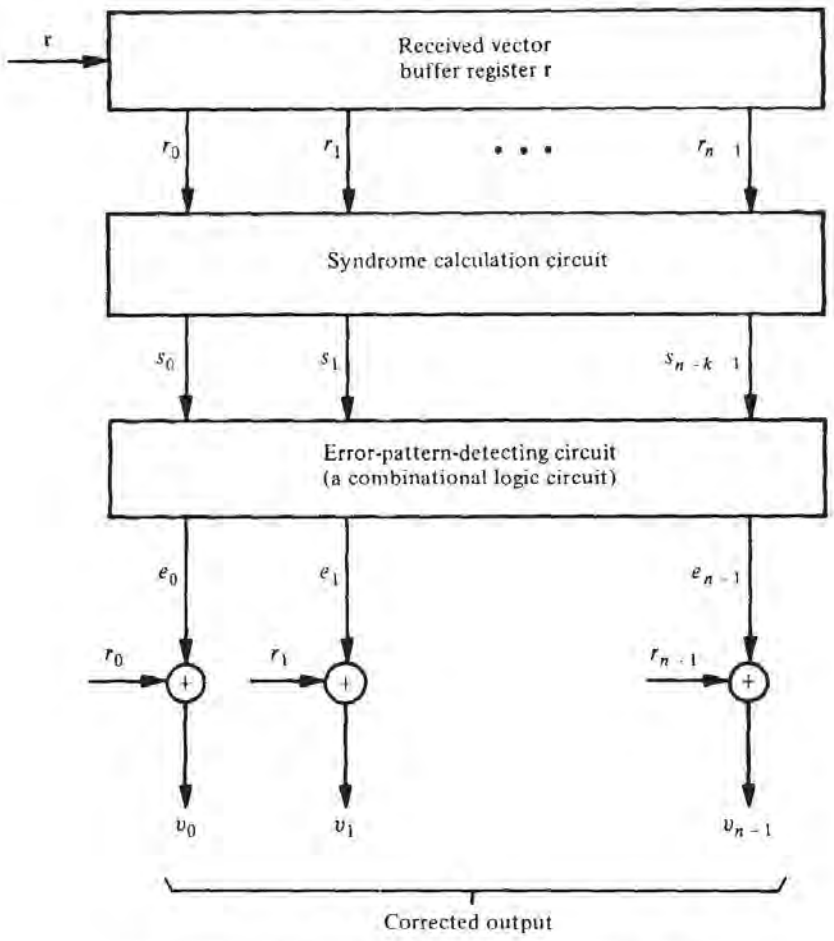


Figure 3.8 General decoder for a linear block code.

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

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.

Ex.1020 / Page 94 of 305
Sandisk Technologies, Inc. et al

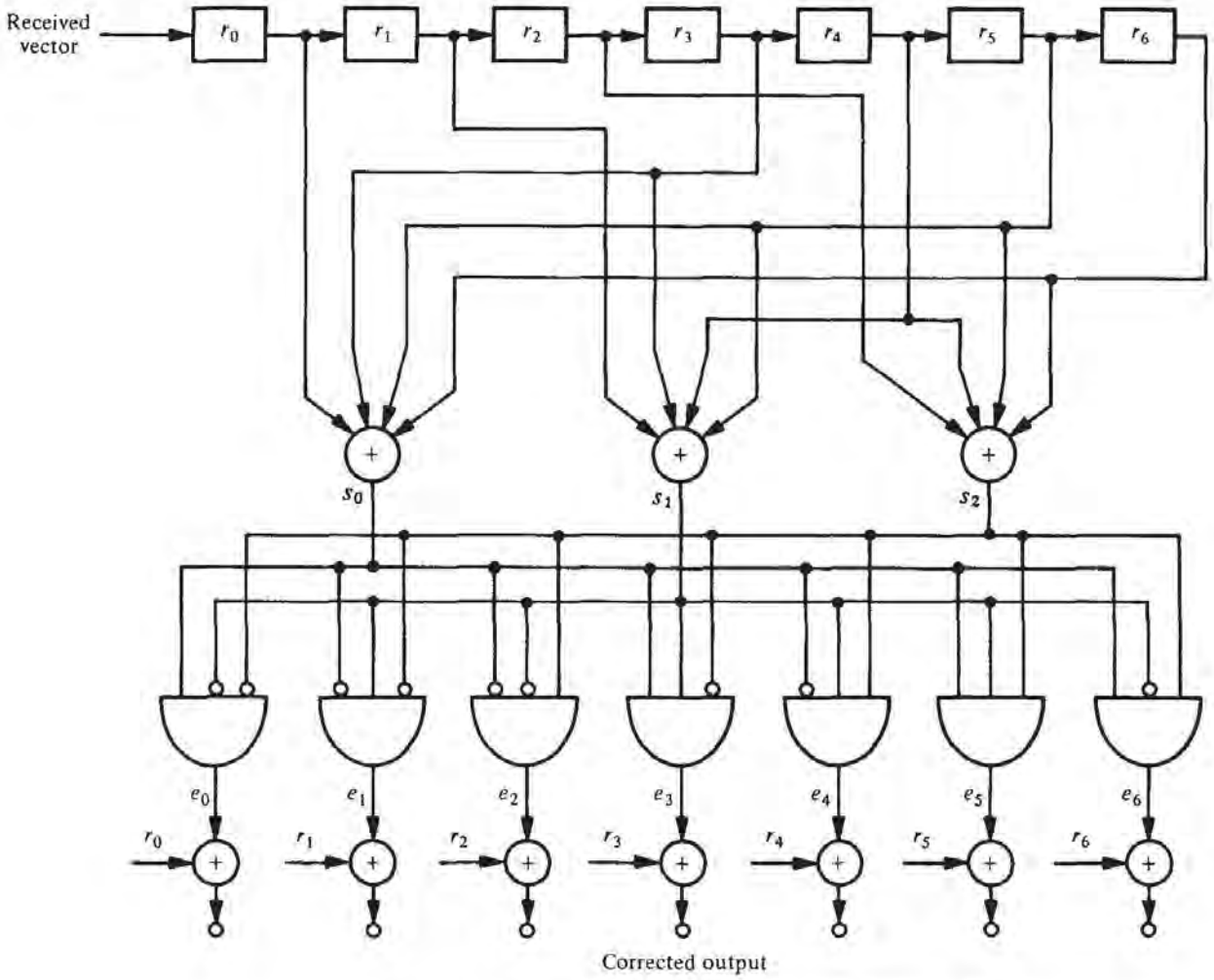


Figure 3.9 Decoding circuit for the (7, 4) code given in Table 3.1.

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 $\{A_0, A_1, \ldots, A_n\}$ be the weight distribution of an (n, k) linear code C and let $\{B_0, B_1, \ldots, B_n\}$ be the weight distribution of its dual code C_d. Now we represent these two weight distributions in polynomial form as follows:

$$\begin{aligned} A(z) &= A_0 + A_1 z + \cdots + A_n z^n, \\ B(z) &= B_0 + B_1 z + \cdots + B_n z^n. \end{aligned} \tag{3.31}$$

Then $A(z)$ and $B(z)$ are related by the following identity:

$$A(z) = 2^{-(n-k)}(1+z)^n B\left(\frac{1-z}{1+z}\right). \tag{3.32}$$

Ex.1020 / Page 95 of 305
Sandisk Technologies, Inc. et al

This identity is known as the *MacWilliams identity* [5]. 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 of 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:

$$\begin{aligned} P_u(E) &= \sum_{i=1}^{n} A_i p^i (1-p)^{n-i} \\ &= (1-p)^n \sum_{i=1}^{n} A_i \left(\frac{p}{1-p}\right)^i. \end{aligned} \tag{3.33}$$

Substituting $z = p/(1-p)$ in $A(z)$ of (3.31) and using the fact that $A_0 = 1$, we obtain the following identity:

$$A\left(\frac{p}{1-p}\right) - 1 = \sum_{i=1}^{n} A_i \left(\frac{p}{1-p}\right)^i. \tag{3.34}$$

Combining (3.33) and (3.34), we have the following expression for the probability of an undetected error:

$$P_u(E) = (1-p)^n \left[A\left(\frac{p}{1-p}\right) - 1 \right]. \tag{3.35}$$

From (3.35) and the MacWilliams identity of (3.32), we finally obtain the following expression for $P_u(E)$:

$$P_u(E) = 2^{-(n-k)} B(1-2p) - (1-p)^n, \tag{3.36}$$

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 **H**, we obtain the following eight vectors in the dual code:

$$\begin{array}{llll} (0\ 0\ 0\ 0\ 0\ 0\ 0), & & (1\ 1\ 0\ 0\ 1\ 0\ 1), \\ (1\ 0\ 0\ 1\ 0\ 1\ 1), & & (1\ 0\ 1\ 1\ 1\ 0\ 0), \\ (0\ 1\ 0\ 1\ 1\ 1\ 0), & & (0\ 1\ 1\ 1\ 0\ 0\ 1), \\ (0\ 0\ 1\ 0\ 1\ 1\ 1), & & (1\ 1\ 1\ 0\ 0\ 1\ 0). \end{array}$$

Ex.1020 / Page 96 of 305
Sandisk Technologies, Inc. et al

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 code words or by examining the 2^{n-k} code words 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 have shown 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. Since 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 of C_j being chosen is

$$P(C_j) = 2^{-k(n-k)}. \tag{3.37}$$

Let A_{jl} denote the number of code words in C_j with weight l. It follows from (3.19) that probability of an undetected error for C_j is given by

$$P_u(E \mid C_j) = \sum_{i=1}^{n} A_{ji} p^i (1 - p)^{n-i}. \tag{3.38}$$

The average probability of an undetected error for a linear code in Γ is defined as

$$\mathbf{P}_u(\mathbf{E}) = \sum_{j=1}^{|\Gamma|} P(C_j) P_u(E \mid C_j), \tag{3.39}$$

where $|\Gamma|$ denotes the number of codes in Γ. Substituting (3.37) and (3.38) into (3.39), we obtain

$$\mathbf{P}_u(\mathbf{E}) = 2^{-k(n-k)} \sum_{i=1}^{n} p^i (1 - p)^{n-i} \sum_{j=1}^{|\Gamma|} A_{ji}. \tag{3.40}$$

A nonzero n-tuple is either contained in exactly $2^{(k-1)(n-k)}$ codes in Γ or contained in none of the codes (left as a problem). Since there are $\binom{n}{i}$ n-tuples of weight i, we have

$$\sum_{j=1}^{|\Gamma|} A_{ji} \leq \binom{n}{i} 2^{(k-1)(n-k)}. \tag{3.41}$$

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:

Ex.1020 / Page 97 of 305
Sandisk Technologies, Inc. et al

$$\mathbf{P}_u(\mathbf{E}) \leq 2^{-(n-k)} \sum_{i=1}^{n} \binom{n}{i} p^i (1-p)^{n-i}$$
$$= 2^{-(n-k)}[1 - (1-p)^n]. \tag{3.42}$$

Since $[1 - (1-p)^n] \leq 1$, it is clear that $\mathbf{P}_u(\mathbf{E}) \leq 2^{-(n-k)}$.

The result above says that there exist (n, k) linear codes with 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 three decades. However, only a few small classes of linear codes have been proved to have $P_u(E)$ satisfying the upper bound $2^{-(n-k)}$. It is still not known whether the other known linear codes satisfy this upper bound. A class of linear codes that satisfies this upper bound is presented in the next section. Other codes with probability of an undetected error decreasing exponentially with $n - k$ are presented in subsequent chapters.

3.7 HAMMING CODES

Hamming codes are the first class of linear codes devised for error correction [6]. These codes and their variations have been widely used for error control in digital communication and data storage systems.

For any positive integer $m \geq 3$, there exists a Hamming code with the following parameters:

Code length:	$n = 2^m - 1$
Number of information symbols:	$k = 2^m - m - 1$
Number of parity-check symbols:	$n - k = m$
Error-correcting capability:	$t = 1 (d_{\min} = 3)$.

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 which 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

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

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

Ex.1020 / Page 98 of 305
Sandisk Technologies, Inc. et al

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 an $(2^m - m - 1) \times (2^m - m - 1)$ identity matrix.

Since 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,

$$\mathbf{h}_i + \mathbf{h}_j + \mathbf{h}_l = \mathbf{0}.$$

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 of 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 $2^m - 1$. Since $n - k = m$, 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. This says that a Hamming code corrects only the 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. Perfect codes are rare [3]. Besides the Hamming codes, the only other nontrivial binary perfect code is the (23, 12) Golay code (see Section 5.3).

Decoding of Hamming codes can be accomplished easily 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:

Code length:	$n = 2^m - l - 1$
Number of information symbols:	$k = 2^m - m - l - 1$
Number of parity-check symbols:	$n - k = m$
Minimum distance:	$d_{\min} \geq 3.$

If we delete columns from $\mathbf{H}$ properly, we may obtain a shortened Hamming code with minimum distance 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. Since 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 exists three columns $\mathbf{h}_j$, $\mathbf{h}_l$, and $\mathbf{h}_s$ in $\mathbf{I}_m$ such that $\mathbf{h}_i + \mathbf{h}_j + \mathbf{h}_l + \mathbf{h}_s = \mathbf{0}$.

Ex.1020 / Page 99 of 305
Sandisk Technologies, Inc. et al

Thus, the shortened Hamming code with $\mathbf{H}'$ as a parity-check matrix has minimum distance 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 even number of 1's. Based on these facts, decoding can be accomplished in the following manner:

1. If the syndrome $\mathbf{s}$ is zero, we assume that no error occurred.
2. If $\mathbf{s}$ is nonzero and it contains 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 even number of 1's, an uncorrectable error pattern has been detected.

A class of single-error-correcting and double-error-detecting shortened Hamming codes which is widely used for error control in computer main/or control storages is presented in Chapter 16.

The weight distribution of a Hamming code of length $n = 2^m - 1$ is known [1–4]. The number of code vectors of weight i, A_i, is simply the coefficient of z^i in the expansion of the following polynomial:

$$A(z) = \frac{1}{n+1}\{(1+z)^n + n(1-z)(1-z^2)^{(n-1)/2}\}. \tag{3.43}$$

This polynomial is the weight enumerator for the Hamming codes.

Example 3.11

Let $m = 3$. Then $n = 2^3 - 1 = 7$ and the weight enumerator for the (7, 4) Hamming code is

$$A(z) = \tfrac{1}{8}\{(1+z)^7 + 7(1-z)(1-z^2)^3\} = 1 + 7z^3 + 7z^4 + z^7.$$

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 code word and $2^m - 1$ code words of weight 2^{m-1}. Thus, its weight enumerator is

$$B(z) = 1 + (2^m - 1)z^{2^{m-1}}. \tag{3.44}$$

The duals of Hamming codes are discussed further in Chapter 7.

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 (3.43) or from (3.36) and (3.44). Computing $P_u(E)$ from (3.36) and (3.44) is easier. Combining (3.36) and (3.44), we obtain

$$P_u(E) = 2^{-m}\{1 + (2^m - 1)(1-2p)^{2^{m-1}}\} - (1-p)^{2^m - 1}. \tag{3.45}$$

Ex.1020 / Page 100 of 305
Sandisk Technologies, Inc. et al

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., $P_u(E) \leq 2^{-m}$] [7]. This can be shown by using the expression of (3.45) (see Problem 3.21).

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 code vectors. Construct a new linear code C_1 with the following parity-check matrix:

$$\mathbf{H}_1 = \begin{bmatrix} 0 & & \\ 0 & & \\ \cdot & & \\ \cdot & \mathbf{H} & \\ \cdot & & \\ 0 & & \\ \hline 1 & 1\ 1\ \cdots\ 1 & \end{bmatrix}.$$

(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 code vector of C_1 has even weight.

(c) Show that C_1 can be obtained from C by adding an extra parity-check digit, denoted v_∞, to the left of each code vector $\mathbf{v}$ 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-weight and odd-weight code vectors. Show that the number of even-weight code vectors is equal to the number of odd-weight code vectors.

3.6. Consider an (n, k) linear code C whose generator matrix $\mathbf{G}$ contains no zero column. Arrange all the code vectors of C as rows of a 2^k-by-n array.

(a) Show that no column of the array contains only zeros.

(b) Show that each column of the array consists of 2^{k-1} zeros and 2^{k-1} ones.

(c) Show that the set of all code vectors with zeros in a particular component 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

$$d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z}) \geq d(\mathbf{x}, \mathbf{z}).$$

Ex.1020 / Page 101 of 305
Sandisk Technologies, Inc. et al

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. Since 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 either contained in exactly $2^{(k-1)(n-k)}$ codes in Γ or contained 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 minimum distance at least $d_1 + d_2$.

3.14. Show that the dual code of the (8, 4) linear code C given in Problem 3.1 is identical to C. C is said to be *self-dual*.

3.15. Form a parity-check matrix for a (15, 11) Hamming code. Devise a decoder for this code.

3.16. 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 inequality above 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* [5]. [*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.]

3.17. Show that the Hamming codes achieve the Hamming bound.

3.18. 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). The bound above is known as the *Plotkin bound* [1–3].)

3.19. Show that there exists an (n, k) linear code with minimum distance at least d if

$$\sum_{i=1}^{d-1} \binom{n}{i} < 2^{n-k}.$$

Ex.1020 / Page 102 of 305
Sandisk Technologies, Inc. et al

[*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.]

3.20. Show that there exists an (n, k) linear code with minimum distance at least $d_{\min}$ which 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 3.19. The second inequality provides a lower bound on the minimum distance attainable with an (n, k) linear code. This bound is known as *Varsharmov–Gilbert* bound [1–3].)

3.21. Show that the probability of an undetected error for Hamming codes on a BSC with transition probability p satisfies the upper bound 2^{-m} for $p \leq \frac{1}{2}$. [*Hint:* Use the inequality $(1-2p) \leq (1-p)^2$.]

3.22. Compute the probability of an undetected error for a (15, 11) Hamming code on a BSC with transition probability $p = 10^{-2}$.

REFERENCES

1. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
2. W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2nd ed., MIT Press, Cambridge, Mass., 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. F. J. MacWilliams, "A Theorem on the Distribution of Weights in a Systematic Code," *Bell Syst. Tech. J.*, 42, pp. 79–94, 1963.
6. R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell Syst. Tech. J.*, 29, pp. 147–160, April 1950.
7. S. K. Leung-Yan-Cheong and M. E. Hellman, "Concerning a Bound on Undetected Error Probability," *IEEE Trans. Inf. Theory*, IT-22, pp. 235–237, March 1976.

Ex.1020 / Page 103 of 305
Sandisk Technologies, Inc. et al

4

Cyclic Codes

Cyclic codes form an important subclass of linear codes. These codes are attractive for two reasons: first, encoding and syndrome computation can be implemented easily by employing shift registers with feedback connections (or linear sequential circuits); and second, because they have considerable inherent algebraic structure, it is possible to find various practical methods for decoding them.

Cyclic codes were first studied by 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. References 2 to 7 contain excellent expositions of cyclic codes.

4.1 DESCRIPTION OF CYCLIC CODES

If the components of an n-tuple $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ are cyclically shifted one place to the right, we obtain another n-tuple,

$$\mathbf{v}^{(1)} = (v_{n-1}, v_0 \ldots, v_{n-2}),$$

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 would be

$$\mathbf{v}^{(i)} = (v_{n-i}, v_{n-i+1}, \ldots, v_{n-1}, v_0, v_1, \ldots, v_{n-i-1}).$$

Clearly, cyclically shifting $\mathbf{v}$ i places to the right is equivalent to cyclically shifting $\mathbf{v}$ $n - i$ places to the left.

Definition 4.1. An (n, k) linear code C is called a *cyclic code* if every cyclic shift of a code vector in C is also a code vector in C.

Ex.1020 / Page 104 of 305
Sandisk Technologies, Inc. et al

The (7, 4) linear code given in Table 4.1 is a cyclic code. Cyclic codes form an important subclass of the linear codes and they possess many algebraic properties that simplify the encoding and the decoding implementations.

TABLE 4.1 A (7, 4) CYCLIC CODE GENERATED BY $g(X) = 1 + X + X^3$

Messages	Code Vectors	Code polynomials
(0 0 0 0)	0 0 0 0 0 0 0	$0 = 0 \cdot g(X)$
(1 0 0 0)	1 1 0 1 0 0 0	$1 + X + X^3 = 1 \cdot g(X)$
(0 1 0 0)	0 1 1 0 1 0 0	$X + X^2 + X^4 = X \cdot g(X)$
(1 1 0 0)	1 0 1 1 1 0 0	$1 + X^2 + X^3 + X^4 = (1 + X) \cdot g(X)$
(0 0 1 0)	0 0 1 1 0 1 0	$X^2 + X^3 + X^5 = X^2 \cdot g(X)$
(1 0 1 0)	1 1 1 0 0 1 0	$1 + X + X^2 + X^5 = (1 + X^2) \cdot g(X)$
(0 1 1 0)	0 1 0 1 1 1 0	$X + X^3 + X^4 + X^5 = (X + X^2) \cdot g(X)$
(1 1 1 0)	1 0 0 0 1 1 0	$1 + X^4 + X^5 = (1 + X + X^2) \cdot g(X)$
(0 0 0 1)	0 0 0 1 1 0 1	$X^3 + X^4 + X^6 = X^3 \cdot g(X)$
(1 0 0 1)	1 1 0 0 1 0 1	$1 + X + X^4 + X^6 = (1 + X^3) \cdot g(X)$
(0 1 0 1)	0 1 1 1 0 0 1	$X + X^2 + X^3 + X^6 = (X + X^3) \cdot g(X)$
(1 1 0 1)	1 0 1 0 0 0 1	$1 + X^2 + X^6 = (1 + X + X^3) \cdot g(X)$
(0 0 1 1)	0 0 1 0 1 1 1	$X^2 + X^4 + X^5 + X^6 = (X^2 + X^3) \cdot g(X)$
(1 0 1 1)	1 1 1 1 1 1 1	$1 + X + X^2 + X^3 + X^4 + X^5 + X^6 = (1 + X^2 + X^3) \cdot g(X)$
(0 1 1 1)	0 1 0 0 0 1 1	$X + X^5 + X^6 = (X + X^2 + X^3) \cdot g(X)$
(1 1 1 1)	1 0 0 1 0 1 1	$1 + X^3 + X^5 + X^6 = (1 + X + X^2 + X^3) \cdot g(X)$

To develop the algebraic properties of a cyclic code, we treat the components of a code vector $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ as the coefficients of a polynomial as follows:

$$\mathbf{v}(X) = v_0 + v_1 X + v_2 X^2 + \cdots + v_{n-1} X^{n-1}.$$

Thus, each code vector 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 vector $\mathbf{v}$ and the polynomial $\mathbf{v}(X)$ is one-to-one. We shall call $\mathbf{v}(X)$ the code polynomial of $\mathbf{v}$. Hereafter, we use the terms "code vector" and "code polynomial" interchangeably. The code polynomial that corresponds to the code vector $\mathbf{v}^{(i)}$ is

$$\mathbf{v}^{(i)}(X) = v_{n-i} + v_{n-i+1} X + \cdots + v_{n-1} X^{i-1} + v_0 X^i + v_1 X^{i+1} + \cdots + v_{n-i-1} X^{n-1}.$$

There exists an interesting algebraic relationship between $\mathbf{v}(X)$ and $\mathbf{v}^{(i)}(X)$. Multiplying $\mathbf{v}(X)$ by X^i, we obtain

$$X^i \mathbf{v}(X) = v_0 X^i + v_1 X^{i+1} + \cdots + v_{n-i-1} X^{n-1} + \cdots + v_{n-1} X^{n+i-1}.$$

The equation above can be manipulated into the following form:

$$\begin{aligned} X^i \mathbf{v}(X) &= v_{n-i} + v_{n-i+1} X + \cdots + v_{n-1} X^{i-1} + v_0 X^i + \cdots + v_{n-i-1} X^{n-1} \\ &\quad + v_{n-i}(X^n + 1) + v_{n-i+1} X(X^n + 1) + \cdots + v_{n-1} X^{i-1}(X^n + 1) \\ &= \mathbf{q}(X)(X^n + 1) + \mathbf{v}^{(i)}(X), \end{aligned} \tag{4.1}$$

Ex.1020 / Page 105 of 305
Sandisk Technologies, Inc. et al

where $\mathbf{q}(X) = v_{n-i} + v_{n-i+1}X + \cdots + v_{n-1}X^{i-1}$. From (4.1) 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 which make possible the simple implementation of encoding and syndrome computation.

Theorem 4.1. 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 $\mathbf{g}'(X) = g'_0 + g'_1X + \cdots + g'_{r-1}X^{r-1} + X^r$. Since 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 also a code polynomial which has 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 with 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 4.2. 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, $g_1 + g_2X + \cdots + g_{r-1}X^{r-2} + X^{r-1}$, which has a 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 4.2 that the nonzero code polynomial of minimum degree in an (n, k) cyclic code C is of the following form:

$$\mathbf{g}(X) = 1 + g_1X + g_2X^2 + \cdots + g_{r-1}X^{r-1} + X^r. \tag{4.2}$$

Consider the (7, 4) cyclic code given in Table 4.1. The nonzero code polynomial with minimum degree is $\mathbf{g}(X) = 1 + X + X^3$.

Consider the polynomials $X\mathbf{g}(X), X^2\mathbf{g}(X), \ldots, X^{n-r-1}\mathbf{g}(X)$, which have degrees $r + 1, r + 2, \ldots, n - 1$, respectively. It follows from (4.1) that $X\mathbf{g}(X) = \mathbf{g}^{(1)}(X)$, $X^2\mathbf{g}(X) = \mathbf{g}^{(2)}(X), \ldots, X^{n-r-1}\mathbf{g}(X) = \mathbf{g}^{(n-r-1)}(X)$; 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 $\mathbf{g}(X), X\mathbf{g}(X), \ldots, X^{n-r-1}\mathbf{g}(X)$,

$$\begin{aligned}\mathbf{v}(X) &= u_0\mathbf{g}(X) + u_1X\mathbf{g}(X) + \cdots + u_{n-r-1}X^{n-r-1}\mathbf{g}(X) \\ &= (u_0 + u_1X + \cdots + u_{n-r-1}X^{n-r-1})\mathbf{g}(X),\end{aligned} \tag{4.3}$$

Ex.1020 / Page 106 of 305
Sandisk Technologies, Inc. et al

is also a code polynomial where $u_i = 0$ or 1. The following theorem characterizes an important property of a cyclic code.

Theorem 4.3. 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

$$\begin{aligned}\mathbf{v}(X) &= (a_0 + a_1X + \cdots + a_{n-r-1}X^{n-r-1})\mathbf{g}(X) \\ &= a_0\mathbf{g}(X) + a_1X\mathbf{g}(X) + \cdots + a_{n-r-1}X^{n-r-1}\mathbf{g}(X).\end{aligned}$$

Since $\mathbf{v}(X)$ is a linear combination of the code polynomials, $\mathbf{g}(X)$, $X\mathbf{g}(X)$, . . ., $X^{n-r-1}\mathbf{g}(X)$, 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 equation above, 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. Since 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 4.3 that these polynomials form all the code polynomials of the (n, k) cyclic code C. Since 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:

$$\mathbf{g}(X) = 1 + g_1X + g_2X^2 + \cdots + g_{n-k-1}X^{n-k-1} + X^{n-k}.$$

Summarizing the results above, we have the following theorem:

Theorem 4.4. In an (n, k) cyclic code, there exists one and only one code polynomial of degree $n - k$,

$$\mathbf{g}(X) = 1 + g_1X + g_2X^2 + \cdots + g_{n-k-1}X^{n-k-1} + X^{n-k}. \tag{4.4}$$

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.

Ex.1020 / Page 107 of 305
Sandisk Technologies, Inc. et al

It follows from Theorem 4.4 that every code polynomial $\mathbf{v}(X)$ in an (n, k) cyclic code can be expressed in the following form:

$$\begin{aligned}\mathbf{v}(X) &= \mathbf{u}(X)\mathbf{g}(X) \\ &= (u_0 + u_1X + \cdots + u_{k-1}X^{k-1})\mathbf{g}(X).\end{aligned}$$

If the coefficients of $\mathbf{u}(X)$, $u_0, u_1, \ldots, u_{k-1}$, 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 (4.4). 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 4.5. 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

$$X^k\mathbf{g}(X) = (X^n + 1) + \mathbf{g}^{(k)}(X), \tag{4.5}$$

where $\mathbf{g}^{(k)}(X)$ is the remainder. It follows from (4.1) 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 (4.5) 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 is answered by the following theorem.

Theorem 4.6. 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 $\mathbf{g}(X)$, $X\mathbf{g}(X)$, . . ., $X^{k-1}\mathbf{g}(X)$, which all have degree $n - 1$ or less. A linear combination of these k polynomials,

$$\begin{aligned}\mathbf{v}(X) &= a_0\mathbf{g}(X) + a_1X\mathbf{g}(X) + \cdots + a_{k-1}X^{k-1}\mathbf{g}(X) \\ &= (a_0 + a_1X + \cdots + a_{k-1}X^{k-1})\mathbf{g}(X),\end{aligned}$$

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 $\mathbf{v}(X) = v_0 + v_1X + \cdots + v_{n-1}X^{n-1}$ be a code polynomial in this code. Multiplying $\mathbf{v}(X)$ by X, we obtain

Ex.1020 / Page 108 of 305
Sandisk Technologies, Inc. et al

$$\begin{aligned} X\mathbf{v}(X) &= v_0X + v_1X^2 + \cdots + v_{n-2}X^{n-1} + v_{n-1}X^n \\ &= v_{n-1}(X^n + 1) + (v_{n-1} + v_0X + \cdots + v_{n-2}X^{n-1}) \\ &= v_{n-1}(X^n + 1) + \mathbf{v}^{(1)}(X), \end{aligned}$$

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 $\mathbf{g}(X)$, $X\mathbf{g}(X)$, . . ., $X^{k-1}\mathbf{g}(X)$. Hence, $\mathbf{v}^{(1)}(X)$ is also a code polynomial. It follows from Definition 4.1 that the linear code generated by $\mathbf{g}(X)$, $X\mathbf{g}(X)$, . . ., $X^{k-1}\mathbf{g}(X)$ is an (n, k) cyclic code. Q.E.D.

Theorem 4.6 actually 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. For the past two decades, 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 4.1

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; each generates a (7, 4) cyclic code. The (7, 4) cyclic code given by Table 4.1 is generated by $\mathbf{g}(X) = 1 + X + X^3$. This code has minimum distance 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 $\mathbf{g}(X)$ results in 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 code vector (1 1 1 0 0 1 0).

Given the generator polynomials $\mathbf{g}(X)$ of an (n, k) cyclic code, the code can be put into systematic form (i.e., the rightmost k digits of each code vector are the unaltered information digits and the leftmost $n - k$ digits are parity-check digits). Suppose that the message to be encoded is $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$. The corresponding message polynomial is

$$\mathbf{u}(X) = u_0 + u_1X + \cdots + u_{k-1}X^{k-1}.$$

Multiplying $\mathbf{u}(X)$ by X^{n-k}, we obtain a polynomial of degree $n - 1$ or less,

$$X^{n-k}\mathbf{u}(X) = u_0X^{n-k} + u_1X^{n-k+1} + \cdots + u_{k-1}X^{n-1}.$$

Dividing $X^{n-k}\mathbf{u}(X)$ by the generator polynomial $\mathbf{g}(X)$, we have

$$X^{n-k}\mathbf{u}(X) = \mathbf{a}(X)\mathbf{g}(X) + \mathbf{b}(X) \tag{4.6}$$

where $\mathbf{a}(X)$ and $\mathbf{b}(X)$ are the quotient and the remainder, respectively. Since the degree

Ex.1020 / Page 109 of 305
Sandisk Technologies, Inc. et al

of $\mathbf{g}(X)$ is $n - k$, the degree of $\mathbf{b}(X)$ must be $n - k - 1$ or less, that is,

$$\mathbf{b}(X) = b_0 + b_1 X + \cdots + b_{n-k-1} X^{n-k-1}.$$

Rearranging (4.6), we obtain the following polynomial of degree $n - 1$ or less:

$$\mathbf{b}(X) + X^{n-k}\mathbf{u}(X) = \mathbf{a}(X)\mathbf{g}(X). \tag{4.7}$$

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

$$\begin{aligned}\mathbf{b}(X) + X^{n-k}\mathbf{u}(X) = b_0 + b_1 X + \cdots + b_{n-k-1} X^{n-k-1} \\ + u_0 X^{n-k} + u_1 X^{n-k+1} + \cdots + u_{k-1} X^{n-1},\end{aligned} \tag{4.8}$$

which corresponds to the code vector

$$(b_0, b_1, \ldots, b_{n-k-1}, u_0, u_1, \ldots, u_{k-1}).$$

We see that the code vector consists of k unaltered information digits $(u_0, u_1, \ldots, u_{k-1})$ 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 process above 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 $1, X, \ldots, X^{n-k-1}$, are taken as parity-check digits and the last k symbols, the coefficients of $X^{n-k}, X^{n-k+1}, \ldots, X^{n-1}$, 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 4.2

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 $\mathbf{g}(X)$,

$$\begin{array}{r} X^3 + X \quad \text{(quotient)} \\ X^3 + X + 1 \,\overline{)\, X^6 \qquad\quad + X^3} \\ \underline{X^6 \quad + X^4 + X^3} \\ X^4 \\ \underline{X^4 \qquad + X^2 + X} \\ X^2 + X \quad \text{(remainder)}, \end{array}$$

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 code vector is $\mathbf{v} = (0\ 1\ 1\ 1\ 0\ 0\ 1)$, where the four rightmost digits are the information digits. The 16 code vectors in systematic form are listed in Table 4.2.

Ex.1020 / Page 110 of 305
Sandisk Technologies, Inc. et al

TABLE 4.2 A (7, 4) CYCLIC CODE GENERATED BY $\mathbf{g}(X) = 1 + X + X^3$

Message	Code word	
(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)	$1 + X + X^2 + X^3 + X^4 + X^5 + X^6 = (1 + X^2 + X^5)\mathbf{g}(X)$

4.2 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 4.1 we have shown that the k code polynomials $\mathbf{g}(X)$, $X\mathbf{g}(X)$, $\ldots$, $X^{k-1}\mathbf{g}(X)$ span C. If the k n-tuples corresponding to these k code polynomials are used as the rows of an $k \times n$ matrix, we obtain the following generator matrix for C:

$$\mathbf{G} = \begin{bmatrix} g_0 & g_1 & g_2 & \cdots & g_{n-k} & 0 & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & g_2 & \cdots & g_{n-k} & 0 & 0 & \cdots & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & \cdots & g_{n-k} & 0 & \cdots & 0 \\ \vdots & & & & & & & & & \vdots \\ 0 & 0 & \cdots & 0 & g_0 & g_1 & g_2 & \cdots & & g_{n-k} \end{bmatrix} \tag{4.9}$$

(Note that $g_0 = g_{n-k} = 1$.) In general, $\mathbf{G}$ is not in systematic form. However, it can be put into systematic form with row operations. For example, the (7, 4) cyclic code given in Table 4.1 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 the first row is added to the third row and the sum of the first two rows is added to the fourth row, we obtain the following matrix:

Ex.1020 / Page 111 of 305
Sandisk Technologies, Inc. et al

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

which is in systematic form. This matrix generates the same code as **G**.

Recall that the generator polynomial $\mathbf{g}(X)$ is a factor of $X^n + 1$, say

$$X^n + 1 = \mathbf{g}(X)\mathbf{h}(X), \tag{4.10}$$

where the polynomial $\mathbf{h}(X)$ has the 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 $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be a code vector in C. Then $\mathbf{v}(X) = \mathbf{a}(X)\mathbf{g}(X)$. Multiplying $\mathbf{v}(X)$ by $\mathbf{h}(X)$, we obtain

$$\begin{aligned} \mathbf{v}(X)\mathbf{h}(X) &= \mathbf{a}(X)\mathbf{g}(X)\mathbf{h}(X) \\ &= \mathbf{a}(X)(X^n + 1) \\ &= \mathbf{a}(X) + X^n\mathbf{a}(X). \end{aligned} \tag{4.11}$$

Since the degree of $\mathbf{a}(X)$ is $k - 1$ or less, the powers $X^k, X^{k+1}, \ldots, X^{n-1}$ 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 (4.11), the coefficients of $X^k, X^{k+1}, \ldots, X^{n-1}$ must be equal to zero. Therefore, we obtain the following $n - k$ equalities:

$$\sum_{i=0}^{k} h_i v_{n-i-j} = 0 \qquad \text{for } 1 \le j \le n - k. \tag{4.12}$$

Now, we take the *reciprocal* of $\mathbf{h}(X)$, which is defined as follows:

$$X^k\mathbf{h}(X^{-1}) = h_k + h_{k-1}X + h_{k-2}X^2 + \cdots + h_0 X^k. \tag{4.13}$$

We can see easily 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:

$$\mathbf{H} = \begin{bmatrix} h_k & h_{k-1} & h_{k-2} & \cdots & h_0 & 0 & \cdots & 0 \\ 0 & h_k & h_{k-1} & \cdots & h_1 & h_0 & \cdots & 0 \\ 0 & 0 & h_k & \cdots & h_2 & h_1 & \cdots & 0 \\ \vdots & & & & & & & \vdots \\ 0 & 0 & \cdots & 0 & h_k & h_{k-1} & \cdots & h_0 \end{bmatrix}. \tag{4.14}$$

It follows from the $n - k$ equalities of (4.12) that any code vector $\mathbf{v}$ in C is orthogonal to every row of **H**. Therefore, **H** is a parity-check matrix of the cyclic code C, and the row space of **H** is the dual code of C. Since the parity-check matrix **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.

Ex.1020 / Page 112 of 305
Sandisk Technologies, Inc. et al

Theorem 4.7. 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 4.3

Consider the (7, 4) cyclic code given in Table 4.1 with generator polynomial $\mathbf{g}(X) = 1 + X + X^3$. The parity polynomial is

$$\mathbf{h}(X) = \frac{X^7 + 1}{\mathbf{g}(X)}$$
$$= 1 + X + X^2 + X^4.$$

The reciprocal of $\mathbf{h}(X)$ is

$$X^4\mathbf{h}(X^{-1}) = X^4(1 + X^{-1} + X^{-2} + X^{-4}).$$
$$= 1 + X^2 + X^3 + X^4.$$

The 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 vectors 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 minimum distance 4. Hence, it is capable of correcting any single error and simultaneously detecting any combination of double errors.

The generator matrix in systematic form can also be formed easily. Dividing X^{n-k+i} by the generator polynomial $\mathbf{g}(X)$ for $i = 0, 1, \ldots, k-1$, we obtain

$$X^{n-k+i} = \mathbf{a}_i(X)\mathbf{g}(X) + \mathbf{b}_i(X), \tag{4.15}$$

where $\mathbf{b}_i(X)$ is the remainder with the following form:

$$\mathbf{b}_i(X) = b_{i0} + b_{i1}X + \cdots + b_{i,n-k-1}X^{n-k-1}.$$

Since $\mathbf{b}_i(X) + X^{n-k+i}$ for $i = 0, 1, \ldots, k-1$ 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

$$\mathbf{G} = \begin{bmatrix} b_{00} & b_{01} & b_{02} & \cdots & b_{0,n-k-1} & 1 & 0 & 0 & \cdots & 0 \\ b_{10} & b_{11} & b_{12} & \cdots & b_{1,n-k-1} & 0 & 1 & 0 & \cdots & 0 \\ b_{20} & b_{21} & b_{22} & \cdots & b_{2,n-k-1} & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots \\ b_{k-1,0} & b_{k-1,1} & b_{k-1,2} & \cdots & b_{k-1,n-k-1} & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}, \tag{4.16}$$

which is the generator matrix of C in systematic form. The corresponding parity-check matrix for C is

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & b_{00} & b_{10} & b_{20} & \cdots & b_{k-1,0} \\ 0 & 1 & 0 & \cdots & 0 & b_{01} & b_{11} & b_{21} & \cdots & b_{k-1,1} \\ 0 & 0 & 1 & \cdots & 0 & b_{02} & b_{12} & b_{22} & \cdots & b_{k-1,2} \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 & b_{0,n-k-1} & b_{1,n-k-1} & b_{2,n-k-1} & \cdots & b_{k-1,n-k-1} \end{bmatrix}. \tag{4.17}$$

Ex.1020 / Page 113 of 305
Sandisk Technologies, Inc. et al

Example 4.4

Again, let us 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

$$X^3 = \mathbf{g}(X) + (1 + X),$$
$$X^4 = X\mathbf{g}(X) + (X + X^2),$$
$$X^5 = (X^2 + 1)\mathbf{g}(X) + (1 + X + X^2),$$
$$X^6 = (X^3 + X + 1)\mathbf{g}(X) + (1 + X^2).$$

Rearranging the equations above, we obtain the following four code polynomials:

$$\mathbf{v}_0(X) = 1 + X + X^3,$$
$$\mathbf{v}_1(X) = X + X^2 + X^4,$$
$$\mathbf{v}_2(X) = 1 + X + X^2 + X^5,$$
$$\mathbf{v}_3(X) = 1 + X^2 + X^6.$$

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 $\mathbf{G}'$ obtain earlier in this section.

4.3 ENCODING OF CYCLIC CODES

We have shown in Section 4.1 that encoding of an (n, k) cyclic code in systematic form consists of three steps; (1) multiply the message polynomial $\mathbf{u}(X)$ by X^{n-k}; (2) divide $X^{n-k}\mathbf{u}(X)$ by $\mathbf{g}(X)$ to obtain the remainder $\mathbf{b}(X)$; and (3) form the code word $\mathbf{b}(X) + X^{n-k}\mathbf{u}(X)$. All these three steps can be accomplished with a division circuit which 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 Fig. 4.1. The encoding operation is carried out as follows:

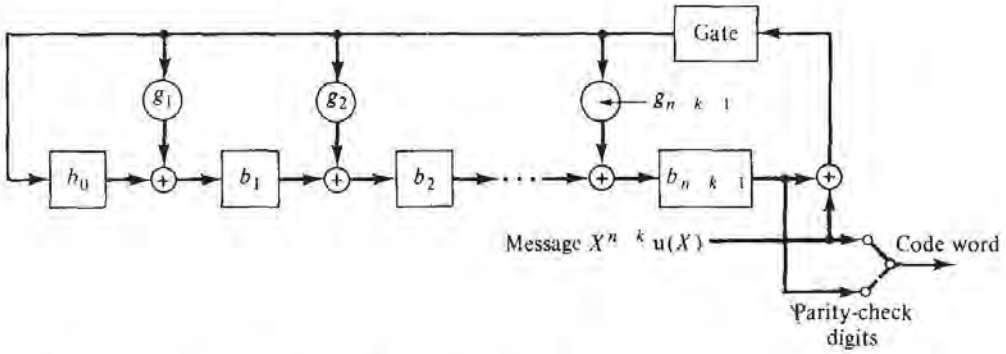


Figure 4.1 Encoding circuit for an (n, k) cyclic code with generator polynomial $\mathbf{g}(X) = 1 + g_1X + g_2X^2 + \cdots + g_{n-k-1}X^{n-k-1} + X^{n-k}$.

Ex.1020 / Page 114 of 305
Sandisk Technologies, Inc. et al

Step 1. With the gate turned on, the k information digits $u_0, u_1, \ldots, u_{k-1}$ [or $\mathbf{u}(X) = u_0 + u_1X + \cdots + u_{k-1}X^{k-1}$ 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 $b_0, b_1, \ldots, b_{n-k-1}$, together with the k information digits, form a complete code vector.

Example 4.5

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 4.2. Suppose that the message $\mathbf{u} = (1\ 0\ 1\ 1)$ is to be encoded. As the message digits are shifted into the register, the contents in the register are 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 (1 0 0). Thus, the complete code vector is (1 0 0 1 0 1 1) and the code polynomial is $1 + X^3 + X^5 + X^6$.

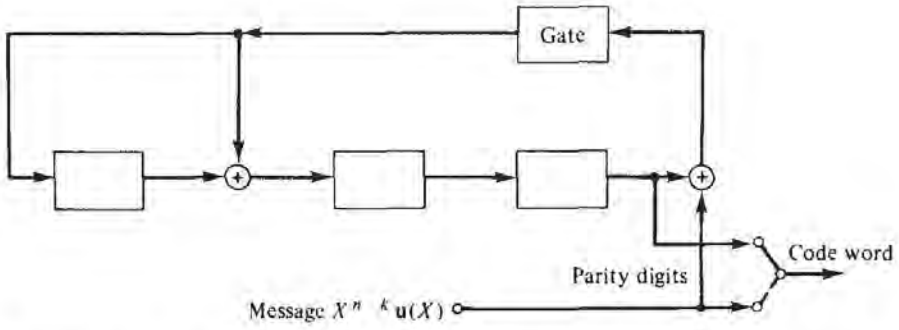


Figure 4.2 Encoder for the (7, 4) cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$.

Encoding of a cyclic code can also be accomplished by using its parity polynomial $\mathbf{h}(X) = h_0 + h_1X + \cdots + h_kX^k$. Let $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be a code vector. We have shown in Section 4.2 that the components of $\mathbf{v}$ satisfy the $n - k$ equalities of (4.12). Since $h_k = 1$, the equalities of (4.12) can be put into the following form:

$$v_{n-k-j} = \sum_{i=0}^{k-1} h_i v_{n-i-j} \qquad \text{for } 1 \leq j \leq n-k \tag{4.18}$$

Ex.1020 / Page 115 of 305
Sandisk Technologies, Inc. et al

which is known as a *difference equation*. For a cyclic code in systematic form, the components $v_{n-k}, v_{n-k+1}, \ldots, v_{n-1}$ of each code vector are the information digits. Given these k information digits, (4.18) is a rule to determine the $n - k$ parity-check digits, $v_0, v_1, \ldots, v_{n-k-1}$. An encoding circuit based on (4.18) is shown in Figure 4.3.

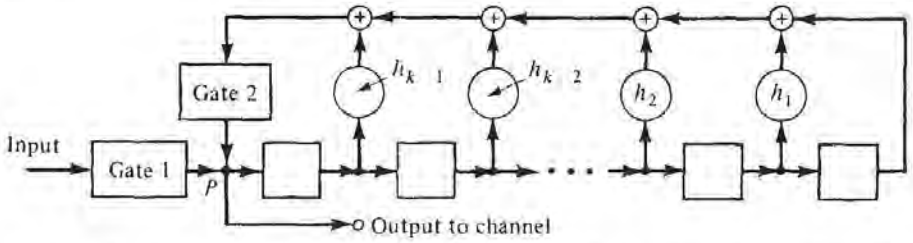


Figure 4.3 Encoding circuit for an (n, k) cyclic code based on the parity polynomial $\mathbf{h}(X) = 1 + h_1X + \cdots + X^k$.

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 $\mathbf{u}(X) = u_0 + u_1X + \cdots + u_{k-1}X^{k-1}$ 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,

$$\begin{aligned} v_{n-k-1} &= h_0v_{n-1} + h_1v_{n-2} + \cdots + h_{k-1}v_{n-k} \\ &= u_{k-1} + h_1u_{k-2} + \cdots + h_{k-1}u_0, \end{aligned}$$

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,

$$\begin{aligned} v_{n-k-2} &= h_0v_{n-2} + h_1v_{n-3} + \cdots + h_{k-1}v_{n-k-1} \\ &= u_{k-2} + h_1u_{k-3} + \cdots + h_{k-2}u_0 + h_{k-1}v_{n-k-1}, \end{aligned}$$

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 encoding circuit above 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 the message digits, the k-stage encoding circuit is more economical; otherwise, the $(n - k)$-stage encoding circuit is preferable.

Ex.1020 / Page 116 of 305
Sandisk Technologies, Inc. et al

Example 4.6

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 4.4. Each code vector 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

$$\begin{aligned} v_{3-j} &= 1 \cdot v_{7-j} + 1 \cdot v_{6-j} + 1 \cdot v_{5-j} + 0 \cdot v_{4-j} \\ &= v_{7-j} + v_{6-j} + v_{5-j} \qquad \text{for } 1 \leq j \leq 3. \end{aligned}$$

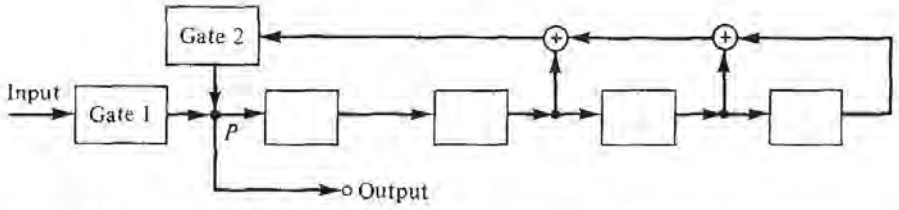


Figure 4.4 Encoding circuit for the (7, 4) cyclic code based on its parity polynomial $\mathbf{h}(X) = 1 + X + X^2 + X^4$.

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

$$v_1 = v_5 + v_4 + v_3 = 1 + 0 + 1 = 0.$$

The third parity-check digit is

$$v_0 = v_4 + v_3 + v_2 = 0 + 1 + 0 = 1.$$

Thus, the code vector that corresponds to the message (1 0 1 1) is (1 0 0 1 0 1 1).

4.4 SYNDROME COMPUTATION AND ERROR DETECTION

Suppose that a code vector is transmitted. Let $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$ be the received vector. Because of the channel noise, the received vector may not be the same as the transmitted code vector. 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 code vector and the decoder accepts $\mathbf{r}$ as the transmitted code vector. If the syndrome is not identical to zero, $\mathbf{r}$ is not a code vector 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 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,

$$\mathbf{r}(X) = r_0 + r_1 X + r_2 X^2 + \cdots + r_{n-1} X^{n-1}.$$

Ex.1020 / Page 117 of 305
Sandisk Technologies, Inc. et al

Dividing $\mathbf{r}(X)$ by the generator polynomial $\mathbf{g}(X)$, we obtain

$$\mathbf{r}(X) = \mathbf{a}(X)\mathbf{g}(X) + \mathbf{s}(X). \tag{4.19}$$

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 4.4 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 computation can be accomplished with a division circuit as shown in Fig. 4.5, 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 $\mathbf{s}(X)$.

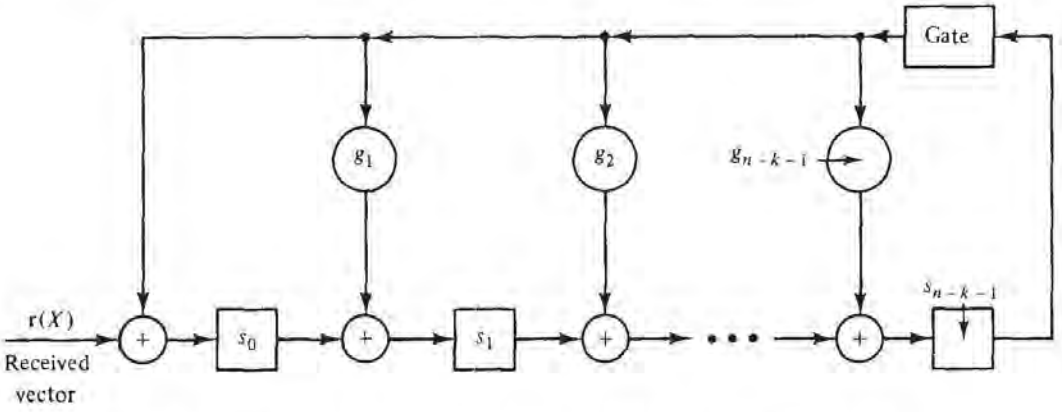


Figure 4.5 An $(n - k)$-stage syndrome circuit with input from the left end.

Because of the cyclic structure of the code, the syndrome $\mathbf{s}(X)$ has the following property.

Theorem 4.8. Let $\mathbf{s}(X)$ be the syndrome of a received polynomial $\mathbf{r}(X) = r_0 + r_1X + \cdots + r_{n-1}X^{n-1}$. Then the remainder $\mathbf{s}^{(1)}(X)$ resulting from dividing $X\mathbf{s}(X)$ by the generator polynomial $\mathbf{g}(X)$ is the syndrome of $\mathbf{r}^{(1)}(X)$, which is a cyclic shift of $\mathbf{r}(X)$.

Proof. It follows from (4.1) that $\mathbf{r}(X)$ and $\mathbf{r}^{(1)}(X)$ satisfy the following relationship:

$$X\mathbf{r}(X) = r_{n-1}(X^n + 1) + \mathbf{r}^{(1)}(X). \tag{4.20}$$

Rearranging (4.20), we have

$$\mathbf{r}^{(1)}(X) = r_{n-1}(X^n + 1) + X\mathbf{r}(X). \tag{4.21}$$

Dividing both sides of (4.21) by $\mathbf{g}(X)$ and using the fact that $X^n + 1 = \mathbf{g}(X)\mathbf{h}(X)$, we obtain

$$\mathbf{c}(X)\mathbf{g}(X) + \boldsymbol{\rho}(X) = r_{n-1}\mathbf{g}(X)\mathbf{h}(X) + X[\mathbf{a}(X)\mathbf{g}(X) + \mathbf{s}(X)], \tag{4.22}$$

where $\boldsymbol{\rho}(X)$ is the remainder resulting from dividing $\mathbf{r}^{(1)}(X)$ by $\mathbf{g}(X)$. Then $\boldsymbol{\rho}(X)$ is the syndrome of $\mathbf{r}^{(1)}(X)$.

Ex.1020 / Page 118 of 305
Sandisk Technologies, Inc. et al

Rearranging (4.22), we obtain the following relationship between $\boldsymbol{\rho}(X)$ and $X\mathbf{s}(X)$:

$$X\mathbf{s}(X) = [\mathbf{c}(X) + r_{n-1}\mathbf{h}(X) + X\mathbf{a}(X)]\mathbf{g}(X) + \boldsymbol{\rho}(X). \tag{4.23}$$

From (4.23) we see that $\boldsymbol{\rho}(X)$ is also the remainder resulting from dividing $X\mathbf{s}(X)$ by $\mathbf{g}(X)$. Therefore, $\boldsymbol{\rho}(X) = \mathbf{s}^{(1)}(X)$. This completes the proof. Q.E.D.

It follows from Theorem 4.8 that the remainder $\mathbf{s}^{(i)}(X)$ resulting from dividing $X^i\mathbf{s}(X)$ by the generator polynomial $\mathbf{g}(X)$ is the syndrome of $\mathbf{r}^{(i)}(X)$, which is the ith cyclic shift of $\mathbf{r}(X)$. This property is useful in decoding of cyclic codes. The syndrome $\mathbf{s}^{(1)}(X)$ of $\mathbf{r}^{(1)}(X)$ can be obtained by shifting (or clocking) the syndrome register once with $\mathbf{s}(X)$ as the initial contents and with the input gate disabled. This is due to the fact that shifting the syndrome register once with $\mathbf{s}(X)$ as the initial contents is equivalent to dividing $X\mathbf{s}(X)$ by $\mathbf{g}(X)$. Thus, after the shift, the register contains $\mathbf{s}^{(1)}(X)$. To obtain the syndrome $\mathbf{s}^{(i)}(X)$ of $\mathbf{r}^{(i)}(X)$, we simply shift the syndrome register i times with $\mathbf{s}(X)$ as the initial contents.

Example 4.7

A syndrome circuit for the (7, 4) cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$ is shown in Figure 4.6. Suppose that the received vector is $\mathbf{r} = (0\ 0\ 1\ 0\ 1\ 1\ 0)$. The syndrome of $\mathbf{r}$ is $\mathbf{s} = (1\ 0\ 1)$. As the received vector is shifted into the circuit, the contents in the register are given in Table 4.3. At the end of the seventh shift, the reg-

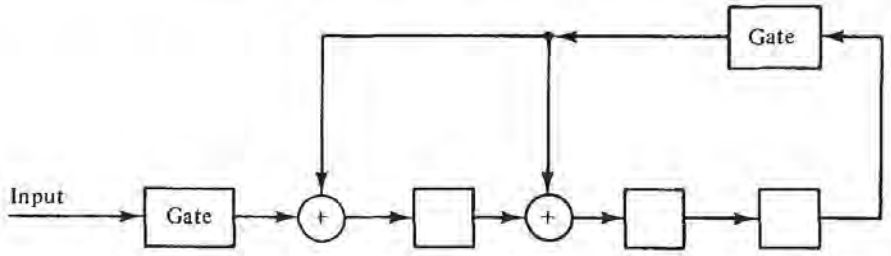


Figure 4.6 Syndrome circuit for the (7, 4) cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$.

TABLE 4.3 CONTENTS OF THE SYNDROME REGISTER SHOWN IN FIGURE 4.6 WITH $\mathbf{r} = (0\ 0\ 1\ 0\ 1\ 1\ 0)$ AS INPUT

Shift	Input	Register contents	
		0 0 0	(initial state)
1	0	0 0 0	
2	1	1 0 0	
3	1	1 1 0	
4	0	0 1 1	
5	1	0 1 1	
6	0	1 1 1	
7	0	1 0 1	(syndrome s)
8	—	1 0 0	(syndrome $\mathbf{s}^{(1)}$)
9	—	0 1 0	(syndrome $\mathbf{s}^{(2)}$)

Ex.1020 / Page 119 of 305
Sandisk Technologies, Inc. et al

ister contains the syndrome $\mathbf{s} = (1\ 0\ 1)$. If the register is shifted once more with the input gate disabled, the new contents will be $\mathbf{s}^{(1)} = (1\ 0\ 0)$, which is the syndrome of $\mathbf{r}^{(1)} = (0\ 0\ 0\ 1\ 0\ 1\ 1)$, a cyclic shift of $\mathbf{r}$.

We may shift the received vector $\mathbf{r}(X)$ into the syndrome register from the right end, as shown in Figure 4.7. 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 $\mathbf{r}(X)$.

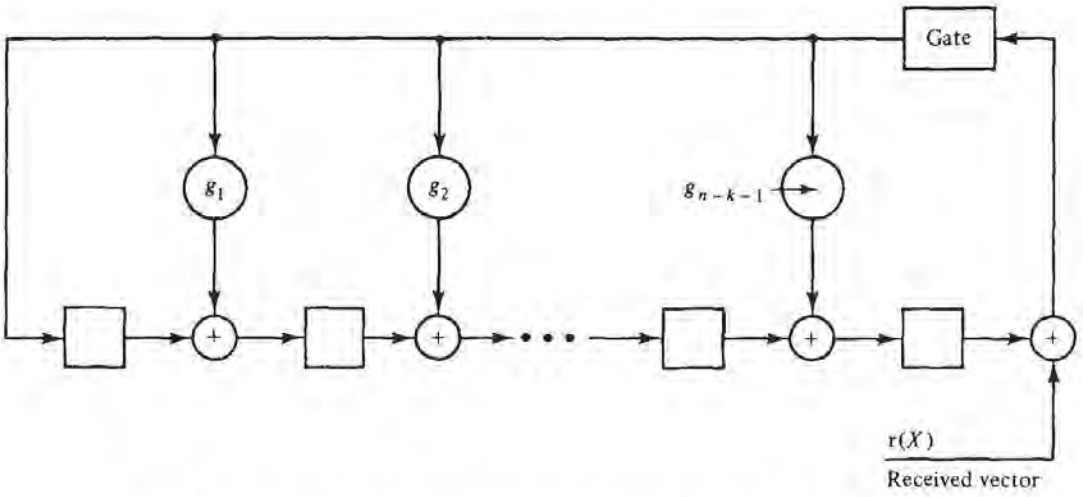


Figure 4.7 An $(n-k)$-stage syndrome circuit with input from the right end.

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 $\boldsymbol{\rho}(X)$ resulting from dividing $X^{n-k}\mathbf{r}(X)$ by the generator polynomial $\mathbf{g}(X)$. Thus, we have

$$X^{n-k}\mathbf{r}(X) = \mathbf{a}(X)\mathbf{g}(X) + \boldsymbol{\rho}(X). \tag{4.24}$$

It follows from (4.1) that $\mathbf{r}(X)$ and $\mathbf{r}^{(n-k)}(X)$ satisfy the following relation:

$$X^{n-k}\mathbf{r}(X) = \mathbf{b}(X)(X^n + 1) + \mathbf{r}^{(n-k)}(X). \tag{4.25}$$

Combining (4.24) and (4.25) and using the fact that $X^n + 1 = \mathbf{g}(X)\mathbf{h}(X)$, we have

$$\mathbf{r}^{(n-k)}(X) = [\mathbf{b}(X)\mathbf{h}(X) + \mathbf{a}(X)]\mathbf{g}(X) + \boldsymbol{\rho}(X).$$

This says that, when $\mathbf{r}^{(n-k)}(X)$ is divided by $\mathbf{g}(X)$, $\boldsymbol{\rho}(X)$ is also the remainder. Therefore, $\boldsymbol{\rho}(X)$ is indeed the syndrome of $\mathbf{r}^{(n-k)}(X)$.

Let $\mathbf{v}(X)$ be the transmitted code word and let $\mathbf{e}(X) = e_0 + e_1X + \cdots + e_{n-1}X^{n-1}$ be the error pattern. Then the received polynomial is

$$\mathbf{r}(X) = \mathbf{v}(X) + \mathbf{e}(X). \tag{4.26}$$

Since $\mathbf{v}(X)$ is a multiple of the generator polynomial $\mathbf{g}(X)$, combining (4.19) and (4.26), we have the following relationship between the error pattern and the syndrome:

$$\mathbf{e}(X) = [\mathbf{a}(X) + \mathbf{b}(X)]\mathbf{g}(X) + \mathbf{s}(X), \tag{4.27}$$

where $\mathbf{b}(X)\mathbf{g}(X) = \mathbf{v}(X)$. This shows that the syndrome is actually equal to the remainder resulting from dividing the error pattern by the generator polynomial. The syn-

Ex.1020 / Page 120 of 305
Sandisk Technologies, Inc. et al

drome 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 be correctly determined from the syndrome.

From (4.27), we see that $\mathbf{s}(X)$ is identical to zero if and only if either the error pattern $\mathbf{e}(X) = \mathbf{0}$ or it is identical to a code vector. If $\mathbf{e}(X)$ is identical to a code vector, $\mathbf{e}(X)$ 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 with the syndrome digits as inputs. 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 $\mathbf{e}(X)$ can be expressed 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. Since 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 *end-around* burst. For example,

$$e = (\overleftarrow{1\ 0\ 1}\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \overrightarrow{1\ 1\ 0\ 1})$$

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 results above, we have the following property:

Theorem 4.9. 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 ith digit position and ending at the $(i + n - k)$th digit position (i.e., errors are confined to digits $\mathbf{e}_i, \mathbf{e}_{i+1}, \cdots, \mathbf{e}_{i+n-k}$ 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 ith 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:

Ex.1020 / Page 121 of 305
Sandisk Technologies, Inc. et al

Theorem 4.10. 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 ith 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 ith 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). This leads to the following conclusion:

Theorem 4.11. For $l > n - k + 1$, the fraction of undetectable error bursts of length l is $2^{-(n-k)}$.

The analysis above shows that cyclic codes are very effective for burst-error detection.

Example 4.8

The (7, 4) cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$ has minimum distance 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.

4.5 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 to an error pattern, and error correction. We have shown in Section 4.4 that syndrome computation for cyclic codes can be accomplished 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 can be achieved with a single EXCLUSIVE-OR gate if correction is carried out in serial manner (i.e., one digit at a time); n EXCLUSIVE-OR gates are required if correction is carried out in parallel manner, as shown in Figure 3.8. The association of the syndrome to 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 the number of errors that we intend to correct. Cyclic codes have considerable algebraic and geometric properties. If these properties are properly used, simplification in the decoding circuit is possible.

The cyclic structure of a cyclic code allows us to decode a received vector $\mathbf{r}(X) = r_0 + r_1 X + r_2 X^2 + \cdots + r_{n-1} X^{n-1}$ in serial manner. 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

Ex.1020 / Page 122 of 305
Sandisk Technologies, Inc. et al

$\mathbf{s}(X)$ corresponds to a correctable error pattern $\mathbf{e}(X) = e_0 + e_1X + \cdots + e_{n-1}X^{n-1}$ 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. By doing this, we obtain $\mathbf{r}^{(1)}(X) = r_{n-1} + r_0X + \cdots + r_{n-2}X^{n-1}$ 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 the 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 $\mathbf{r}_1(X) = r_0 + r_1X + \cdots + r_{n-2}X^{n-2} + (r_{n-1} \oplus e_{n-1})X^{n-1}$. The effect of the error digit e_{n-1} on the syndrome is then removed from the syndrome $\mathbf{s}(X)$. This can be achieved 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 cyclically shift $\mathbf{r}_1(X)$ and the syndrome register once simultaneously. This shift results in a received polynomial $\mathbf{r}_1^{(1)}(X) = (r_{n-1} \oplus e_{n-1}) + r_0X + \cdots + r_{n-2}X^{n-1}$. 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 $\mathbf{g}(X)$. Since the remainders resulting from dividing $X\mathbf{s}(X)$ and X^n by $\mathbf{g}(X)$ are $\mathbf{s}^{(1)}(X)$ and 1, respectively, we have

$$\mathbf{s}_1^{(1)}(X) = \mathbf{s}^{(1)}(X) + 1.$$

Therefore, if 1 is added to the left end of the syndrome register while it is shifted, we obtain $\mathbf{s}_1^{(1)}(X)$. 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 $\mathbf{e}(X)$ 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 $\mathbf{r}(X)$ 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 4.8. 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 described as follows:

Step 1. The syndrome is formed by shifting the entire received vector into the syndrome register. At the same time the received vector is stored into 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 which 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

Ex.1020 / Page 123 of 305
Sandisk Technologies, Inc. et al

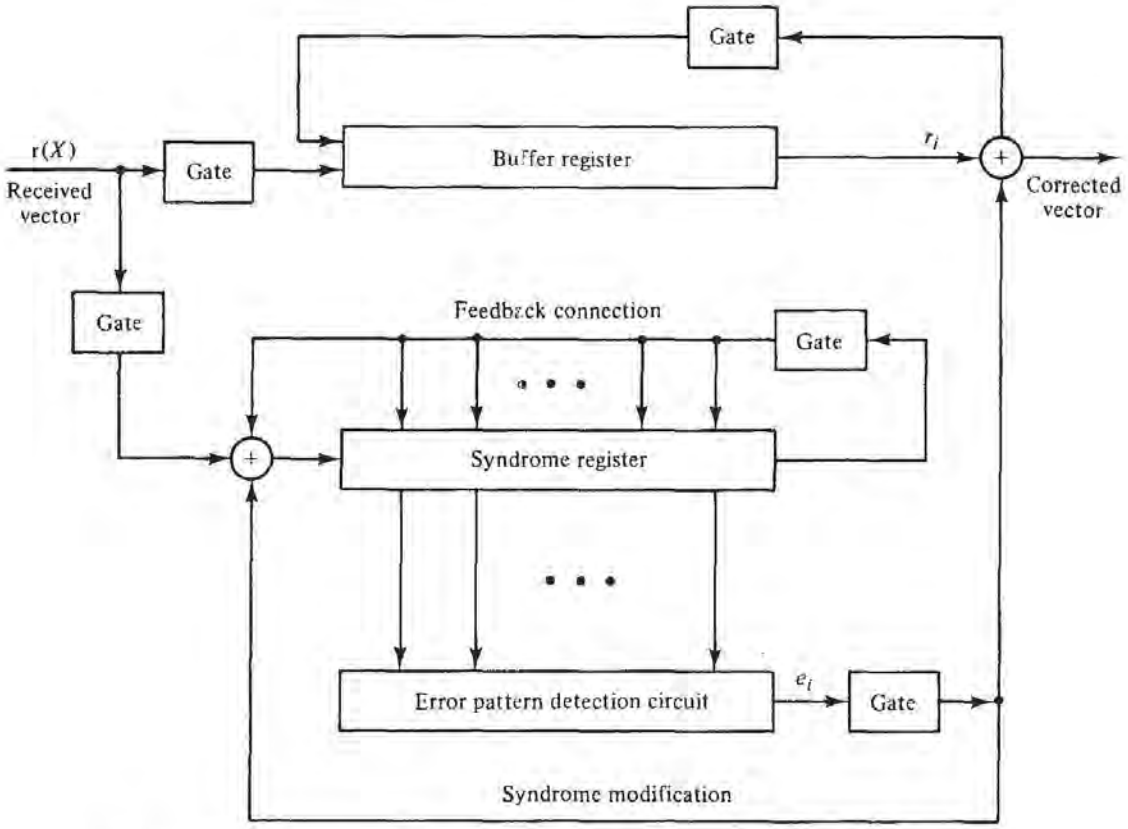


Figure 4.8 General cyclic code decoder with received polynomial $\mathbf{r}(X)$ shifted into the syndrome register from the left end.

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 correct 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 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 or not 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 above until the entire received vector is read out of the buffer register.

Ex.1020 / Page 124 of 305
Sandisk Technologies, Inc. et al

The decoder above is known as Meggitt decoder [8], which applies in principle to any cyclic code. But whether or not it is practical depends entirely on its error-pattern detection circuit. There are cases in which the error-pattern detection circuits are simple. Several of these cases are discussed in subsequent chapters.

Example 4.9

Consider the decoding of the (7, 4) cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$. This code has minimum distance 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 $\mathbf{r}(X) = r_0 + r_1 X + r_2 X^2 + r_3 X^3 + r_4 X^4 + r_5 X^5 + r_6 X^6$ is shifted into the syndrome register from the left end. The seven single-error patterns and their corresponding syndromes are listed in Table 4.4.

TABLE 4.4 ERROR PATTERNS AND THEIR SYNDROMES WITH THE RECEIVED POLYNOMIAL $\mathbf{r}(X)$ SHIFTED INTO THE SYNDROME REGISTER FROM THE LEFT END

Error pattern $\mathbf{e}(X)$	Syndrome $\mathbf{s}(X)$	Syndrome vector (s_0, s_1, s_2)
$\mathbf{e}_6(X) = X^6$	$\mathbf{s}(X) = 1 + X^2$	(1 0 1)
$\mathbf{e}_5(X) = X^5$	$\mathbf{s}(X) = 1 + X + X^2$	(1 1 1)
$\mathbf{e}_4(X) = X^4$	$\mathbf{s}(X) = X + X^2$	(0 1 1)
$\mathbf{e}_3(X) = X^3$	$\mathbf{s}(X) = 1 + X$	(1 1 0)
$\mathbf{e}_2(X) = X^2$	$\mathbf{s}(X) = X^2$	(0 0 1)
$\mathbf{e}_1(X) = X^1$	$\mathbf{s}(X) = X$	(0 1 0)
$\mathbf{e}_0(X) = X^0$	$\mathbf{s}(X) = 1$	(1 0 0)

We see that $\mathbf{e}_6(X) = X^6$ 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 $\mathbf{r}(X)$ 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., $\mathbf{e}_i(X) = X^i$]. After the entire received polynomial has been shifted into the syndrome register, the syndrome in the register is not (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 is the erroneous digit. Therefore, only the syndrome (1 0 1) needs to be detected. This can be accomplished with a single three-input AND gate. The complete decoding circuit is shown in Figure 4.9. Figure 4.10 illustrates the decoding process. Suppose that the code vector $\mathbf{v} = (1\ 0\ 0\ 1\ 0\ 1\ 1)$ [or $\mathbf{v}(X) = 1 + X^3 + X^5 + X^6$] is transmitted and $\mathbf{r} = (1\ 0\ 1\ 1\ 0\ 1\ 1)$ [or $\mathbf{r}(X) = 1 + X^2 + X^3 + X^5 + X^6$] 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 4.10, 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.

Ex.1020 / Page 125 of 305
Sandisk Technologies, Inc. et al

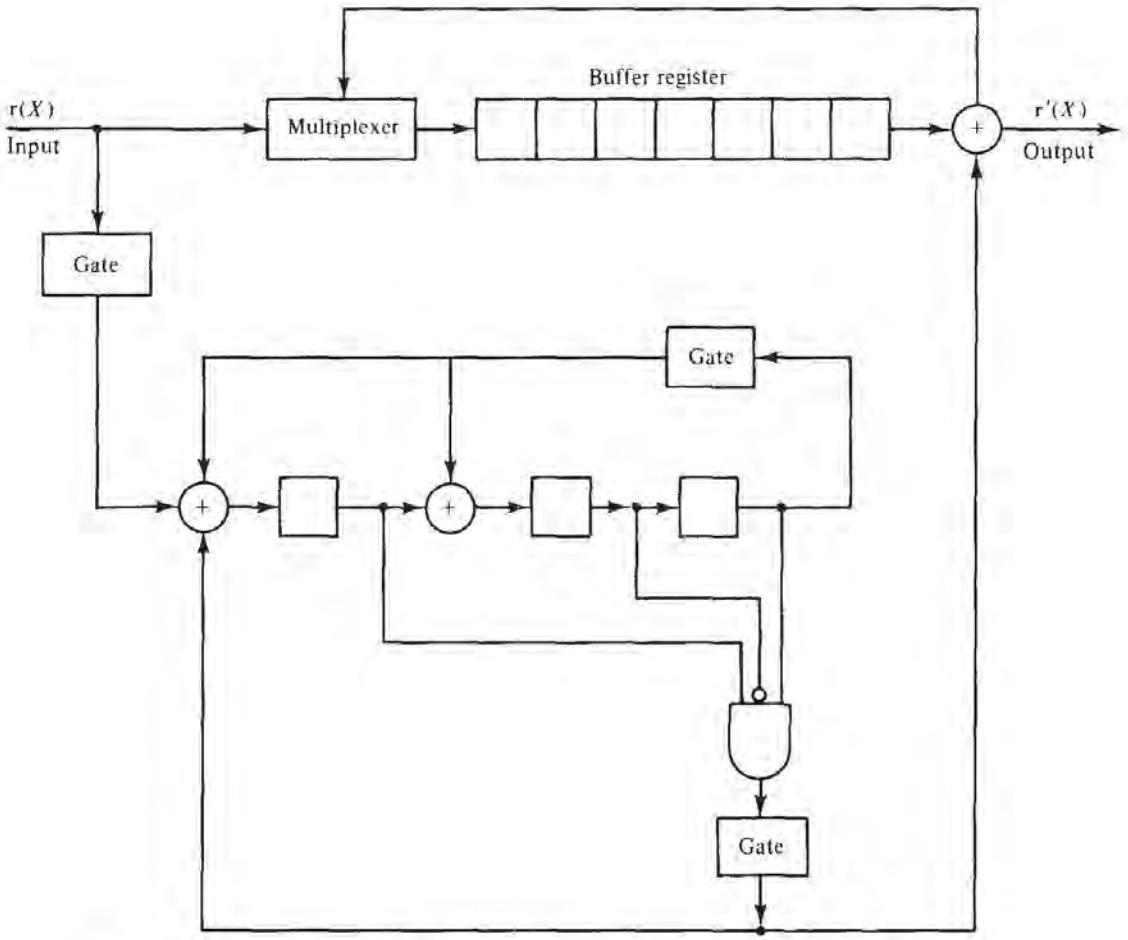


Figure 4.9 Decoding circuit for the (7, 4) cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$.

The (7, 4) cyclic code considered in Example 4.9 is actually the same code considered in Example 3.9. Comparing the decoding circuit shown in Figure 3. 9 and the decoding circuit shown in Figure 4.9, we see that the circuit shown in Figure 4.9 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 4.9 takes a longer time to decode a received vector because the decoding is carried out in serial manner. In general, speed and simplicity cannot be achieved at the same time, and a trade-off between them must be made.

The Meggitt decoder described above decodes a received polynomial $\mathbf{r}(X) = r_0 + r_1X + \cdots + r_{n-1}X^{n-1}$ 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 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 the reverse order are left as an exercise.

Ex.1020 / Page 126 of 305
Sandisk Technologies, Inc. et al

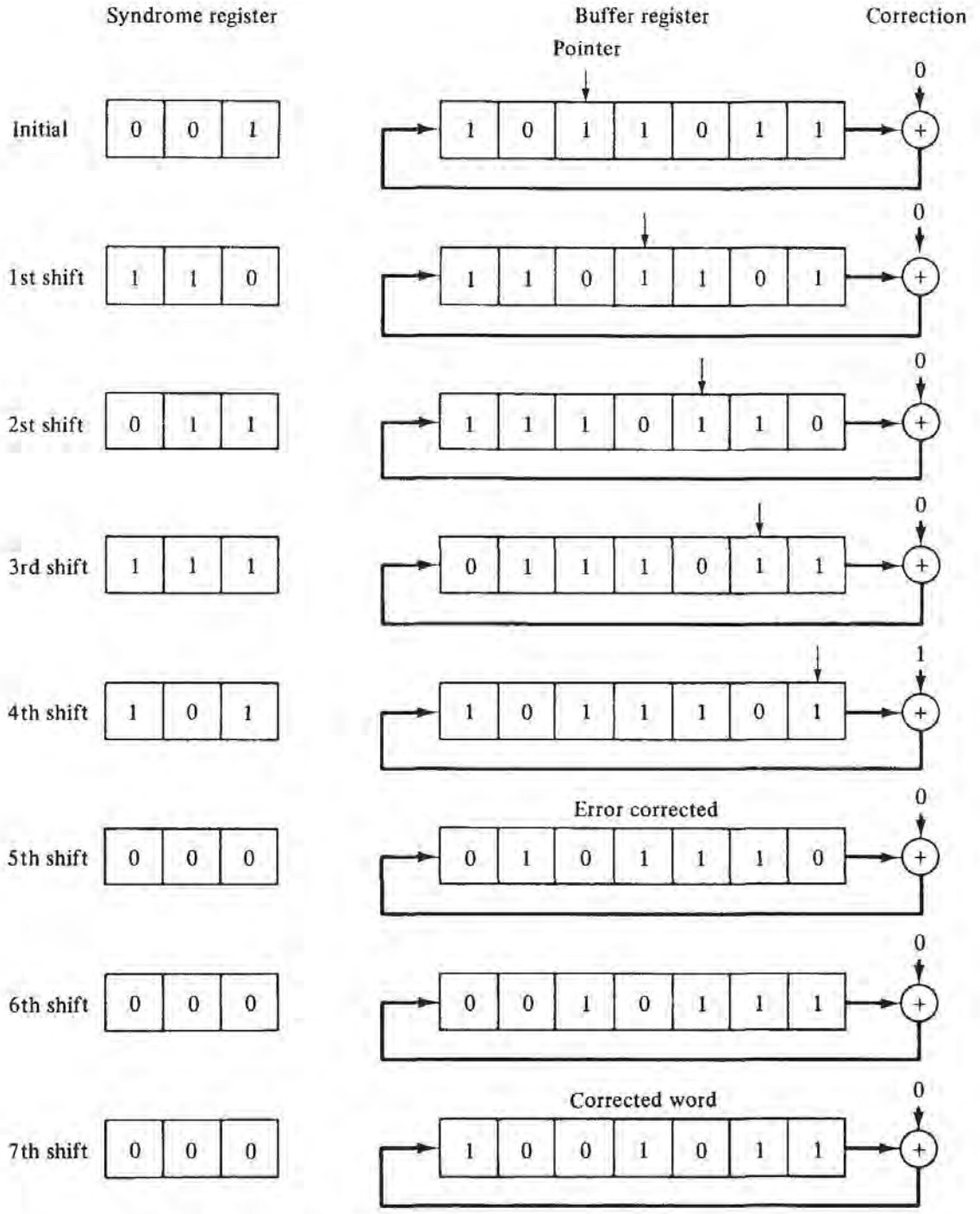


Figure 4.10 Error-correction process of the circuit shown in Figure 4.9.

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

Ex.1020 / Page 127 of 305
Sandisk Technologies, Inc. et al

r_{n-1} is corrected, the error effect must be removed from $\mathbf{s}^{(n-k)}(X)$. The new syndrome, denoted $\mathbf{s}_1^{(n-k)}(X)$, is the sum of $\mathbf{s}^{(n-k)}(X)$ and the remainder $\boldsymbol{\rho}(X)$ resulting from dividing X^{n-k-1} by the generator polynomial $\mathbf{g}(X)$. Since the degree of X^{n-k-1} is less than the degree of $\mathbf{g}(X)$,

$$\boldsymbol{\rho}(X) = X^{n-k-1}.$$

Therefore

$$\mathbf{s}_1^{(n-k)}(X) = \mathbf{s}^{(n-k)}(X) + X^{n-k-1}.$$

This 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 4.11. The decoding process of the decoder shown in Figure 4.11 is identical to the decoding process of the decoder shown in Figure 4.8.

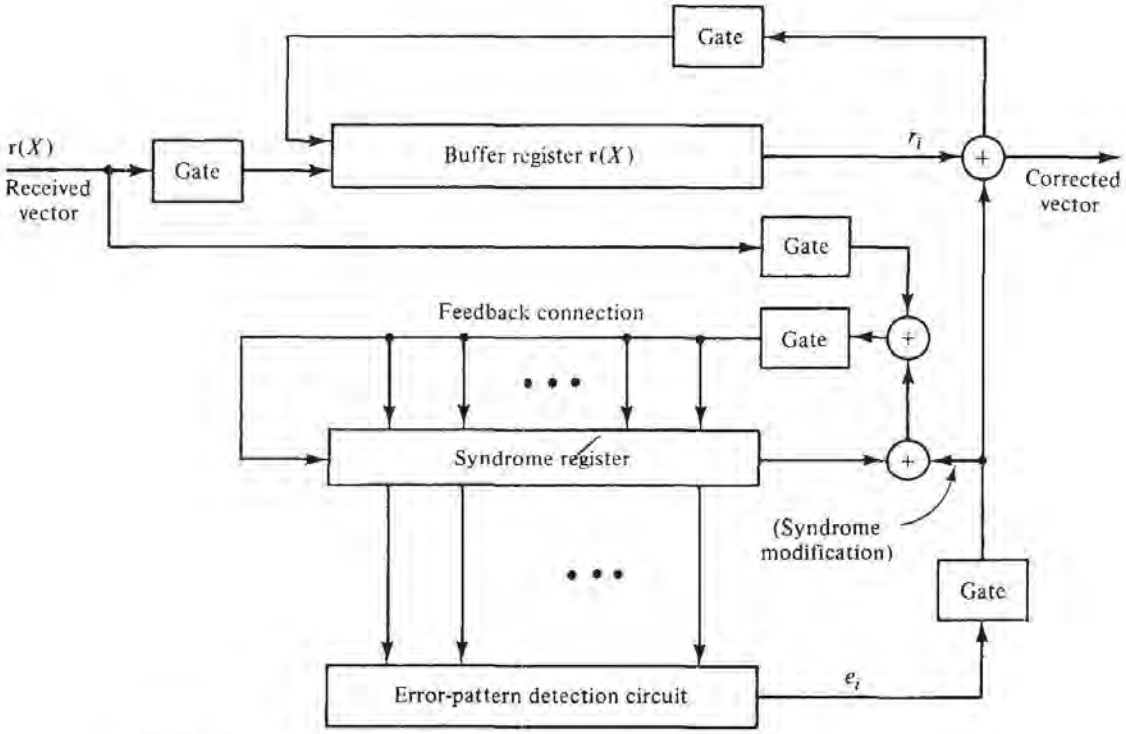


Figure 4.11 General cyclic code decoder with received polynomial $\mathbf{r}(X)$ shifted into the syndrome register from the right end.

Example 4.10

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 4.5.

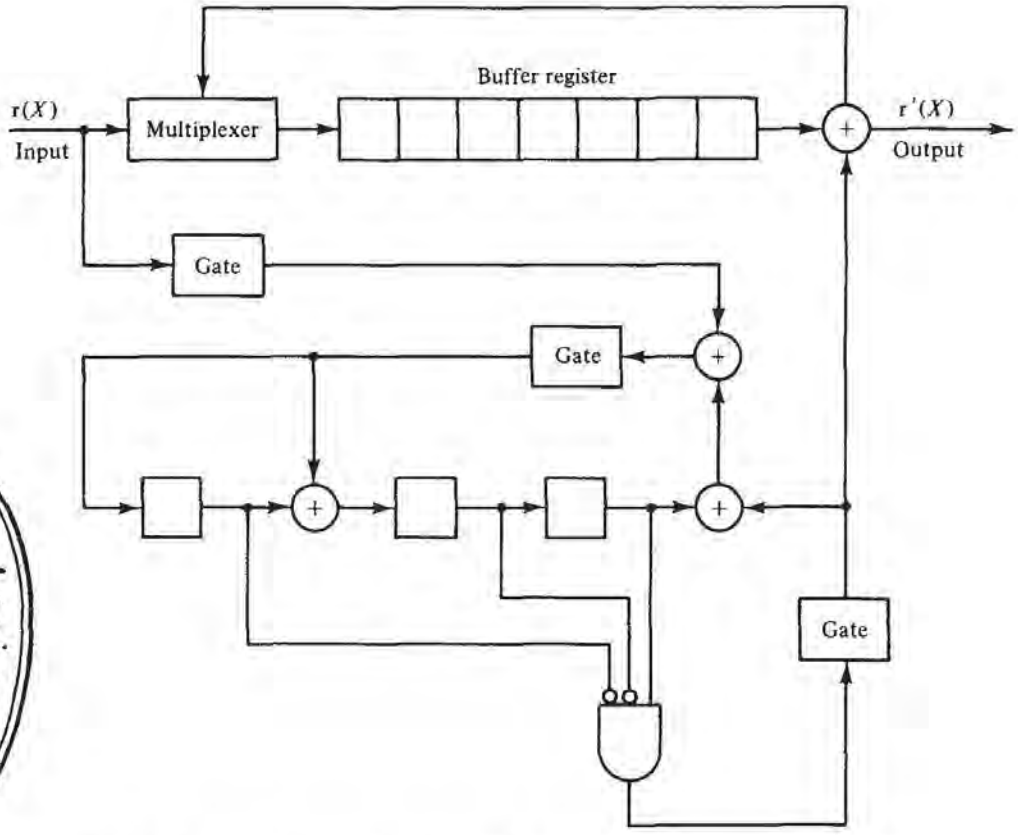
We see that only when $\mathbf{e}(X) = X^6$ occurs, the syndrome is (0 0 1) after the entire received polynomial $\mathbf{r}(X)$ has been shifted into the syndrome register. If the single error occurs at the location X^i with $i \neq 6$, the syndrome in the register will not be (0 0 1) after the entire received polynomial $\mathbf{r}(X)$ has been shifted into the

Ex.1020 / Page 128 of 305
Sandisk Technologies, Inc. et al

TABLE 4.5 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)

syndrome register. However, after another $6 - i$ shifts, the syndrome register will contain (0 0 1). Based on this fact, we obtain another decoding circuit for the (7, 4) cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$, as shown in Figure 4.12. We see that the circuit shown in Figure 4.9 and the circuit shown in Figure 4.12 have the same complexity.



دانشگاه صنعتی شریف
کتابخانه مرکزی

Figure 4.12 Decoding circuit for the (7, 4) cyclic code generated by $\mathbf{g}(X) = 1 + X + X^3$.

Ex.1020 / Page 129 of 305
Sandisk Technologies, Inc. et al

4.6 CYCLIC HAMMING CODES

The Hamming codes presented in Section 3.7 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 above is indeed a Hamming code. For this purpose, we examine its parity-check matrix in systematic form. The method presented in Section 4.2 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

$$X^{m+i} = \mathbf{a}_i(X)\mathbf{p}(X) + \mathbf{b}_i(X), \tag{4.28}$$

where the remainder $\mathbf{b}_i(X)$ is of the form

$$\mathbf{b}_i(X) = b_{i0} + b_{i1}X + \cdots + b_{i,m-1}X^{m-1}.$$

Since 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 (4.28) that

$$X^{m+i} = \mathbf{a}_i(X)\mathbf{p}(X) + X^j.$$

Rearranging the equation above, we have

$$X^j(X^{m+i-j} + 1) = \mathbf{a}_i(X)\mathbf{p}(X).$$

Since X^j and $\mathbf{p}(X)$ are relatively prime, the equation above 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 (4.28) 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 two equations above, the following relation is obtained:

$$X^{m+i}(X^{j-i} + 1) = [\mathbf{a}_i(X) + \mathbf{a}_j(X)]\mathbf{p}(X).$$

This equation implies that $\mathbf{p}(X)$ divides $X^{j-i} + 1$. This is impossible since $i \neq j$ and $j - i < 2^m - 1$. Therefore, $\mathbf{b}_i(X) \neq \mathbf{b}_j(X)$.

Let $\mathbf{H} = [\mathbf{I}_m \quad \mathbf{Q}]$ be the parity-check matrix 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 $\mathbf{b}_i = (b_{i0}, b_{i1}, \ldots, b_{i,m-1})$ be the m-tuple corresponding to $\mathbf{b}_i(X)$. It follows from (4.17) 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 analysis above 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,

Ex.1020 / Page 130 of 305
Sandisk Technologies, Inc. et al

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.

Decoding of cyclic Hamming codes can be easily implemented. 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 $\mathbf{p}(X)$. Since $\mathbf{p}(X)$ 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 $(0, 0, \ldots, 0, 1)$. If a single error occurs at any other location of $\mathbf{r}(X)$, the resultant syndrome will be different from $(0, 0, \ldots, 0, 1)$. Based on this fact, only a single m-input AND gate is needed to detect the syndrome pattern $(0, 0, \ldots, 0, 1)$. The inputs to this AND gate are $s'_0, s'_1, \ldots, s'_{m-2}$ 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 4.13. The decoding operation is described in the following steps:

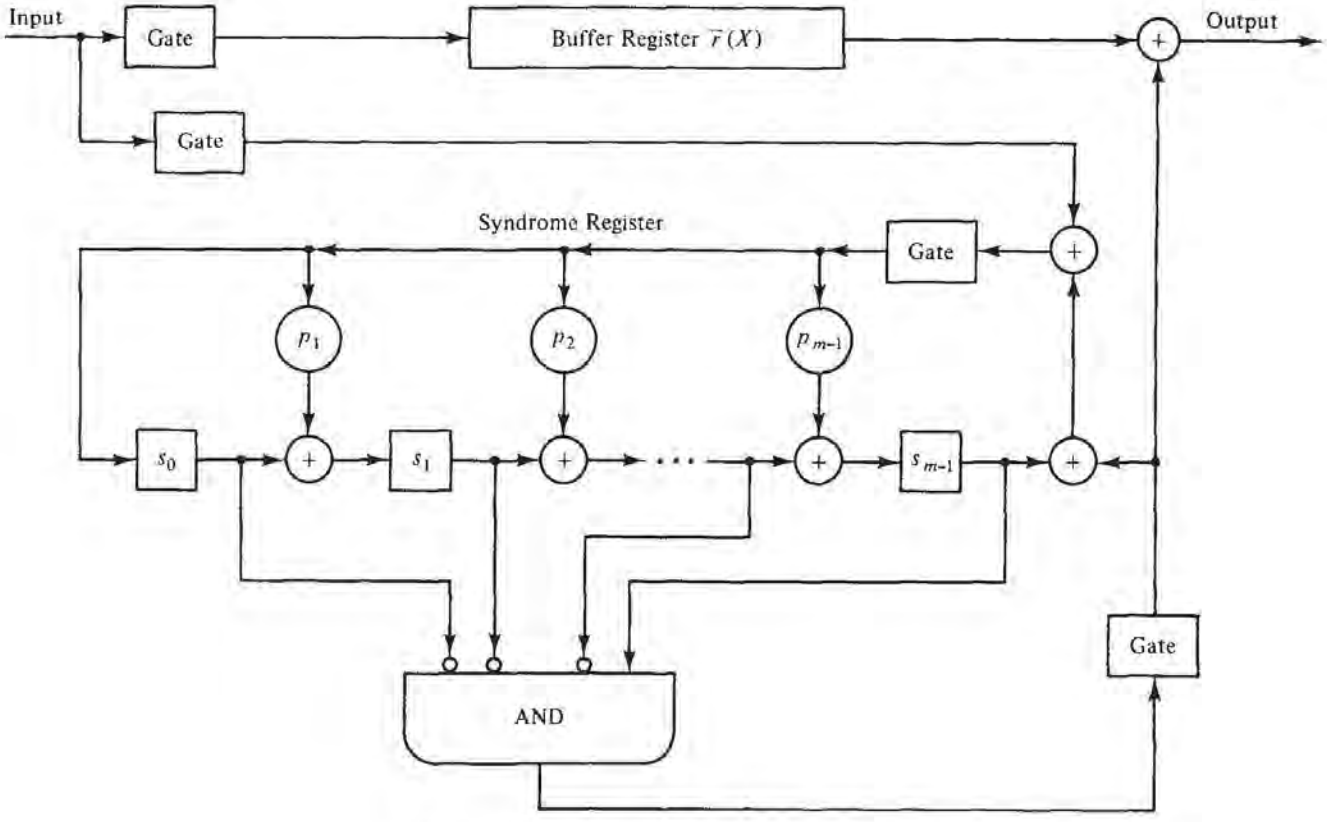


Figure 4.13 Decoder for a cyclic Hamming code.

Ex.1020 / Page 131 of 305
Sandisk Technologies, Inc. et al

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 into 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, . . ., 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 above 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. Since 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. Since $\mathbf{p}(X)$ is a proper factor of $\mathbf{g}(X)$, C_1 contains C_2 as a proper subcode. In fact, C_2 consists of the even-weight code vectors of C_1 as all its vectors. This is due to the fact that 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. Since $X^i + X^j$ 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

$$X^i + X^j + X^k = \mathbf{c}(X)\mathbf{p}(X) + \mathbf{d}(X). \tag{4.29}$$

Ex.1020 / Page 132 of 305
Sandisk Technologies, Inc. et al

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,

$$X^l = \mathbf{f}(X)\mathbf{p}(X) + \mathbf{d}(X). \tag{4.30}$$

The integer l cannot be equal to any of the three integers i, j, and k. Suppose that $l = i$. From (4.29) and (4.30) we would obtain

$$X^j + X^k = [\mathbf{c}(X) + \mathbf{f}(X)]\mathbf{p}(X).$$

This 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 (4.29) and (4.30), we obtain

$$X^i + X^j + X^k + X^l = [\mathbf{c}(X) + \mathbf{f}(X)]\mathbf{p}(X).$$

Since $X + 1$ 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 vector 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 minimum weight (or distance) 4. Hence it is capable of correcting any single error and simultaneously detecting any combination of double errors.

The decoding circuit for the single-error-correcting Hamming code shown in Figure 4.13 can be modified to decode the single-error-correcting and double-error-detecting Hamming code. Let $\mathbf{r}(X)$ be the received polynomial. Dividing $X^m\mathbf{r}(X)$ by $\mathbf{p}(X)$ and $\mathbf{r}(X)$ by $(X + 1)$, respectively, we have

$$X^m\mathbf{r}(X) = \mathbf{a}_1(X)\mathbf{p}(X) + \mathbf{s}_p(X)$$

and

$$\mathbf{r}(X) = \mathbf{a}_2(X)(X + 1) + \sigma,$$

where $\mathbf{s}_p(X)$ is of degree $m - 1$ or less and σ is either 0 or 1. If $\mathbf{s}_p(X) = 0$ and $\sigma = 0$, $\mathbf{r}(X)$ is divisible by $(1 + X)\mathbf{p}(X)$ and is a code polynomial; otherwise, $\mathbf{r}(X)$ is not a code polynomial. We define the syndrome of $\mathbf{r}(X)$ as

$$\mathbf{s}(X) = X\mathbf{s}_p(X) + \sigma.$$

If a single error occurs, $\mathbf{s}_p(X) \neq 0$ and $\sigma = 1$. However, when an error pattern with double errors occurs, we would have $\mathbf{s}_p(X) \neq 0$ and $\sigma = 0$. Based on these facts, we may implement a decoder for a single-error-correcting and double-error-detecting cyclic Hamming code as shown in Figure 4.14. The error-correction and error-detection operations are described as follows:

1. For $\sigma = 0$ and $\mathbf{s}_p(X) = 0$, the decoder assumes that there is no error in the received polynomial. No corrective action takes place.
2. For $\sigma = 1$ and $\mathbf{s}_p(X) \neq 0$, the decoder assumes that a single error has occurred and proceeds with the error-correction process as described in the decoding of a single-error-correcting cyclic Hamming code.
3. For $\sigma = 0$ and $\mathbf{s}_p(X) \neq 0$, the decoder assumes that double errors have occurred. The error alarm is turned on.

Ex.1020 / Page 133 of 305
Sandisk Technologies, Inc. et al

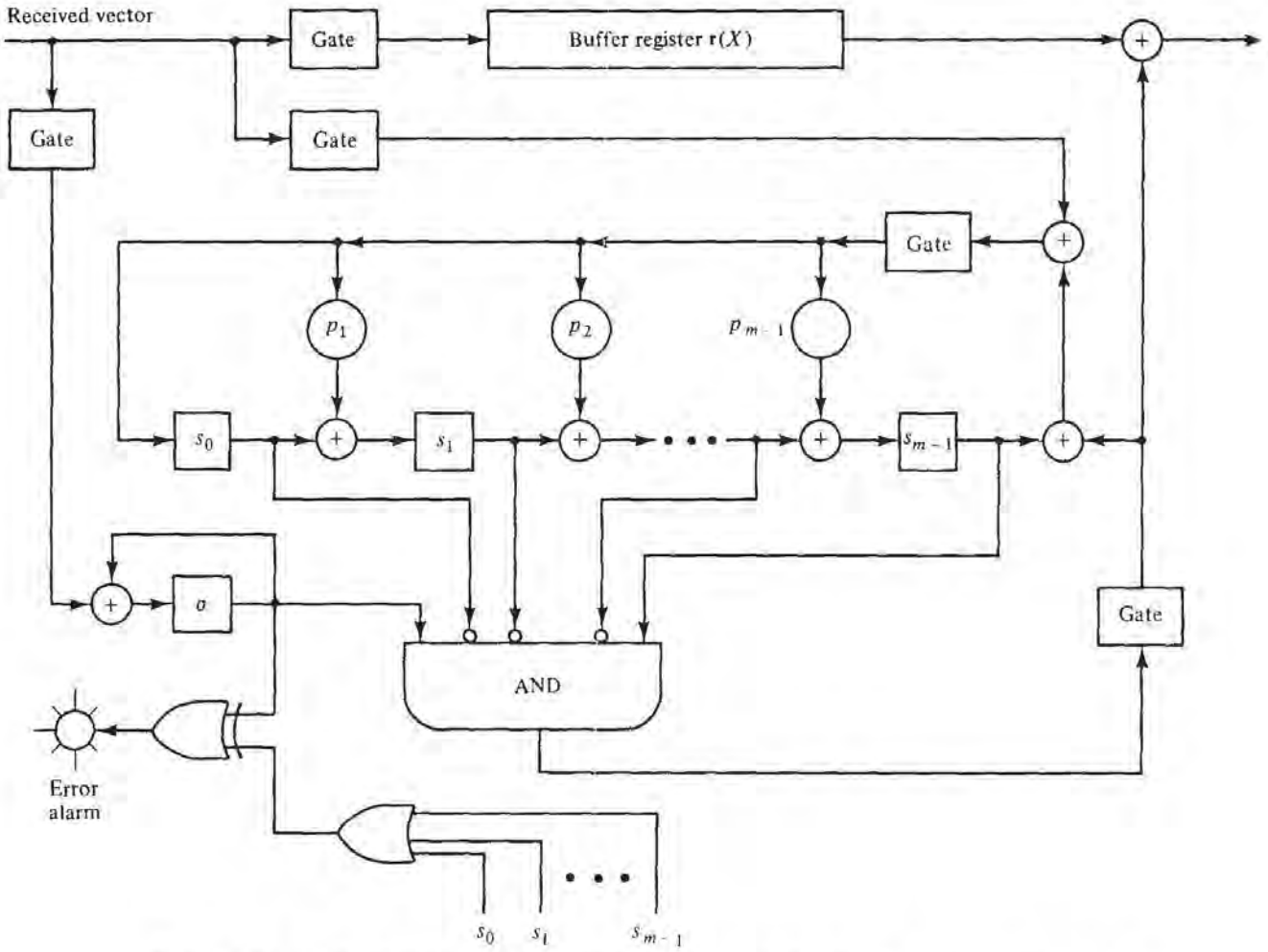


Figure 4.14 Decoding circuit for a single-error-correcting and double-error-detecting cyclic Hamming code.

4. For $\sigma = 1$ and $\mathbf{s}_p(X) = 0$, the error alarm is also turned on. This happens when an error pattern with odd number (greater than 1) of errors has occurred and the error pattern is divisible by $\mathbf{p}(X)$.

Since the distance 4 Hamming code C_2 of length $2^m - 1$ consists of the even-weight code vectors of the distance 3 Hamming code C_1 of length $2^m - 1$ as its code vectors, 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,

$$A_2(z) = \tfrac{1}{2}[A_1(z) + A_1(-z)] \tag{4.31}$$

(see Problem 4.8). Since $A_1(z)$ is known and is given by (3.43), $A_2(z)$ can be determined from (3.43) and (4.31):

$$A_2(z) = \frac{1}{2(n+1)}\{(1+z)^n + (1-z)^n + 2n(1-z^2)^{(n-1)/2}\}, \tag{4.32}$$

Ex.1020 / Page 134 of 305
Sandisk Technologies, Inc. et al

where $n = 2^m - 1$. The dual of a distance 4 cyclic Hamming code is a $(2^m - 1, m + 1)$ cyclic code which has the following weight distribution:

$$B_0 = 1, \qquad B_{2^{m-1}-1} = 2^m - 1, \qquad B_{2^{m-1}} = 2^m - 1, \qquad B_{2^m-1} = 1.$$

Therefore, the weight enumerator for the dual of a distance 4 cyclic Hamming code is

$$B_2(z) = 1 + (2^m - 1)z^{2^{m-1}-1} + (2^m - 1)z^{2^{m-1}} + z^{2^m-1}. \tag{4.33}$$

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 either from (3.33) and (4.32) or from (3.36) and (4.33). Computing $P_u(E)$ from (3.36) and (4.33), we obtain the following expression:

$$P_u(E) = 2^{-(m+1)}\{1 + 2(2^m - 1)(1 - p)(1 - 2p)^{2^{m-1}-1} + (1 - 2p)^{2^m-1}\} - (1 - p)^{2^m-1}. \tag{4.34}$$

Again, from (4.34), 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 4.21).

Distance 3 and distance 4 cyclic Hamming codes are often used in communication systems for error detection.

4.7 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 code vectors for which the l leading high-order information digits are identical to zero. There are 2^{k-l} such code vectors and they form a linear subcode of C. If the l zero information digits are deleted from each of these code vectors, 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 $\mathbf{r}(X) = r_0 + r_1X + \cdots + r_{n-l-1}X^{n-l-1}$ 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

Ex.1020 / Page 135 of 305
Sandisk Technologies, Inc. et al

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 $\mathbf{g}(X)$. Since 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 for 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

$$X^{n-k+l}\mathbf{r}(X) = \mathbf{a}_1(X)\mathbf{g}(X) + \mathbf{s}^{(n-k+l)}(X), \tag{4.35}$$

where $\mathbf{s}^{(n-k+l)}(X)$ is the remainder and is the syndrome for decoding the received digit r_{n-l-1}. Next, we divide X^{n-k+l} by $\mathbf{g}(X)$. Let $\boldsymbol{\rho}(X) = \rho_0 + \rho_1 X + \cdots + \rho_{n-k-1} X^{n-k-1}$ be the remainder resulting from this division. Then we have the following relation:

$$\boldsymbol{\rho}(X) = X^{n-k+l} + \mathbf{a}_2(X)\mathbf{g}(X). \tag{4.36}$$

Multiplying both sides of (4.36) by $\mathbf{r}(X)$ and using the equality of (4.35), we obtain the following relation between $\boldsymbol{\rho}(X)\mathbf{r}(X)$ and $\mathbf{s}^{(n-k+l)}(X)$:

$$\boldsymbol{\rho}(X)\mathbf{r}(X) = [\mathbf{a}_1(X) + \mathbf{a}_2(X)\mathbf{r}(X)]\mathbf{g}(X) + \mathbf{s}^{(n-k+l)}(X). \tag{4.37}$$

The equality above suggests that we can obtain the syndrome $\mathbf{s}^{(n-k+l)}(X)$ by multiplying $\mathbf{r}(X)$ by $\boldsymbol{\rho}(X)$ and dividing the product $\boldsymbol{\rho}(X)\mathbf{r}(X)$ by $\mathbf{g}(X)$. Computing $\mathbf{s}^{(n-k+l)}(X)$ this way, the extra l shifts of the syndrome register can be avoided. Simultaneously multiplying $\mathbf{r}(X)$ by $\boldsymbol{\rho}(X)$ and dividing $\boldsymbol{\rho}(X)\mathbf{r}(X)$ by $\mathbf{g}(X)$ can be accomplished by a circuit as shown in Figure 4.15. 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.

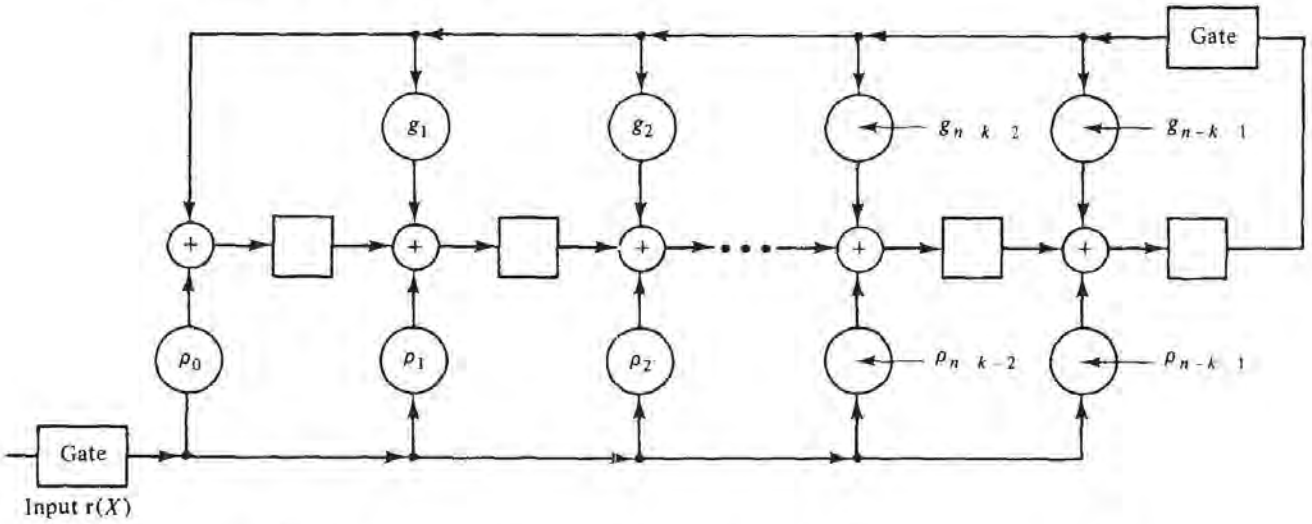


Figure 4.15 Circuit for multiplying $\mathbf{r}(X)$ by $\boldsymbol{\rho}(X) = \rho_0 + \rho_1 X + \cdots + \rho_{n-k-1} X^{n-k-1}$ and dividing $\boldsymbol{\rho}(X)\mathbf{r}(X)$ by $\mathbf{g}(X) = 1 + g_1 X + \cdots + X^{n-k}$.

Example 4.11

For $m = 5$, there exists a (31, 26) cyclic Hamming code generated by $\mathbf{g}(X) = 1 + X^2 + X^5$. 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

Ex.1020 / Page 136 of 305
Sandisk Technologies, Inc. et al

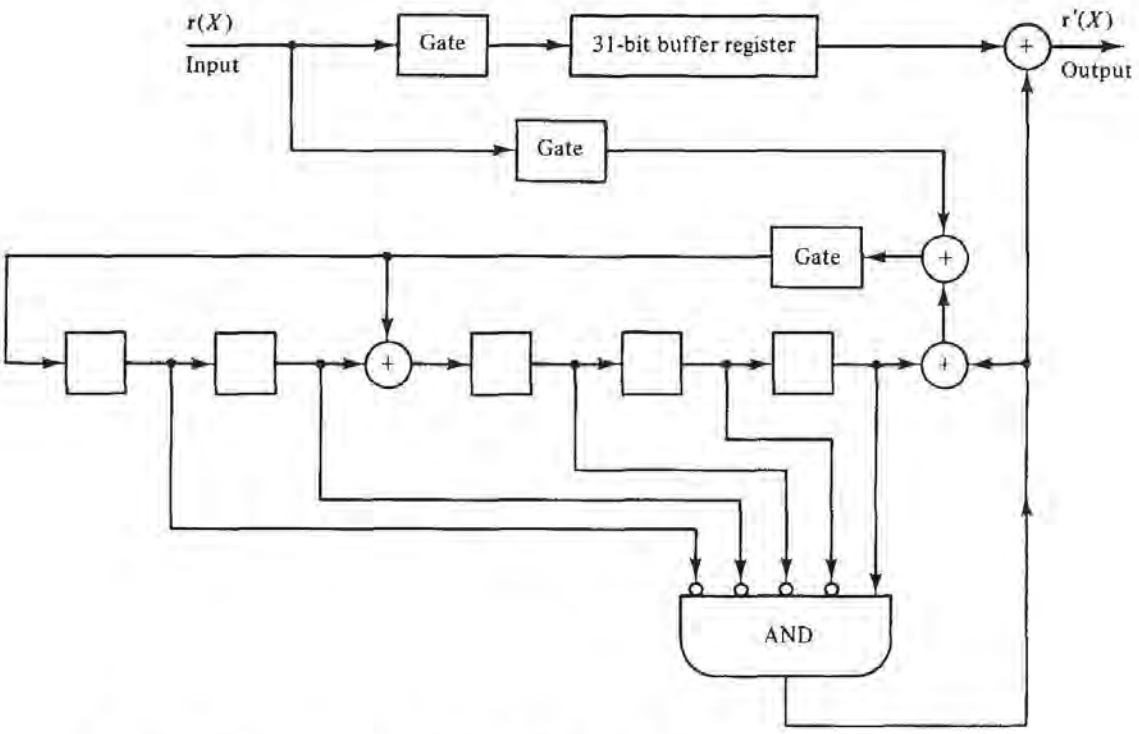


Figure 4.16 Decoding circuit for the (31, 26) cyclic Hamming code generated by $\mathbf{g}(X) = 1 + X^2 + X^5$.

4.16. 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 $\boldsymbol{\rho}(X)$. Dividing X^{n-k+3} by $\mathbf{g}(X) = 1 + X^2 + X^5$, we have

$$
\begin{array}{r}
X^3 + 1 \\
X^5 + X^2 + 1 \overline{)\, X^8} \\
\underline{X^8 + X^5 + X^3} \\
X^5 + X^3 \\
\underline{X^5 + X^2 + 1} \\
X^3 + X^2 + 1
\end{array}
$$

and $\boldsymbol{\rho}(X) = 1 + X^2 + X^3$. The modified decoding circuit for the (28, 23) shortened code is shown in Figure 4.17.

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 $\mathbf{e}(X)$ 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 $\boldsymbol{\rho}(X) = \rho_0 + \rho_1 X + \cdots + \rho_{n-k-1} X^{n-k-1}$ be the remainder resulting from dividing $X^{n-l-1} \cdot X^{n-k} = X^{2n-k-l-1}$ by the generator poly-

Ex.1020 / Page 137 of 305
Sandisk Technologies, Inc. et al

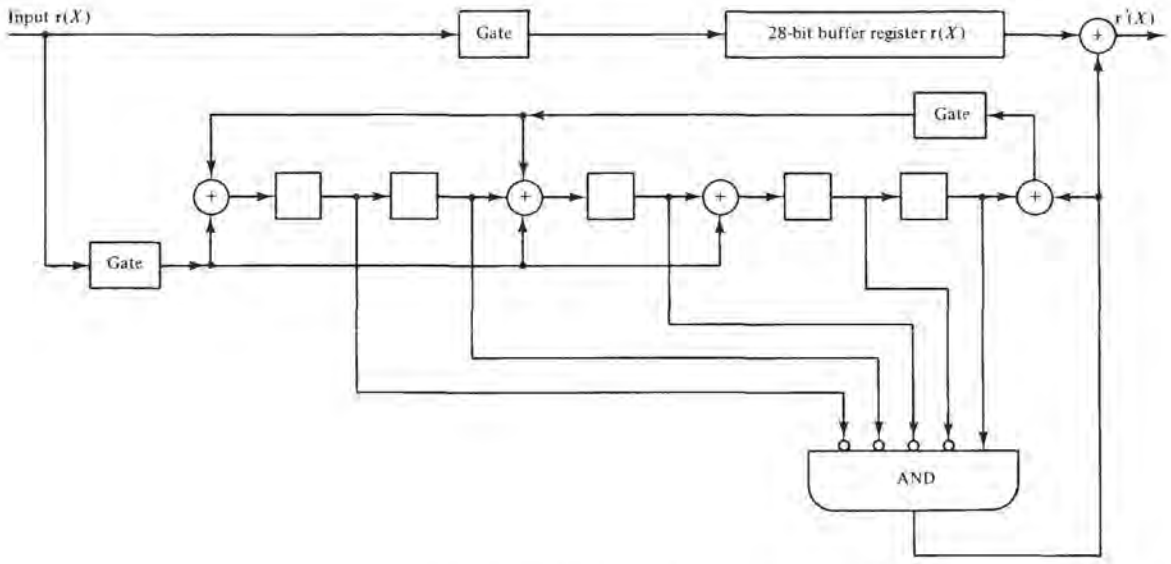


Figure 4.17 Decoding circuit for the (28, 23) shortened cyclic code generated by $g(X) = 1 + X^2 + X^5$.

Ex.1020 / Page 138 of 305
Sandisk Technologies, Inc. et al

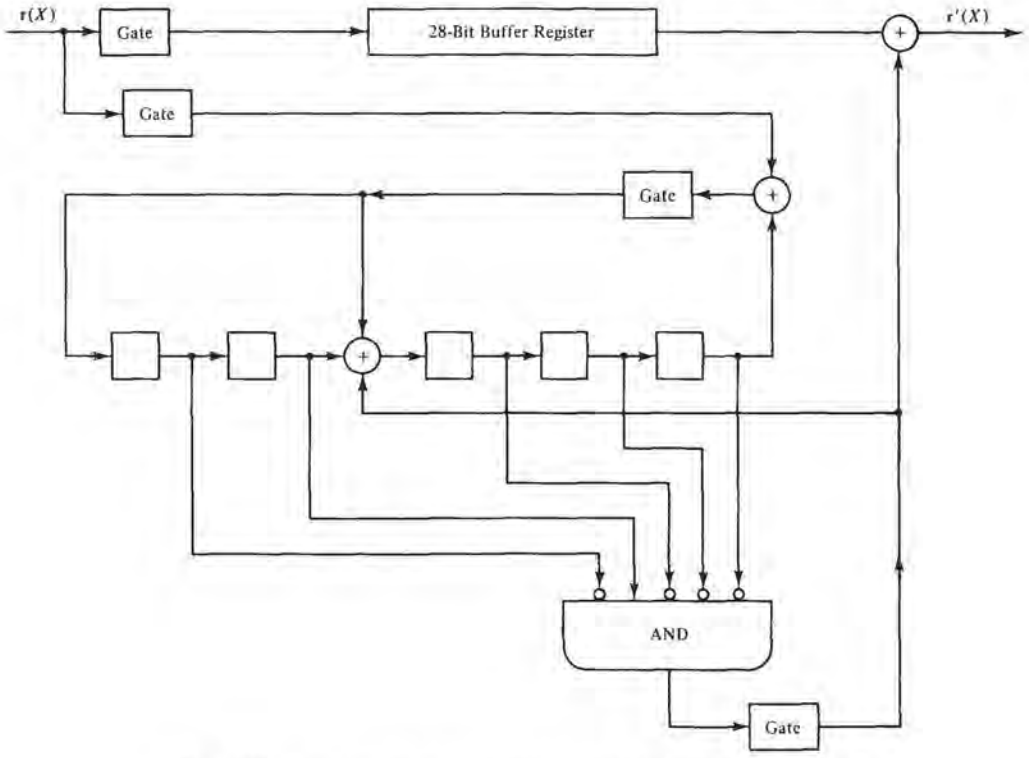


Figure 4.18 Another decoding circuit for the (28, 23) shortened Hamming code generated by $g(X) = 1 + X^2 + X^5$.

nomial $\mathbf{g}(X)$. Then the effect of the error digit e_{n-l-1} on the syndrome is removed by adding $\boldsymbol{\rho}(X)$ to the syndrome in the syndrome register.

Example 4.12

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 $\mathbf{g}(X) = 1 + X^2 + X^5$. This resultant syndrome is (0 1 0 0 0). 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 (0 1 0 0 0). By doing this, the extra three shifts of the syndrome register can be avoided. The resultant decoding circuit with syndrome resetting is shown in Figure 4.18.

PROBLEMS

4.1. Consider the (15, 11) cyclic Hamming code generated by $\mathbf{g}(X) = 1 + X + X^4$.
- **(a)** Determine the parity polynomial $\mathbf{h}(X)$ of this code.
- **(b)** Determine the generator polynomial of its dual code.
- **(c)** Find the generator and parity matrices in systematic form for this code.

4.2. Devise an encoder and a decoder for the (15, 11) cyclic Hamming code generated by $\mathbf{g}(X) = 1 + X + X^4$.

4.3. 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.

4.4. Shorten the (15, 11) cyclic Hamming by deleting the seven leading high-order message digits. The resultant code is a (8, 4) shortened cyclic code. Design a decoder for this code which eliminates the extra shifts of the syndrome register.

4.5. 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 which requires no extra shifts of the syndrome register.

4.6. Let $\mathbf{g}(X)$ be the generator polynomial of a binary cyclic code of length n.
- **(a)** Show that, if $\mathbf{g}(X)$ has $X + 1$ as a factor, the code contains no code vectors of odd weight.
- **(b)** If n is odd and $X + 1$ is not a factor of $\mathbf{g}(X)$, show that the code contains a code vector consisting of all 1's.
- **(c)** Show that the code has minimum weight at least 3 if n is the smallest integer such that $\mathbf{g}(X)$ divides $X^n + 1$.

4.7. 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 $\mathbf{g}(X)$.
- **(a)** Show that $\mathbf{g}^*(X)$ also generates an (n, k) cyclic code.

Ex.1020 / Page 140 of 305
Sandisk Technologies, Inc. et al

(b) Let C^* denote the cyclic code generated by $\mathbf{g}^*(X)$. Show that C and C^* have the same weight distribution. [*Hint:* Show that

$$\mathbf{v}(X) = v_0 + v_1X + \cdots + v_{n-2}X^{n-2} + v_{n-1}X^{n-1}$$

is a code polynomial in C if and only if

$$X^{n-1}\mathbf{v}(X^{-1}) = v_{n-1} + v_{n-2}X + \cdots + v_1X^{n-2} + v_0X^{n-1}$$

is a code polynomial in C^*.]

4.8. Consider a cyclic code C of length n which consists of both odd-weight and even-weight code vectors. 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

$$A_1(z) = \tfrac{1}{2}[A(z) + A(-z)].$$

4.9. 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.

4.10. 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.

4.11. 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 4.10.)

4.12. For a cyclic code, if an error pattern $\mathbf{e}(X)$ is detectable, show that its ith cyclic shift $\mathbf{e}^{(i)}(X)$ is also detectable.

4.13. In decoding 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 4.11. 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 4.8.

4.14. 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.

4.15. Let $\mathbf{g}(X)$ be the generator polynomial of an (n, k) cyclic code C.

(a) Show that $\mathbf{g}(X^\lambda)$ generates a $(\lambda n, \lambda k)$ cyclic code.

(b) Show that the code generated by $\mathbf{g}(X^\lambda)$ has the same minimum weight as that of the code generated by $\mathbf{g}(X)$.

4.16. 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.]

4.17. 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?

Ex.1020 / Page 141 of 305
Sandisk Technologies, Inc. et al

If your answer is "yes," find the shortest cyclic code with $\phi(X)$ as the generator polynomial.

4.18. 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.

4.19. Consider the Galois field GF(2^m), which is constructed based on the 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 which has α as a root is a code polynomial in the Hamming code generated by $\mathbf{p}(X)$.

4.20. Let C_1 and C_2 be two cyclic codes of length n which 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?

4.21. Show that the probability of an undetected error for the distance 4 cyclic Hamming codes is upper bounded by $2^{-(m+1)}$.

4.22. 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 code vector in C_d and let $\mathbf{v}^{(i)}(X)$ be the ith 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 code vector and $2^m - 1$ code vectors of weight 2^{m-1}.

[*Hint:* For part (a), use (4.1) 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).]

4.23. For an (n, k) cyclic code, show that the syndrome of an end-around burst of length $n - k$ cannot be zero.

4.24. Design a Meggitt decoder that decodes a received polynomial $\mathbf{r}(X) = r_0 + r_1 X + \cdots + r_{n-1}X^{n-1}$ 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.

REFERENCES

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.
3. W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2nd ed., MIT Press, Cambridge, Mass., 1972.

Ex.1020 / Page 142 of 305
Sandisk Technologies, Inc. et al

4. T. Kasami, N. Tokura, Y. Iwadare, and Y. Inagaki, *Coding Theory*, Corona, Tokyo, 1974 (in Japanese).
5. I. F. Blake and R. C. Mullin, *The Mathematical Theory of Coding*, Academic Press, New York, 1975.
6. F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, 1977.
7. R. J. McEliece, *The Theory of Information and Coding*, Addison-Wesley, Reading, Mass., 1977.
8. J. E. Meggitt, "Error Correcting Codes and Their Implementation," *IRE Trans. Inf. Theory*, IT-7, pp. 232–244, October 1961.

Ex.1020 / Page 143 of 305
Sandisk Technologies, Inc. et al

5

Error-Trapping Decoding for Cyclic Codes

In principle, the general decoding method of Meggitt's applies to any cyclic code, but refinements are necessary for practical implementation. In this chapter a practical variation of Meggitt decoding, called *error-trapping decoding*, is presented. A decoder based on this decoding technique employs a very simple combinational logic circuit for error detection and correction. Error-trapping decoding was devised independently by Kasami [1], Mitchell [2, 3], and Rudolph [4]. It is most effective for decoding single-error-correcting codes, some short double-error-correcting codes, and burst-error-correcting codes. When it is applied to long and high rate codes with large error-correcting capability, it becomes very ineffective and much error-correcting capability will be sacrificed. Several improved error-trapping methods for decoding multiple-error-correcting codes have been devised. One such method due to Kasami [5] is presented in this chapter.

Error-trapping decoding is particularly effective for decoding burst-error-correcting codes, discussed in Chapter 9.

5.1 ERROR-TRAPPING DECODING

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 vector $\mathbf{v}(X)$ 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 have shown in Section 4.4 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 $\mathbf{g}(X)$

Ex.1020 / Page 144 of 305
Sandisk Technologies, Inc. et al

[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, $X^k, X^{k+1}, \ldots, X^{n-1}$ of $\mathbf{r}(X)$ [i.e., $\mathbf{e}(X) = e_k X^k + e_{k+1} X^{k+1} + \cdots + e_{n-1} X^{n-1}$]. If $\mathbf{r}(X)$ is cyclically shifted $n - k$ times, the errors will be confined to $n - k$ low-order parity positions, $X^0, X^1, \ldots, X^{n-k-1}$ of $\mathbf{r}^{(n-k)}(X)$. The corresponding error pattern is then

$$\mathbf{e}^{(n-k)}(X) = e_k + e_{k+1} X + \cdots + e_{n-1} X^{n-k-1}.$$

Since 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

$$\begin{aligned} X^k \mathbf{s}^{(n-k)}(X) &= \mathbf{e}(X) \\ &= e_k X^k + r_{k+1} X^{k+1} + \cdots + e_{n-1} X^{n-1}. \end{aligned}$$

*This 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 vector is the transmitted code vector.

Suppose that errors are not confined to the $n - k$ high-order positions but are confined to $n - k$ consecutive positions, say $X^i, X^{i+1}, \ldots, X^{(n-k)+i-1}$, 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 premultiplying $\mathbf{r}(X)$ 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, $X^k, X^{k+1}, \ldots, X^{n-1}$ 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*. 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, error correction can be accomplished by simply 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 $n - k$*

Ex.1020 / Page 145 of 305
Sandisk Technologies, Inc. et al

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 can be shown as follows. An error pattern $\mathbf{e}(X)$ with t or fewer errors which are confined to $n - k$ consecutive positions must be of the form $\mathbf{e}(X) = X^j B(X)$, where $B(X)$ 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 certain number of cyclic shifts of $\mathbf{e}(X)$.] Dividing $\mathbf{e}(X)$ by the generator polynomial $\mathbf{g}(X)$, we have

$$X^j B(X) = \mathbf{a}(X)\mathbf{g}(X) + \mathbf{s}(X),$$

where $\mathbf{s}(X)$ is the syndrome of $X^j B(X)$. Since $\mathbf{s}(X) + X^j B(X)$ is a multiple of $\mathbf{g}(X)$, it is a code polynomial. The syndrome $\mathbf{s}(X)$ cannot have weight t or less unless $\mathbf{s}(X) = X^j B(X)$. Suppose that the weight of $\mathbf{s}(X)$ is t or less and $\mathbf{s}(X) \neq X^j B(X)$. Then $\mathbf{s}(X) + X^j B(X)$ is a nonzero code vector 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 described above, an error-trapping decoder can be implemented as shown in Figure 5.1. The decoding operation can be described in the following steps:

Step 1. The received polynomial $\mathbf{r}(X)$ is shifted into the buffer and syndrome registers simultaneously with gates 1 and 3 turned on and all the other gates turned off. Since we are only interested in the recovery of the k information digits, the buffer register has only to store the k received information digits.

Step 2. As soon as the entire $\mathbf{r}(X)$ 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,

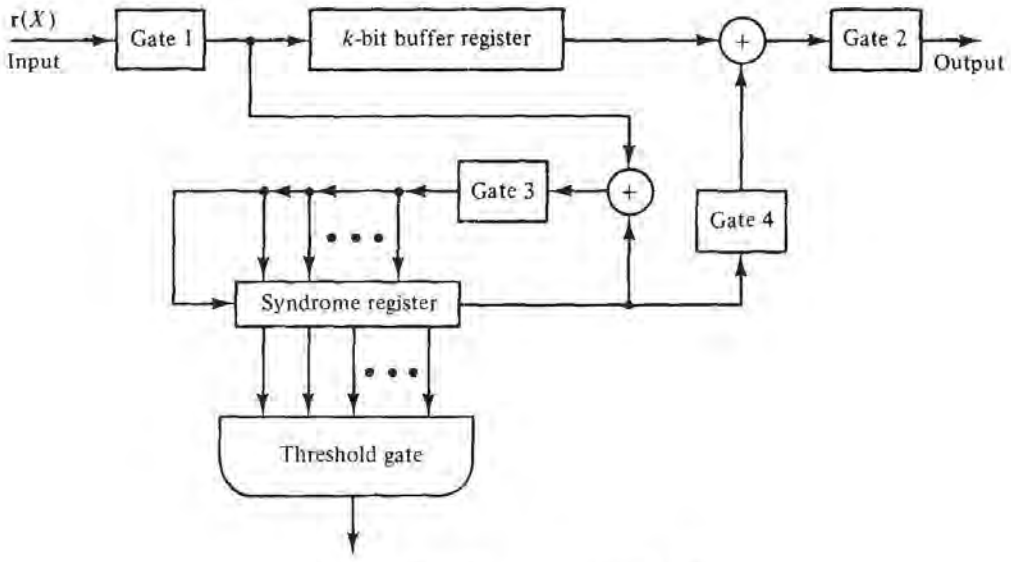


Figure 5.1 Error-trapping decoder.

Ex.1020 / Page 146 of 305
Sandisk Technologies, Inc. et al

$X^{k+1}, \ldots, X^{n-1}$ of $\mathbf{r}(X)$. Now, gates 2 and 4 are turned on and the other gates 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. (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. Cyclically shift the syndrome register 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 $X^{k-1}, X^k, \ldots, X^{n-2}$ of $\mathbf{r}(X)$ and the contents in the syndrome register are identical to the errors at these locations. Since 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 goes down to t or less. If the weight goes down to t or less at the end of the ith shift, for $1 \leq i \leq k$, the first i received digits, $r_{n-i}, r_{n-i+1}, \ldots, r_{n-1}$, in the buffer register are *error-free* and the contents in the syndrome register are identical to the errors at the locations of $X^{k-i}, X^{k-i+1}, \ldots, X^{n-i-1}$. 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 will be 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 that 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. We keep on shifting the syndrome register. 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, $X^{n-l}, X^{n-l+1}, \ldots, X^{n-1}, X^0, X^1, \ldots, X^{n-k-l-1}$ 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 $X^{n-l}, X^{n-l+1}, \ldots, X^{n-1}$ of $\mathbf{r}(X)$. Since we do not need the errors at the $n - k - l$ parity locations, we shift the syndrome register $n - k - l$ times with all the gates turned off. Now, the l errors at the locations $X^{n-l}, X^{n-l+1}, \ldots, X^{n-1}$ 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.

If the weight of the syndrome never goes down to t or less by the time the syndrome register has been shifted a total of n times, either an uncorrectable error

Ex.1020 / Page 147 of 305
Sandisk Technologies, Inc. et al

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 $\mathbf{r}(X)$, 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 and it causes undesirable 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 achieved by shifting the received vector $\mathbf{r}(X)$ into the syndrome register from the *left* end as shown in Figure 5.2. This variation is based on the following facts. If the errors are confined to $n - k$

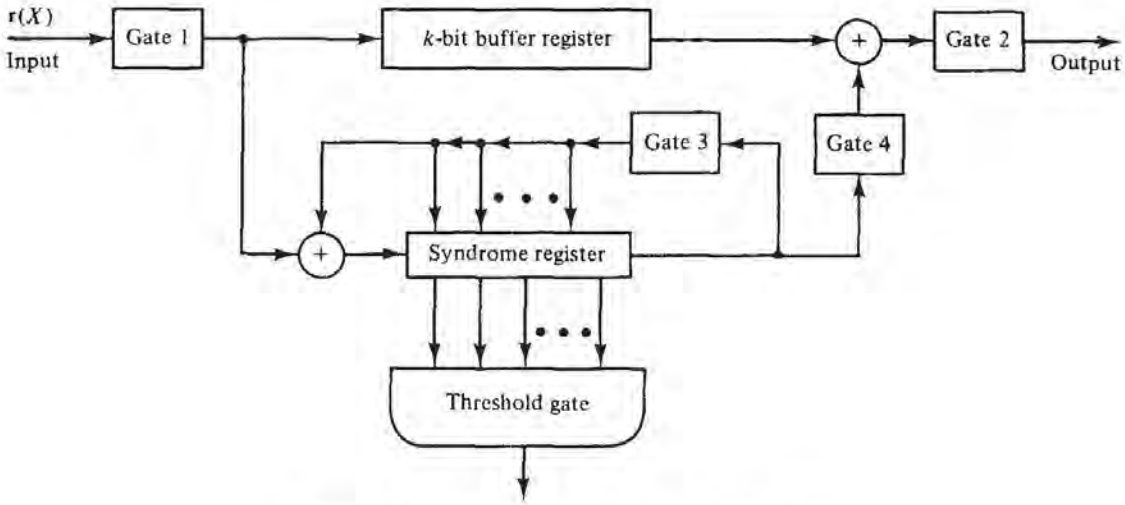


Figure 5.2 Another error-trapping decoder.

low-order parity positions $X^0, X^1, \ldots, X^{n-k-1}$ 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 $X^0, X^1, \ldots, X^{n-k-1}$ 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 $X^i, X^{i+1}, \ldots, X^{(n-k)+i-1}$. 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, $X^i, X^{i+1}, \ldots, X^{(n-k)+i-1}$ of $\mathbf{r}(X)$. The operation of the decoder shown in Figure 5.2 is described as follows:

Step 1. Gates 1 and 3 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 $\mathbf{r}(X)$.

Ex.1020 / Page 148 of 305
Sandisk Technologies, Inc. et al

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 $X^0, X^1, \ldots, X^{n-k-1}$ 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 positions $X^{n-1}, X^0, X^1, \ldots, X^{n-k-2}$ 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 $X^0, X^1, \ldots, X^{n-k-2}$ 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 $(0 \quad 0 \quad \cdots \quad 0 \quad 1)$. 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 goes down to t or less. If the weight goes down to t or less after the ith 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 goes down 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 goes down 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 error-trapping decoding described above, 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 5.1. However, when $n-k$ is much smaller

Ex.1020 / Page 149 of 305
Sandisk Technologies, Inc. et al

than k, the first implementation of error trapping is more advantageous in decoding speed than the one shown in Figure 5.2.

The decoding of cyclic Hamming codes presented in Chapter 4 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 codes and burst-error-correcting codes (decoding of burst-error-correcting codes is discussed in Chapter 9). It is also effective for decoding some short double-error-correcting codes. When it 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 will be sacrificed. Several refinements of this simple decoding technique [1–9] have been devised to extend its application to multiple-error-correcting codes. One of the refinements is presented in the next section.

Example 5.1

Consider the (15, 7) cyclic code generated by $\mathbf{g}(X) = 1 + X^4 + X^6 + X^7 + X^8$. This code has minimum distance $d_{\min} = 5$, 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. 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 5.3, 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) generated by $\mathbf{g}(X) = 1 + X^4 + X^6 + X^7 + X^8$ is shown in Figure 5.4.

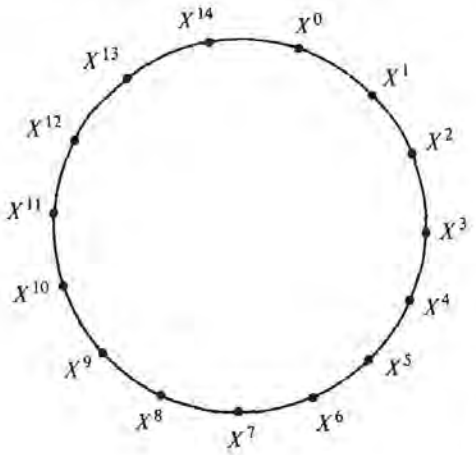


Figure 5.3 Ring arrangement of code digit positions.

5.2 IMPROVED ERROR-TRAPPING DECODING

The error-trapping decoding discussed in Section 5.1 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 equipment. The complexity of the additional equipment depends on how many errors outside an $(n - k)$-digit span are to be corrected. An improvement proposed by Kasami [5] is discussed here.

Ex.1020 / Page 150 of 305
Sandisk Technologies, Inc. et al

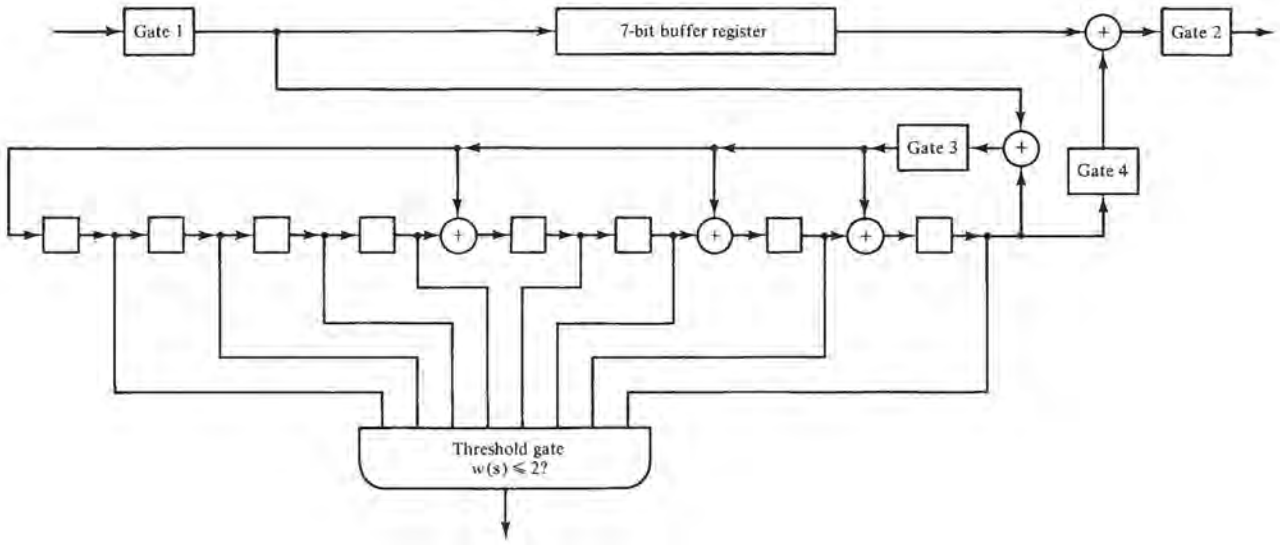


Figure 5.4 Error-trapping decoder for the (15, 7) cyclic code generated by $g(X) = 1 + X^4 + X^6 + X^7 + X^8$.

The error pattern $\mathbf{e}(X) = e_0 + e_1X + e_2X^2 + \cdots + e_{n-1}X^{n-1}$, which corrupted the transmitted code vector, can be divided into two parts:

$$\mathbf{e}_p(X) = e_0 + e_1X + \cdots + e_{n-k-1}X^{n-k-1}$$
$$\mathbf{e}_I(X) = e_{n-k}X^{n-k} + \cdots + e_{n-1}X^{n-1},$$

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

$$\mathbf{e}_I(X) = \mathbf{q}(X)\mathbf{g}(X) + \boldsymbol{\rho}(X), \tag{5.1}$$

where $\boldsymbol{\rho}(X)$ is the remainder with degree $n - k - 1$ or less. Adding $\mathbf{e}_p(X)$ to both sides of (5.1), we obtain

$$\mathbf{e}(X) = \mathbf{e}_p(X) + \mathbf{e}_I(X) = \mathbf{q}(X)\mathbf{g}(X) + \boldsymbol{\rho}(X) + \mathbf{e}_p(X). \tag{5.2}$$

Since $\mathbf{e}_p(X)$ has degree $n - k - 1$ or less, $\boldsymbol{\rho}(X) + \mathbf{e}_p(X)$ must be the remainder resulting from dividing the error pattern $\mathbf{e}(X)$ by the generator polynomial. Thus, $\boldsymbol{\rho}(X) + \mathbf{e}_p(X)$ is equal to the syndrome of the received vector $\mathbf{r}(X)$,

$$\mathbf{s}(X) = \boldsymbol{\rho}(X) + \mathbf{e}_p(X). \tag{5.3}$$

Rearranging (5.3), we have

$$\mathbf{e}_p(X) = \mathbf{s}(X) + \boldsymbol{\rho}(X). \tag{5.4}$$

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 $[\boldsymbol{\phi}_j(X)]_{j=1}^{N}$ of degree $k - 1$ or less, such that, for any correctable error pattern $\mathbf{e}(X)$, there is one polynomial $\boldsymbol{\phi}_j(X)$ such that $X^{n-k}\boldsymbol{\phi}_j(X)$ matches the message section of $\mathbf{e}(X)$ or the message section of a cyclic shift of $\mathbf{e}(X)$. The polynomials $\boldsymbol{\phi}_j(X)$'s are called the *covering polynomials*. Let $\boldsymbol{\rho}_j(X)$ be the remainder resulting from dividing $X^{n-k}\boldsymbol{\phi}_j(X)$ by the generator polynomial $\mathbf{g}(X)$ of the code.

The decoding procedure can be described in the following steps:

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 $\mathbf{s}(X) + \boldsymbol{\rho}_j(X)$ for each $j = 1, 2, \ldots, N$ (i.e., $w[\mathbf{s}(X) + \boldsymbol{\rho}_j(X)]$ for $j = 1, 2, \ldots, N$).

Step 3. If, for some l,

$$w[\mathbf{s}(X) + \boldsymbol{\rho}_l(X)] \leq t - w[\boldsymbol{\phi}_l(X)],$$

then $X^{n-k}\boldsymbol{\phi}_l(X)$ matches the error pattern in the message section of $\mathbf{e}(X)$ and $\mathbf{s}(X) + \boldsymbol{\rho}_l(X)$ matches the error pattern in the parity section of $\mathbf{e}(X)$. Thus,

$$\mathbf{e}(X) = \mathbf{s}(X) + \boldsymbol{\rho}_l(X) + X^{n-k}\boldsymbol{\phi}_l(X).$$

Correction is then accomplished 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 $\mathbf{s}(X) + \boldsymbol{\rho}_j(X)$ for $j = 1, 2, \ldots, N$.

Step 4. If $w[\mathbf{s}(X) + \boldsymbol{\rho}_j(X)] > t - w[\boldsymbol{\phi}_j(X)]$ for all $j = 1, 2, \ldots, N$, both syndrome and buffer registers are shifted cyclically once. Then the new con-

Ex.1020 / Page 152 of 305
Sandisk Technologies, Inc. et al

tents $\mathbf{s}^{(1)}(X)$ of the syndrome register 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 $\mathbf{s}^{(1)}(X) + \boldsymbol{\rho}_j(X)$ is computed for $j = 1, 2, \ldots, N$. If, for some l,

$$w[\mathbf{s}^{(1)}(X) + \boldsymbol{\rho}_l(X)] \leq t - w[\boldsymbol{\phi}_l(X)],$$

then $X^{n-k}\boldsymbol{\phi}_l(X)$ matches the errors in the message section of $\mathbf{e}^{(1)}(X)$ and $\mathbf{s}^{(1)}(X) + \boldsymbol{\rho}_l(X)$ matches the errors in the parity section of $\mathbf{e}^{(1)}(X)$. Thus,

$$\mathbf{e}^{(1)}(X) = \mathbf{s}^{(1)}(X) + \boldsymbol{\rho}_l(X) + X^{n-k}\boldsymbol{\phi}_l(X).$$

Correction is then accomplished by taking the modulo-2 sum $\mathbf{r}^{(1)}(X) + \mathbf{e}^{(1)}(X)$. If

$$w[\mathbf{s}^{(1)}(X) + \boldsymbol{\rho}_j(X)] > t - w[\boldsymbol{\phi}_j(X)]$$

for all $j = 1, 2, \ldots, N$, 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 ith shift) is found such that, for some l,

$$w[\mathbf{s}^{(i)}(X) + \boldsymbol{\rho}_l(X)] \leq t - w[\boldsymbol{\phi}_l(X)].$$

Then

$$\mathbf{e}^{(i)}(X) = \mathbf{s}^{(i)}(X) + \boldsymbol{\rho}_l(X) + X^{n-k}\boldsymbol{\phi}_l(X),$$

where $\mathbf{e}^{(i)}(X)$ is the ith cyclic shift of $\mathbf{e}(X)$. If the weight $w[\mathbf{s}^{(i)}(X) + \boldsymbol{\rho}_j(X)]$ never goes down to $t - w[\boldsymbol{\phi}_j(X)]$ 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 described above depends on N, the number of covering polynomials in $\{\boldsymbol{\phi}_j(X)\}_{j=1}^{N}$. The combinational logical circuitry consists of N $(n - k)$-input threshold gates. To find the set of covering polynomials $\{\boldsymbol{\phi}(_jX)\}_{j=1}^{N}$ for a specific code is not an easy problem. Several methods for finding this set can be found in References 5, 10, and 11.

This improved error-trapping method is applicable to many double- and triple-error-correcting codes. However, it is still only applicable 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 References 4, 7, and 8.

5.3 THE GOLAY CODE

The (23, 12) Golay code is the only known multiple-error-correcting binary perfect code which is capable of correcting any combination of three or fewer random errors in a block of 23 digits. This code has abundant and beautiful algebraic struc-

Ex.1020 / Page 153 of 305
Sandisk Technologies, Inc. et al

ture. Since its discovery by Golay in 1949 [12], 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 recent book by MacWilliams and Sloane [13] presents a thorough coverage of this code and its cousins [the extended (23, 12) Golay code and two ternary Golay codes], where a whole chapter is devoted to their algebraic structure. Besides its beautiful structure, the (23, 12) Golay code has been used in several real communication systems.

The (23, 12) Golay code is either generated 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 5.1 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 pattern $\mathbf{e}(X) = X^{11} + X^{22}$. The two errors are never confined to $n - k = 11$ consecutive positions, no matter how many times we cyclically shift $\mathbf{e}(X)$. 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, using the simple error-trapping scheme for decoding the Golay code, 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.

Kasami Decoder [5]

The Golay code can be easily decoded by Kasami's error-trapping technique. The set of polynomials $\{\boldsymbol{\phi}_j(X)\}_{j=1}^{N}$ is chosen as follows:

$$\boldsymbol{\phi}_1(X) = 0, \qquad \boldsymbol{\phi}_2(X) = X^5, \qquad \boldsymbol{\phi}_3(X) = X^6.$$

Let $\mathbf{g}_1(X) = 1 + X^2 + X^4 + X^5 + X^6 + X^{10} + X^{11}$ be the generator polynomial. Dividing $X^{11}\boldsymbol{\phi}_j(X)$ by $\mathbf{g}_1(X)$ for $j = 1, 2, 3$, we obtain the following remainders:

$$\boldsymbol{\rho}_1(X) = 0,$$
$$\boldsymbol{\rho}_2(X) = X + X^2 + X^5 + X^6 + X^8 + X^9,$$
$$\boldsymbol{\rho}_3(X) = X\boldsymbol{\rho}_2(X) = X^2 + X^3 + X^6 + X^7 + X^9 + X^{10}.$$

A decoder based on the Kasami's error-trapping scheme is shown in Figure 5.5. The received vector $\mathbf{r}(X) = r_0 + r_1X + r_2X^2 + \cdots + r_{22}X^{22}$ 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

Ex.1020 / Page 154 of 305
Sandisk Technologies, Inc. et al

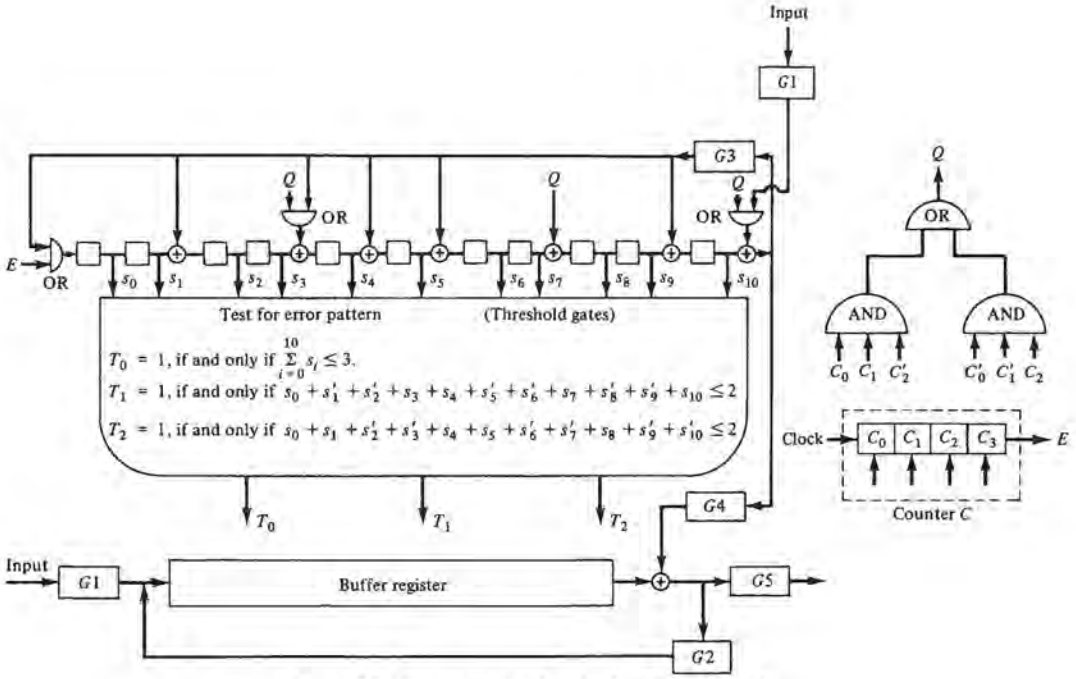


Figure 5.5 Error-trapping decoder for the Golay code.

high-order positions $X^{12}, X^{13}, \ldots, X^{22}$ of $\mathbf{r}(X)$, the syndrome matches the errors in those positions. The error-correction procedure of this decoder is described in the following steps:

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 $\mathbf{s}(X) = s_0 + s_1X + \cdots + s_{10}X^{10}$ 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 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 $w[\mathbf{s}(X)] > 3$, the weight of $\mathbf{s}(X) + \boldsymbol{\rho}_2(X)$ is tested. If $w[\mathbf{s}(X) + \boldsymbol{\rho}_2(X)] \leq 2$, then $\mathbf{s}(X) + \boldsymbol{\rho}_2(X) = s_0 + s_1'X + s_2'X^2 + s_3X^3 + s_4X^4 + s_5'X^5 + s_6'X^6 + s_7X^7 + s_8'X^8 + s_9'X^9 + s_{10}X^{10}$ 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 Q, which is 1 when and only when C counts 3 and 4, is fed into the syndrome register to form the error pattern $\mathbf{s}(X) + \boldsymbol{\rho}_2(X)$. 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 $w[\mathbf{s}(X) + \boldsymbol{\rho}_2(X)] > 2$, the weight of $\mathbf{s}(X) + \boldsymbol{\rho}_3(X)$ is tested. If $w[\mathbf{s}(X) + \boldsymbol{\rho}_3(X)] \leq 2$, then $\mathbf{s}(X) + \boldsymbol{\rho}_3(X) = s_0 + s_1X + s_2'X^2 + s_3'X^3 + s_4X^4 + s_5X^5 + s_6'X^6 + s_7'X^7 + s_8X^8 + s_9'X^9 + s_{10}'X^{10}$ is identical to the error pattern in the 11 high-order positions of the received word and a single error occurs at positions X^6. The correction is the same as step (b), except that counter C starts to count from 3. If $w[\mathbf{s}(X)] > 3$, $w[\mathbf{s}(X) + \boldsymbol{\rho}_2(X)] > 2$, and $w[\mathbf{s}(X) + \boldsymbol{\rho}_3(X)] > 2$, 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.

Ex.1020 / Page 156 of 305
Sandisk Technologies, Inc. et al

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 code vector. 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 code vector.

Systematic Search Decoder [14]

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 can be described 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. Since not all the errors are in the parity-check section, an error must be corrected in this manner.

In every pattern of 3 or fewer errors, there is at least one error which, 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 to 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 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 [15]

$$A(z) = 1 + 253z^7 + 506z^8 + 1288z^{11} + 1288z^{12} + 506z^{15} + 253z^{16} + z^{23}.$$

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} [16] [i.e., $P_u(E) \leq 2^{-11}$].

Ex.1020 / Page 157 of 305
Sandisk Technologies, Inc. et al

PROBLEMS

5.1. 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.

5.2. **(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?

5.3. 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 5.5 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.]

5.4. Use the decoder of Figure 5.5 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 in the syndrome register.

5.5. 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 $\mathbf{p}(X) = 1 + X + X^4$, which is a primitive polynomial of degree 4. Determine the smallest integer n such that $\mathbf{g}(X) = (X^3 + 1)\mathbf{p}(X)$ divides $X^n + 1$.

REFERENCES

1. T. Kasami, "A Decoding Method for Multiple-Error-Correcting Cyclic Codes by Using Threshold Logics," Conv. Rec. Inf. Process. Soc. Jap. (in Japanese), Tokyo, November 1961.
2. M. E. Mitchell et al., "Coding and Decoding Operation Research," G. E. Advanced

Ex.1020 / Page 158 of 305
Sandisk Technologies, Inc. et al

Electronics Final Report on Contract AF 19 (604)-6183, Air Force Cambridge Research Labs., Cambridge, Mass., 1961.

3. M. E. Mitchell, "Error-Trap Decoding of Cyclic Codes," G. E. Report No. 62MCD3, General Electric Military Communications Dep., Oklahoma City, Okla., December 1962.
4. L. Rudolph, "Easily Implemented Error-Correction Encoding–Decoding," G. E. Report No. 62MCD2, General Electric Corporation, Oklahoma City, Okla., December 1962.
5. T. Kasami, "A Decoding Procedure For Multiple-Error-Correction Cyclic Codes," *IEEE Trans. Inf. Theory*, IT-10, pp. 134–139, April 1964.
6. E. Prange, "The Use of Coset Equivalence in the Analysis and Decoding of Group Codes," AFCRC-TR-59-164, Air Force Cambridge Research Labs., Cambridge, Mass., June 1959.
7. E. Prange, "The Use of Information Sets in Decoding Cyclic Codes," *IEEE Trans. Inf. Theory*, IT-8, pp. 85–89, September 1962.
8. F. J. MacWilliams, "Permutation Decoding of Systematic Codes," *Bell Syst. Tech. J.*, 43, Part I, pp. 485–505, January 1964.
9. L. Rudolph and M. E. Mitchell, "Implementation of Decoders for Cyclic Codes," *IEEE Trans. Inf. Theory*, IT-10, pp. 259–260, July 1964.
10. D. C. Foata, "On a Program for Ray-Chaudhuri's Algorithm for a Minimum Cover of an Abstract Complex," *Commun. ACM*, 4, pp. 504–506, November 1961.
11. I. B. Pyne and E. J. McCluskey, "The Reduction of Redundancy in Solving Prime Implicant Tables," *IRE Trans. Electron. Comput.*, EC-11, pp. 473–482, August 1962.
12. M. J. E. Golay, "Notes on Digital Coding," *Proc. IRE*, 37, p. 657, June 1949.
13. F.J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, 1977.
14. E. J. Weldon, Jr., "A Comparison of an Interleaved Golay Code and a Three-Dimensional Product Code," Final Report, USNELC Contract N0095368M5345, August 1968.
15. W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2nd ed., MIT Press, Cambridge, Mass., 1972.
16. 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. Inf. Theory*, IT-25 (1), pp. 110–112, January 1979.

Ex.1020 / Page 159 of 305
Sandisk Technologies, Inc. et al

6

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]. Generalization of the binary BCH codes to codes in p^m symbols (where p is a prime) was obtained 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 introduced by Reed and Solomon in 1960 [5] independently of the works by Hocquenghem, Bose, and Chaudhuri.

The first decoding algorithm for binary BCH codes was devised by Peterson in 1960 [3]. Since then, Peterson's algorithm has been 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 which is the most important subclass from the standpoint of both theory and implementation. For nonbinary BCH codes, we discuss only the Reed–Solomon codes. For a detailed description of the BCH codes, their algebraic properties and decoding algorithms, the reader is referred to References 9 and 13 to 15.

Ex.1020 / Page 160 of 305
Sandisk Technologies, Inc. et al

6.1 DESCRIPTION OF THE 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:

Block length:	$n = 2^m - 1$
Number of parity-check digits:	$n - k \leq mt$
Minimum distance:	$d_{\min} \geq 2t + 1.$

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) which has

$$\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2t} \tag{6.1}$$

as its roots [i.e., $\mathbf{g}(\alpha^i) = 0$ for $1 \leq i \leq 2t$]. It follows from Theorem 2.7 that $\mathbf{g}(X)$ has $\alpha, \alpha^2, \ldots, \alpha^{2t}$ 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* of $\phi_1(X), \phi_2(X), \ldots, \phi_{2t}(X)$, that is,

$$\mathbf{g}(X) = \text{LCM}\,\{\phi_1(X), \phi_2(X), \ldots, \phi_{2t}(X)\}. \tag{6.2}$$

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

$$\mathbf{g}(X) = \text{LCM}\,\{\phi_1(X), \phi_3(X), \ldots, \phi_{2t-1}(X)\}. \tag{6.3}$$

Since 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, 16]. 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 defined above are usually called *primitive* (or *narrow-sense*) BCH codes.

From (6.3), we see that the single-error-correcting BCH code of length $2^m - 1$ is generated by

$$\mathbf{g}(X) = \phi_1(X).$$

Since 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.

Ex.1020 / Page 161 of 305
Sandisk Technologies, Inc. et al

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	255	163	12	511	268	29
15	11	1		155	13		259	30
	7	2		147	14		250	31
	5	3		139	15		241	36
31	26	1		131	18		238	37
	21	2		123	19		229	38
	16	3		115	21		220	39
	11	5		107	22		211	41
	6	7		99	23		202	42
63	57	1		91	25		193	43
	51	2		87	26		184	45
	45	3		79	27		175	46
	39	4		71	29		166	47
	36	5		63	30		157	51
	30	6		55	31		148	53
	24	7		47	42		139	54
	18	10		45	43		130	55
	16	11		37	45		121	58
	10	13		29	47		112	59
	7	15		21	55		103	61
127	120	1		13	59		94	62
	113	2		9	63		85	63
	106	3	511	502	1		76	85
	99	4		493	2		67	87
	92	5		484	3		58	91
	85	6		475	4		49	93
	78	7		466	5		40	95
	71	9		457	6		31	109
	64	10		448	7		28	111
	57	11		439	8		19	119
	50	13		430	9		10	121
	43	14		421	10	1023	1013	1
	36	15		412	11		1003	2
	29	21		403	12		993	3
	22	23		394	13		983	4
	15	27		385	14		973	5
	8	31		376	15		963	6
255	247	1		367	16		953	7
	239	2		358	18		943	8
	231	3		349	19		933	9
	223	4		340	20		923	10
	215	5		331	21		913	11
	207	6		322	22		903	12
	199	7		313	23		893	13
	191	8		304	25		883	14
	187	9		295	26		873	15
	179	10		286	27		863	16
	171	11		277	28		858	17

Ex.1020 / Page 162 of 305
Sandisk Technologies, Inc. et al

TABLE 6.1 Continued.

n	k	t	n	k	t	n	k	t
1023	848	18	1023	553	52	1023	268	103
	838	19		543	53		258	106
	828	20		533	54		248	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
	768	26		473	61		193	118
	758	27		463	62		183	119
	748	28		453	63		173	122
	738	29		443	73		163	123
	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						

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

$$g(X) = \text{LCM } \{\phi_1(X), \phi_3(X)\}.$$

Since $\phi_1(X)$ and $\phi_3(X)$ are two distinct irreducible polynomials,

$$\begin{aligned} g(X) &= \phi_1(X)\phi_3(X) \\ &= (1 + X + X^4)(1 + X + X^2 + X^3 + X^4) \\ &= 1 + X^4 + X^6 + X^7 + X^8. \end{aligned}$$

Ex.1020 / Page 163 of 305
Sandisk Technologies, Inc. et al

Thus, the code is a (15, 7) cyclic code with $d_{\min} \geq 5$. Since the generator polynomial is 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 $d_{\min} \geq 7$. Since 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 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.

TABLE 6.2 GALOIS FIELD GF(2^6) WITH $p(\alpha) = 1 + \alpha + \alpha^6 = 0$

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	α^3	(0 0 0 1 0 0)
α^4	α^4	(0 0 0 0 1 0)
α^5	α^5	(0 0 0 0 0 1)
α^6	$1 + \alpha$	(1 1 0 0 0 0)
α^7	$\alpha + \alpha^2$	(0 1 1 0 0 0)
α^8	$\alpha^2 + \alpha^3$	(0 0 1 1 0 0)
α^9	$\alpha^3 + \alpha^4$	(0 0 0 1 1 0)
α^{10}	$\alpha^4 + \alpha^5$	(0 0 0 0 1 1)
α^{11}	$1 + \alpha + \alpha^5$	(1 1 0 0 0 1)
α^{12}	$1 + \alpha^2$	(1 0 1 0 0 0)
α^{13}	$\alpha + \alpha^3$	(0 1 0 1 0 0)
α^{14}	$\alpha^2 + \alpha^4$	(0 0 1 0 1 0)
α^{15}	$\alpha^3 + \alpha^5$	(0 0 0 1 0 1)
α^{16}	$1 + \alpha + \alpha^4$	(1 1 0 0 1 0)
α^{17}	$\alpha + \alpha^2 + \alpha^5$	(0 1 1 0 0 1)
α^{18}	$1 + \alpha + \alpha^2 + \alpha^3$	(1 1 1 1 0 0)
α^{19}	$\alpha + \alpha^2 + \alpha^3 + \alpha^4$	(0 1 1 1 1 0)
α^{20}	$\alpha^2 + \alpha^3 + \alpha^4 + \alpha^5$	(0 0 1 1 1 1)
α^{21}	$1 + \alpha + \alpha^3 + \alpha^4 + \alpha^5$	(1 1 0 1 1 1)
α^{22}	$1 + \alpha^2 + \alpha^4 + \alpha^5$	(1 0 1 0 1 1)
α^{23}	$1 + \alpha^3 + \alpha^5$	(1 0 0 1 0 1)
α^{24}	$1 + \alpha^4$	(1 0 0 0 1 0)
α^{25}	$\alpha + \alpha^5$	(0 1 0 0 0 1)
α^{26}	$1 + \alpha + \alpha^2$	(1 1 1 0 0 0)
α^{27}	$\alpha + \alpha^2 + \alpha^3$	(0 1 1 1 0 0)
α^{28}	$\alpha^2 + \alpha^3 + \alpha^4$	(0 0 1 1 1 0)
α^{29}	$\alpha^3 + \alpha^4 + \alpha^5$	(0 0 0 1 1 1)
α^{30}	$1 + \alpha + \alpha^4 + \alpha^5$	(1 1 0 0 1 1)

Ex.1020 / Page 164 of 305
Sandisk Technologies, Inc. et al

TABLE 6.2 Continued.

α^{31}	$1 + \alpha^2 + \alpha^5$	(1 0 1 0 0 1)
α^{32}	$1 + \alpha^3$	(1 0 0 1 0 0)
α^{33}	$\alpha + \alpha^4$	(0 1 0 0 1 0)
α^{34}	$\alpha^2 + \alpha^5$	(0 0 1 0 0 1)
α^{35}	$1 + \alpha + \alpha^3$	(1 1 0 1 0 0)
α^{36}	$\alpha + \alpha^2 + \alpha^4$	(0 1 1 0 1 0)
α^{37}	$\alpha^2 + \alpha^3 + \alpha^5$	(0 0 1 1 0 1)
α^{38}	$1 + \alpha + \alpha^3 + \alpha^4$	(1 1 0 1 1 0)
α^{39}	$\alpha + \alpha^2 + \alpha^4 + \alpha^5$	(0 1 1 0 1 1)
α^{40}	$1 + \alpha + \alpha^2 + \alpha^3 + \alpha^5$	(1 1 1 1 0 1)
α^{41}	$1 + \alpha^2 + \alpha^3 + \alpha^4$	(1 0 1 1 1 0)
α^{42}	$\alpha + \alpha^3 + \alpha^4 + \alpha^5$	(0 1 0 1 1 1)
α^{43}	$1 + \alpha + \alpha^2 + \alpha^4 + \alpha^5$	(1 1 1 0 1 1)
α^{44}	$1 + \alpha^2 + \alpha^3 + \alpha^5$	(1 0 1 1 0 1)
α^{45}	$1 + \alpha^3 + \alpha^4$	(1 0 0 1 1 0)
α^{46}	$\alpha + \alpha^4 + \alpha^5$	(0 1 0 0 1 1)
α^{47}	$1 + \alpha + \alpha^2 + \alpha^5$	(1 1 1 0 0 1)
α^{48}	$1 + \alpha^2 + \alpha^3$	(1 0 1 1 0 0)
α^{49}	$\alpha + \alpha^3 + \alpha^4$	(0 1 0 1 1 0)
α^{50}	$\alpha^2 + \alpha^4 + \alpha^5$	(0 0 1 0 1 1)
α^{51}	$1 + \alpha + \alpha^3 + \alpha^5$	(1 1 0 1 0 1)
α^{52}	$1 + \alpha^2 + \alpha^4$	(1 0 1 0 1 0)
α^{53}	$\alpha + \alpha^3 + \alpha^5$	(0 1 0 1 0 1)
α^{54}	$1 + \alpha + \alpha^2 + \alpha^4$	(1 1 1 0 1 0)
α^{55}	$\alpha + \alpha^2 + \alpha^3 + \alpha^5$	(0 1 1 1 0 1)
α^{56}	$1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4$	(1 1 1 1 1 0)
α^{57}	$\alpha + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^5$	(0 1 1 1 1 1)
α^{58}	$1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^5$	(1 1 1 1 1 1)
α^{59}	$1 + \alpha^2 + \alpha^3 + \alpha^4 + \alpha^5$	(1 0 1 1 1 1)
α^{60}	$1 + \alpha^3 + \alpha^4 + \alpha^5$	(1 0 0 1 1 1)
α^{61}	$1 + \alpha^4 + \alpha^5$	(1 0 0 0 1 1)
α^{62}	$1 + \alpha^5$	(1 0 0 0 0 1)

$\alpha^{63} = 1$

TABLE 6.3 MINIMAL POLYNOMIALS OF THE ELEMENTS IN GF(2^6)

Elements	Minimal polynomials
$\alpha, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}, \alpha^{32}$	$1 + X + X^6$
$\alpha^3, \alpha^6, \alpha^{12}, \alpha^{24}, \alpha^{48}, \alpha^{33}$	$1 + X + X^2 + X^4 + X^6$
$\alpha^5, \alpha^{10}, \alpha^{20}, \alpha^{40}, \alpha^{17}, \alpha^{34}$	$1 + X + X^2 + X^5 + X^6$
$\alpha^7, \alpha^{14}, \alpha^{28}, \alpha^{56}, \alpha^{49}, \alpha^{35}$	$1 + X^3 + X^6$
$\alpha^9, \alpha^{18}, \alpha^{36}$	$1 + X^2 + X^3$
$\alpha^{11}, \alpha^{22}, \alpha^{44}, \alpha^{25}, \alpha^{50}, \alpha^{37}$	$1 + X^2 + X^3 + X^5 + X^6$
$\alpha^{13}, \alpha^{26}, \alpha^{52}, \alpha^{41}, \alpha^{19}, \alpha^{38}$	$1 + X + X^3 + X^4 + X^6$
$\alpha^{15}, \alpha^{30}, \alpha^{60}, \alpha^{57}, \alpha^{51}, \alpha^{39}$	$1 + X^2 + X^4 + X^5 + X^6$
α^{21}, α^{42}	$1 + X + X^2$
$\alpha^{23}, \alpha^{46}, \alpha^{29}, \alpha^{58}, \alpha^{53}, \alpha^{43}$	$1 + X + X^4 + X^5 + X^6$
$\alpha^{27}, \alpha^{54}, \alpha^{45}$	$1 + X + X^3$
$\alpha^{31}, \alpha^{62}, \alpha^{61}, \alpha^{59}, \alpha^{55}, \alpha^{47}$	$1 + X^5 + X^6$

Ex.1020 / Page 165 of 305
Sandisk Technologies, Inc. et al

TABLE 6.4 GENERATOR POLYNOMIALS OF ALL THE BCH CODES OF LENGTH 63

n	k	t	$g(X)$
63	57	1	$g_1(X) = 1 + X + X^6$
	51	2	$g_2(X) = (1 + X + X^6)(1 + X + X^2 + X^4 + X^6)$
	45	3	$g_3(X) = (1 + X + X^2 + X^5 + X^6)g_2(X)$
	39	4	$g_4(X) = (1 + X^3 + X^6)g_3(X)$
	36	5	$g_5(X) = (1 + X^2 + X^3)g_4(X)$
	30	6	$g_6(X) = (1 + X^2 + X^3 + X^5 + X^6)g_5(X)$
	24	7	$g_7(X) = (1 + X + X^3 + X^4 + X^6)g_6(X)$
	18	10	$g_{10}(X) = (1 + X^2 + X^4 + X^5 + X^6)g_7(X)$
	16	11	$g_{11}(X) = (1 + X + X^2)g_{10}(X)$
	10	13	$g_{13}(X) = (1 + X + X^4 + X^5 + X^6)g_{11}(X)$
	7	15	$g_{15}(X) = (1 + X + X^3)g_{13}(X)$

It follows from the definition of a t-error-correcting BCH code of length $n = 2^m - 1$ that each code polynomial has $\alpha, \alpha^2, \ldots, \alpha^{2t}$ and their conjugates as roots. Now, let $\mathbf{v}(X) = v_0 + v_1X + \cdots + v_{n-1}X^{n-1}$ be a polynomial with coefficients from GF(2). If $\mathbf{v}(X)$ has $\alpha, \alpha^2, \ldots, \alpha^{2t}$ as roots, it follows from Theorem 2.10 that $\mathbf{v}(X)$ is divisible by the minimal polynomials $\phi_1(X), \phi_2(X), \ldots, \phi_{2t}(X)$ of $\alpha, \alpha^2, \ldots, \alpha^{2t}$. Obviously, $\mathbf{v}(X)$ is divisible by their least common multiple (the generator polynomial),

$$g(X) = \text{LCM}\{\phi_1(X), \phi_2(X), \ldots, \phi_{2t}(X)\}.$$

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 $\mathbf{v} = (v_0, v_1, v_2, \ldots, v_{n-1})$ is a code word if and only if the polynomial $\mathbf{v}(X) = v_0 + v_1X + \cdots + v_{n-1}X^{n-1}$ has $\alpha, \alpha^2, \ldots, \alpha^{2t}$ as roots. This definition is useful in proving the minimum distance of the code.

Let $\mathbf{v}(X) = v_0 + v_1X + \cdots + v_{n-1}X^{n-1}$ be a code polynomial in a t-error-correcting BCH code of length $n = 2^m - 1$. Since α^i is a root of $\mathbf{v}(X)$ for $1 \leq i \leq 2t$, then

$$\mathbf{v}(\alpha^i) = v_0 + v_1\alpha^i + v_2\alpha^{2i} + \cdots + v_{n-1}\alpha^{(n-1)i} = 0 \tag{6.4}$$

This equality can be written as a matrix product as follows:

$$(v_0, v_1, \ldots, v_{n-1}) \cdot \begin{bmatrix} 1 \\ \alpha^i \\ \alpha^{2i} \\ \cdot \\ \cdot \\ \cdot \\ \alpha^{(n-1)i} \end{bmatrix} = 0 \tag{6.5}$$

for $1 \leq i \leq 2t$. The condition given by (6.5) simply says that the inner product of $(v_0, v_1, \ldots, v_{n-1})$ and $(1, \alpha^i, \alpha^{2i}, \ldots, \alpha^{(n-1)i})$ is equal to zero. Now we form the following matrix:

Ex.1020 / Page 166 of 305
Sandisk Technologies, Inc. et al

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{n-1} \\ 1 & (\alpha^2) & (\alpha^2)^2 & (\alpha^2)^3 & \cdots & (\alpha^2)^{n-1} \\ 1 & (\alpha^3) & (\alpha^3)^2 & (\alpha^3)^3 & \cdots & (\alpha^3)^{n-1} \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ 1 & (\alpha^{2t}) & (\alpha^{2t})^2 & (\alpha^{2t})^3 & \cdots & (\alpha^{2t})^{n-1} \end{bmatrix}. \tag{6.6}$$

It follows from (6.5) that if $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ is a code word in the t-error-correcting BCH code, then

$$\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}. \tag{6.7}$$

On the other hand, if an n-tuple $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ 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 code word in the t-error-correcting BCH code. Hence, the code is the null space of the matrix $\mathbf{H}$, and $\mathbf{H}$ is the 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 2.7). This says that if the inner product of $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ and the ith row of $\mathbf{H}$ is zero, the inner product of $\mathbf{v}$ and the jth row of $\mathbf{H}$ is also zero. For this reason, the jth 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:

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{n-1} \\ 1 & (\alpha^3) & (\alpha^3)^2 & (\alpha^3)^3 & \cdots & (\alpha^3)^{n-1} \\ 1 & (\alpha^5) & (\alpha^5)^2 & (\alpha^5)^3 & \cdots & (\alpha^5)^{n-1} \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ \cdot & & & & & \cdot \\ 1 & (\alpha^{2t-1}) & (\alpha^{2t-1})^2 & (\alpha^{2t-1})^3 & \cdots & (\alpha^{2t-1})^{n-1} \end{bmatrix}. \tag{6.8}$$

Note that the entries of $\mathbf{H}$ are elements in GF(2^m). Each element in GF(2^m) can be represented by a m-tuple over GF(2). If each entry of $\mathbf{H}$ is replaced by 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

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & \alpha^{15} & \alpha^{18} & \alpha^{21} & \alpha^{24} & \alpha^{27} & \alpha^{30} & \alpha^{33} & \alpha^{36} & \alpha^{39} & \alpha^{42} \end{bmatrix}$$

[use (6.8)]. Using Table 2.8, the fact that $\alpha^{15} = 1$, and representing each entry of $\mathbf{H}$ by its corresponding 4-tuple, we obtain the following binary parity-check matrix for the code:

Ex.1020 / Page 167 of 305
Sandisk Technologies, Inc. et al

$$\mathbf{H} = \left[\begin{array}{cccccccccccccc} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{array}\right].$$

Now we are ready to prove that the t-error-correcting BCH code defined above indeed has minimum distance 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. Suppose that there exists a nonzero code vector $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ with weight $\delta \leq 2t$. Let $v_{j_1}, v_{j_2}, \ldots, v_{j_\delta}$ be the nonzero components of $\mathbf{v}$ (i.e., $v_{j_1} = v_{j_2} = \ldots = v_{j_\delta} = 1$). Using (6.6) and (6.7), we have

$$\begin{aligned} \mathbf{0} &= \mathbf{v} \cdot \mathbf{H}^T \\ &= (v_{j_1}, v_{j_2}, \ldots, v_{j_\delta}) \cdot \begin{bmatrix} \alpha^{j_1} & (\alpha^2)^{j_1} & \cdots & (\alpha^{2t})^{j_1} \\ \alpha^{j_2} & (\alpha^2)^{j_2} & \cdots & (\alpha^{2t})^{j_2} \\ \alpha^{j_3} & (\alpha^2)^{j_3} & \cdots & (\alpha^{2t})^{j_3} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \alpha^{j_\delta} & (\alpha^2)^{j_\delta} & \cdots & (\alpha^{2t})^{j_\delta} \end{bmatrix} \\ &= (1, 1, \ldots, 1) \cdot \begin{bmatrix} \alpha^{j_1} & (\alpha^{j_1})^2 & \cdots & (\alpha^{j_1})^{2t} \\ \alpha^{j_2} & (\alpha^{j_2})^2 & \cdots & (\alpha^{j_2})^{2t} \\ \alpha^{j_3} & (\alpha^{j_3})^2 & \cdots & (\alpha^{j_3})^{2t} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \alpha^{j_\delta} & (\alpha^{j_\delta})^2 & \cdots & (\alpha^{j_\delta})^{2t} \end{bmatrix}. \end{aligned}$$

The equality above implies the following equality:

$$(1, 1, \ldots, 1) \cdot \begin{bmatrix} \alpha^{j_1} & (\alpha^{j_1})^2 & \cdots & (\alpha^{j_1})^{\delta} \\ \alpha^{j_2} & (\alpha^{j_2})^2 & \cdots & (\alpha^{j_2})^{\delta} \\ \alpha^{j_3} & (\alpha^{j_3})^2 & \cdots & (\alpha^{j_3})^{\delta} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \alpha^{j_\delta} & (\alpha^{j_\delta})^2 & \cdots & (\alpha^{j_\delta})^{\delta} \end{bmatrix} = \mathbf{0}, \tag{6.9}$$

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,

Ex.1020 / Page 168 of 305
Sandisk Technologies, Inc. et al

$$\begin{vmatrix} \alpha^{j_1} & (\alpha^{j_1})^2 & \cdots & (\alpha^{j_1})^\delta \\ \alpha^{j_2} & (\alpha^{j_2})^2 & \cdots & (\alpha^{j_2})^\delta \\ \alpha^{j_3} & (\alpha^{j_3})^2 & \cdots & (\alpha^{j_3})^\delta \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \alpha^{j_\delta} & (\alpha^{j_\delta})^2 & \cdots & (\alpha^{j_\delta})^\delta \end{vmatrix} = 0.$$

Taking out the common factor from each row of the determinant above, we obtain

$$\alpha^{(j_1+j_2+\cdots+j_\delta)} \cdot \begin{vmatrix} 1 & \alpha^{j_1} & \cdots & \alpha^{(\delta-1)j_1} \\ 1 & \alpha^{j_2} & \cdots & \alpha^{(\delta-1)j_2} \\ 1 & \alpha^{j_3} & \cdots & \alpha^{(\delta-1)j_3} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ 1 & \alpha^{j_\delta} & \cdots & \alpha^{(\delta-1)j_\delta} \end{vmatrix} = 0. \tag{6.10}$$

The determinant in the equality above is a *Vandermonde determinant* which 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 code vector $\mathbf{v}$ of weight $\delta \leq 2t$ is *invalid*. This implies that the minimum weight of the t-error-correcting BCH code defined above 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. There are many cases where the true minimum distance of a BCH code is equal to its designed distance. However, there are also cases where 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

$$\beta, \beta^2, \ldots, \beta^{2t}$$

as roots. Let $\psi_1(X), \psi_2(X), \ldots, \psi_{2t}(X)$ be the minimal polynomials of $\beta, \beta^2, \ldots, \beta^{2t}$, respectively. Then

$$\mathbf{g}(X) = \text{LCM}\,\{\psi_1(X), \psi_2(X), \ldots, \psi_{2t}(X)\}.$$

Since $\beta^n = 1, \beta, \beta^2, \ldots, \beta^{2t}$ 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 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(2^5) 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

$$\beta, \beta^2, \beta^3, \beta^4$$

Ex.1020 / Page 169 of 305
Sandisk Technologies, Inc. et al

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 which has as roots the following consecutive powers of β:

$$\beta^{l_0}, \beta^{l_0+1}, \ldots, \beta^{l_0+d_0-2}$$

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

$$\mathbf{g}(X) = \text{LCM}\,\{\psi_0(X), \psi_1(X), \ldots, \psi_{d_0-2}(X)\}$$

and the length of the code is

$$n = \text{LCM}\,\{n_0, n_1, \ldots, n_{d_0-2}\}.$$

The BCH code defined above has minimum distance 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 $[(d_0 - 1)/2]$ 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 which 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 minimum distance at least d_0. This lower bound on the minimum distance is called the *BCH bound.*

In the rest of this chapter, we consider only the primitive BCH codes.

6.2 DECODING OF THE BCH CODES

Suppose that a code word $\mathbf{v}(X) = v_0 + v_1X + v_2X^2 + \cdots + v_{n-1}X^{n-1}$ is transmitted and the transmission errors result in the following received vector:

$$\mathbf{r}(X) = r_0 + r_1X + r_2X^2 + \cdots + r_{n-1}X^{n-1}.$$

Let $\mathbf{e}(X)$ be the error pattern. Then

$$\mathbf{r}(X) = \mathbf{v}(X) + \mathbf{e}(X). \tag{6.11}$$

Ex.1020 / Page 170 of 305
Sandisk Technologies, Inc. et al

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,

$$\mathbf{S} = (S_1, S_2, \ldots, S_{2t}) = \mathbf{r} \cdot \mathbf{H}^T, \tag{6.12}$$

where $\mathbf{H}$ is given by (6.6). From (6.6) and (6.12) we find that the ith component of the syndrome is

$$\begin{aligned} S_i &= \mathbf{r}(\alpha^i) \\ &= r_0 + r_1\alpha^i + r_2\alpha^{2i} + \cdots + r_{n-1}\alpha^{(n-1)i} \end{aligned} \tag{6.13}$$

for $1 \leq i \leq 2t$. Note that the syndrome components are elements in the field GF(2^m). These components can be computed 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 $\phi_i(X)$. Since $\phi_i(\alpha^i) = 0$, we have

$$S_i = \mathbf{r}(\alpha^i) = \mathbf{b}_i(\alpha^i). \tag{6.14}$$

Thus, the syndrome component S_i is 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

$$\mathbf{r} = (1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0)$$

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 $\phi_1(X) = 1 + X + X^4$, the remainder is

$$\mathbf{b}_1(X) = X^2.$$

Dividing $\mathbf{r}(X) = 1 + X^8$ by $\phi_3(X) = 1 + X + X^2 + X^3 + X^4$, the remainder is

$$\mathbf{b}_3(X) = 1 + X^3.$$

Using GF(2^4) given by Table 2.8 and substituting α, α^2, and α^4 into $\mathbf{b}_1(X)$, we obtain

$$S_1 = \alpha^2, \qquad S_2 = \alpha^4, \qquad S_4 = \alpha^8.$$

Substituting α^3 into $\mathbf{b}_3(X)$, we obtain

$$S_3 = 1 + \alpha^9 = 1 + \alpha + \alpha^3 = \alpha^7.$$

Thus,

$$\mathbf{S} = (\alpha^2, \alpha^4, \alpha^7, \alpha^8).$$

Ex.1020 / Page 171 of 305
Sandisk Technologies, Inc. et al

Since $\alpha^1, \alpha^2, \ldots, \alpha^{2t}$ are roots of each code polynomial, $\mathbf{v}(\alpha^i) = 0$ for $1 \leq i \leq 2t$. It follows from (6.11) and (6.13) that we obtain the following relationship between the syndrome components and the error pattern:

$$S_i = \mathbf{e}(\alpha^i) \tag{6.15}$$

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 ν errors at locations $X^{j_1}, X^{j_2}, \ldots, X^{j_\nu}$, that is,

$$\mathbf{e}(X) = X^{j_1} + X^{j_2} + \cdots + X^{j_\nu}, \tag{6.16}$$

where $0 \leq j_1 < j_2 < \cdots < j_\nu < n$. From (6.15) and (6.16) we obtain the following set of equations:

$$\begin{aligned} S_1 &= \alpha^{j_1} + \alpha^{j_2} + \cdots + \alpha^{j_\nu} \\ S_2 &= (\alpha^{j_1})^2 + (\alpha^{j_2})^2 + \cdots + (\alpha^{j_\nu})^2 \\ S_3 &= (\alpha^{j_1})^3 + (\alpha^{j_2})^3 + \cdots + (\alpha^{j_\nu})^3 \\ &\vdots \\ S_{2t} &= (\alpha^{j_1})^{2t} + (\alpha^{j_2})^{2t} + \cdots + (\alpha^{j_\nu})^{2t}, \end{aligned} \tag{6.17}$$

where $\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_\nu}$ are unknown. *Any method for solving these equations is a decoding algorithm for the BCH codes.* Once $\alpha^{j_1}, \alpha^{j_2}, \ldots, \alpha^{j_\nu}$ have been found, the powers $j_1, j_2, \ldots, j_\nu$ 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 less (i.e., $\nu \leq t$), 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 to determine α^{j_l} for $l = 1, 2, \ldots, \nu$ from the syndrome components S_i's.

For convenience, let

$$\beta_l = \alpha^{j_l} \tag{6.18}$$

for $1 \leq l \leq \nu$. We call these elements the *error location numbers* since they tell us the locations of the errors. Now the equations of (6.17) can be expressed in the following form:

$$\begin{aligned} S_1 &= \beta_1 + \beta_2 + \cdots + \beta_\nu \\ S_2 &= \beta_1^2 + \beta_2^2 + \cdots + \beta_\nu^2 \\ &\vdots \\ S_{2t} &= \beta_1^{2t} + \beta_2^{2t} + \cdots + \beta_\nu^{2t}. \end{aligned} \tag{6.19}$$

These $2t$ equations are symmetric functions in $\beta_1, \beta_2, \ldots, \beta_\nu$, which are known as *power-sum symmetric functions.* Now, we define the following polynomial:

$$\begin{aligned} \sigma(X) &= (1 + \beta_1 X)(1 + \beta_2 X) \cdots (1 + \beta_\nu X) \\ &= \sigma_0 + \sigma_1 X + \sigma_2 X^2 + \cdots + \sigma_\nu X^\nu. \end{aligned} \tag{6.20}$$

Ex.1020 / Page 172 of 305
Sandisk Technologies, Inc. et al

The roots of $\sigma(X)$ are $\beta_1^{-1}, \beta_2^{-1}, \ldots, \beta_v^{-1}$, 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:

$$\begin{cases} \sigma_0 = 1 \\ \sigma_1 = \beta_1 + \beta_2 + \cdots + \beta_v \\ \sigma_2 = \beta_1\beta_2 + \beta_2\beta_3 + \cdots + \beta_{v-1}\beta_v \\ \quad\vdots \\ \sigma_v = \beta_1\beta_2\cdots\beta_v. \end{cases} \tag{6.21}$$

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_i's. In fact, they are related to the syndrome components by the following *Newton's identities:*

$$\begin{cases} S_1 + \sigma_1 = 0 \\ S_2 + \sigma_1 S_1 + 2\sigma_2 = 0 \\ S_3 + \sigma_1 S_2 + \sigma_2 S_1 + 3\sigma_3 = 0 \\ \quad\vdots \\ S_v + \sigma_1 S_{v-1} + \cdots + \sigma_{v-1} S_1 + v\sigma_v = 0 \\ S_{v+1} + \sigma_1 S_v + \cdots + \sigma_{v-1} S_2 + \sigma_v S_1 = 0 \\ \quad\vdots \end{cases} \tag{6.22}$$

For 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 functions $\sigma_1, \sigma_2, \ldots, \sigma_v$ from the equations of (6.22), the error-location numbers $\beta_1, \beta_2, \ldots, \beta_v$ 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 $\sigma(X)$ would produce an error pattern with minimum number of errors. If $v \leq t$, this $\sigma(X)$ will give the actual error pattern $\mathbf{e}(X)$. In the following, we describe a procedure to determine the polynomial $\sigma(X)$ of minimum degree which satisfies the first $2t$ equations of (6.22) (since we only know S_1 through S_{2t}).

At this point, it would be appropriate to outline the error-correcting procedure for the BCH codes. The procedure consists of three major steps:

1. Compute the syndrome $\mathbf{S} = (S_1, S_2, \ldots, S_{2t})$ from the received polynomial $\mathbf{r}(X)$.
2. Determine the error-location polynomial $\sigma(X)$ from the syndrome components $S_1, S_2, \ldots, S_{2t}$.

Ex.1020 / Page 173 of 305
Sandisk Technologies, Inc. et al

3. Determine the error-location numbers $\beta_1, \beta_2, \ldots, \beta_v$ 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.

Iterative Algorithm for Finding the Error-Location Polynomial $\sigma(X)$

In the following, we present Berlekamp's iterative algorithm for finding the error-location polynomial. We only describe 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], Kasami et al. [14], and MacWilliams and Sloane [15].

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), a correction term is added 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, a correction term is added to $\sigma^{(2)}(X)$ to form $\sigma^{(3)}(X)$. Iteration continues until $\sigma^{(2t)}(X)$ is obtained. 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

$$\sigma^{(\mu)}(X) = 1 + \sigma_1^{(\mu)} X + \sigma_2^{(\mu)} X^2 + \cdots + \sigma_{l_\mu}^{(\mu)} X^{l_\mu} \tag{6.23}$$

be the minimum-degree polynomial determined at the μth 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:

$$d_\mu = S_{\mu+1} + \sigma_1^{(\mu)} S_\mu + \sigma_2^{(\mu)} S_{\mu-1} + \cdots + \sigma_{l_\mu}^{(\mu)} S_{\mu+1-l_\mu}. \tag{6.24}$$

This quantity d_μ is called the μth *discrepancy*. If $d_\mu = 0$, the coefficients of $\sigma^{(\mu)}(X)$ satisfy the $(\mu + 1)$th Newton's identity. In this event, we set

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X).$$

Ex.1020 / Page 174 of 305
Sandisk Technologies, Inc. et al

If $d_\mu \neq 0$, the coefficients of $\sigma^{(\mu)}(X)$ do not satisfy the $(\mu + 1)$th Newton's identity and a correction term must be added to $\sigma^{(\mu)}(X)$ to obtain $\sigma^{(\mu+1)}(X)$. To accomplish this correction, we go back to the steps *prior to* the μth step and determine a polynomial $\sigma^{(\rho)}(X)$ such that the ρth discrepancy $d_\rho \neq 0$ and $\rho - l_\rho$ [l_ρ is the degree of $\sigma^{(\rho)}(X)$] has the largest value. Then

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X) + d_\mu d_\rho^{-1} X^{(\mu-\rho)} \sigma^{(\rho)}(X), \tag{6.25}$$

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

TABLE 6.5

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	S_1	0	0
1				
2				
.				
.				
.				
$2t$				

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 μth 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 $d_\mu \neq 0$, find another row ρ prior to the μth 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

$$l_{\mu+1} = \max(l_\mu, l_\rho + \mu - \rho). \tag{6.26}$$

In either case,

$$d_{\mu+1} = S_{\mu+2} + \sigma_1^{(\mu+1)} S_{\mu+1} + \cdots + \sigma_{l_{\mu+1}}^{(\mu+1)} S_{\mu+2-l_{\mu+1}}, \tag{6.27}$$

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 it has degree greater than t, there are more 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 code vector of all zeros,

$$\mathbf{v} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$$

is transmitted and the vector

$$\mathbf{r} = (0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0)$$

Ex.1020 / Page 175 of 305
Sandisk Technologies, Inc. et al

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:

$$\mathbf{b}_1(X) = 1,$$
$$\mathbf{b}_3(X) = 1 + X^2 + X^3,$$
$$\mathbf{b}_5(X) = X^2.$$

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

$$S_3 = 1 + \alpha^6 + \alpha^9 = \alpha^{10},$$
$$S_6 = 1 + \alpha^{12} + \alpha^{18} = \alpha^5.$$

Substituting α^5 into $\mathbf{b}_5(X)$, we have

$$S_5 = \alpha^{10}.$$

Using the iterative procedure described above, we obtain Table 6.6. Thus, the error-location polynomial is

$$\sigma(X) = \sigma^{(6)}(X) = 1 + X + \alpha^5 X^3.$$

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 code vector.

TABLE 6.6

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	$1 + X + \alpha^5 X^3$	—	—	—	

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

Ex.1020 / Page 176 of 305
Sandisk Technologies, Inc. et al

iteration to find the error-location polynomial $\sigma(X)$. Let $\sigma^{(\mu)}(X)$ and d_μ be the solution and discrepancy obtained at the μth step of iteration. Let l_μ be the degree of $\sigma^{(\mu)}(X)$. Chen [17] 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. Based on this fact, if the number of errors in the received polynomial $\mathbf{r}(X)$ is ν ($\nu \leq t$), only $t + \nu$ steps of iteration is needed to determine the error-location polynomial $\sigma(X)$. If ν is small (this is often the case), the reduction in the number of iteration steps results in an increase of decoding speed.

The iterative algorithm for finding $\sigma(X)$ described above not only applies to binary BCH codes but also to nonbinary BCH codes.

Simplified Algorithm for Finding $\sigma(X)$

For binary BCH codes, it is not necessary to fill out the empty $2t$ rows of Table 6.5 for finding $\sigma(X)$. A simplified algorithm [9, 13] can be obtained that requires filling out a table with only t empty rows. Such a table is presented as Table 6.7. Assuming

TABLE 6.7

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				
⋮				
t				

that we have filled out all rows up to and including the μth 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 $d_\mu \neq 0$, find another row preceding the μth row, say the ρth, such that the number $2\rho - l_\rho$ in the last column is as large as possible and $d_\rho \neq 0$. Then

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X) + d_\mu d_\rho^{-1} X^{2(\mu-\rho)} \sigma^{(\rho)}(X). \tag{6.28}$$

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

$$d_{\mu+1} = S_{2\mu+3} + \sigma_1^{(\mu+1)} S_{2\mu+2} + \sigma_2^{(\mu+1)} S_{2\mu+1} + \cdots + \sigma_{l_{\mu+1}}^{(\mu+1)} S_{2\mu+3-l_{\mu+1}}. \tag{6.29}$$

The polynomial $\sigma^{(t)}(X)$ in the last row should be the required $\sigma(X)$. If it has degree 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 one-half of the computation required in the general algorithm. However, we must remember that the

Ex.1020 / Page 177 of 305
Sandisk Technologies, Inc. et al

simplified algorithm applies only to binary BCH codes. Again, if the number of errors in the 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, 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 [17]. 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.

TABLE 6.8

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 $\rho = 1$)

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 $1, \alpha, \alpha^2, \ldots, \alpha^{n-1}$ ($n = 2^m - 1$) 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. The error-location polynomial has been found to be

$$\sigma(X) = 1 + X + \alpha^5 X^3.$$

By substituting $1, \alpha, \alpha^2, \ldots, \alpha^{14}$ 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 substitution method described above 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 to carry out the substitution and error correction. Chien's procedure for searching error-location numbers is described next. The received vector

$$\mathbf{r}(X) = r_0 + r_1 X + r_2 X^2 + \cdots + r_{n-1} X^{n-1}$$

is decoded on a bit-by-bit basis. 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

Ex.1020 / Page 178 of 305
Sandisk Technologies, Inc. et al

to testing whether its inverse, α, is a root of $\sigma(X)$. If α is a root, then

$$1 + \sigma_1\alpha + \sigma_2\alpha^2 + \cdots + \sigma_\nu\alpha^\nu = 0.$$

Therefore, to decode r_{n-1}, the decoder forms $\sigma_1\alpha, \sigma_2\alpha^2, \ldots, \sigma_\nu\alpha^\nu$. If the sum $1 + \sigma_1\alpha + \sigma_2\alpha^2 + \cdots + \sigma_\nu\alpha^\nu = 0$, 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 $\sigma_1\alpha^l, \sigma_2\alpha^{2l}, \ldots, \sigma_\nu\alpha^{\nu l}$ and tests the sum

$$1 + \sigma_1\alpha^l + \sigma_2\alpha^{2l} + \cdots + \sigma_\nu\alpha^{\nu l}.$$

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 testing procedure for error locations described above can be implemented in a straightforward manner by a circuit such as that shown in Figure 6.1 [6]. The t

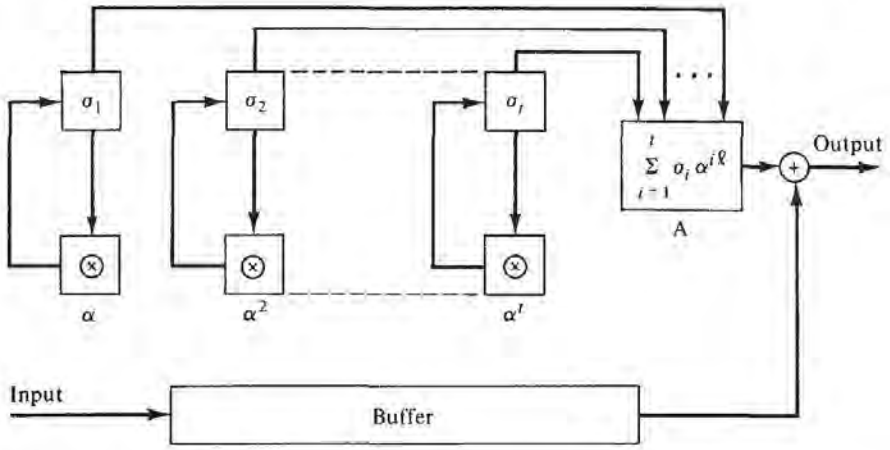


Figure 6.1 Cyclic error location search unit.

σ-registers are initially stored with $\sigma_1, \sigma_2, \ldots, \sigma_t$ calculated in step 2 of the decoding ($\sigma_{\nu+1} = \sigma_{\nu+2} = \cdots = \sigma_t = 0$ for $\nu < t$). Immediately before r_{n-1} is read out of the buffer, the t multipliers $\otimes$ are pulsed once. The multiplications are performed and $\sigma_1\alpha, \sigma_2\alpha^2, \ldots, \sigma_\nu\alpha^\nu$ are stored in the σ-registers. The output of the logic circuit A is 1 if and only if the sum $1 + \sigma_1\alpha + \sigma_2\alpha^2 + \cdots + \sigma_\nu\alpha^\nu = 0$; 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. Having decoded r_{n-1}, the t multipliers are pulsed again. Now $\sigma_1\alpha^2, \sigma_2\alpha^4$, and $\sigma_\nu\alpha^{2\nu}$ are stored in the σ-registers. The sum

$$1 + \sigma_1\alpha^2 + \sigma_2\alpha^4 + \cdots + \sigma_\nu\alpha^{2\nu}$$

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.

The decoding algorithm described above also applies to nonprimitive BCH code. The $2t$ syndrome components are given by

$$S_i = \mathbf{r}(\beta^i)$$

for $1 \leq i \leq 2t$.

Ex.1020 / Page 179 of 305
Sandisk Technologies, Inc. et al

6.3 IMPLEMENTATION OF GALOIS FIELD ARITHMETIC

From the discussion above, we see that the 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. Addition of two field elements can be accomplished with the circuit shown in Figure 6.2. First, the vector representations of the two elements to be added are loaded into the two 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).

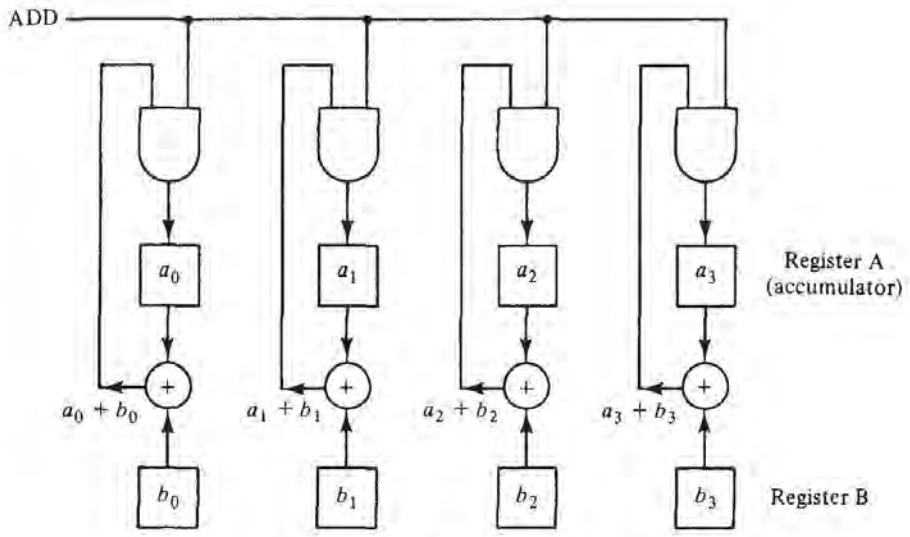


Figure 6.2 Galois field adder.

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 $\phi(X) = 1 + X + X^4$. The element β can be expressed 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 the equality above by α and using the fact $\alpha^4 = 1 + \alpha$, we obtain the following equality:

$$\alpha\beta = b_3 + (b_0 + b_3)\alpha + b_1\alpha^2 + b_2\alpha^3.$$

Ex.1020 / Page 180 of 305
Sandisk Technologies, Inc. et al

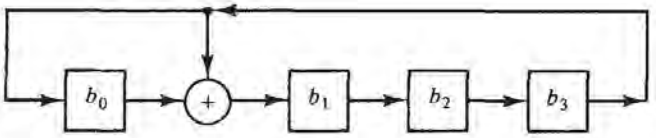


Figure 6.3 Circuit for multiplying arbitrary element in GF(2^4) by α.

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 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). 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 above 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 expression above, we obtain a circuit as 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 $\alpha^3\beta$.

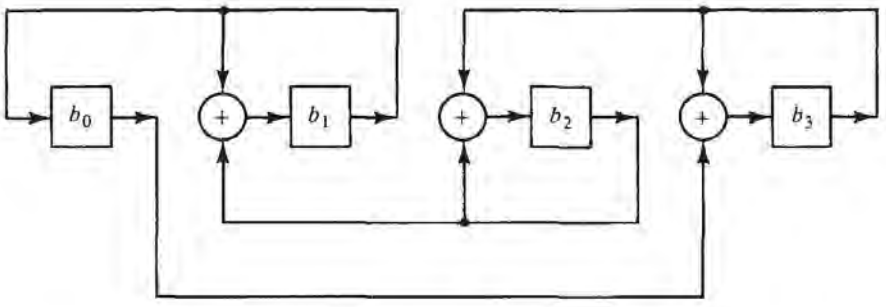


Figure 6.4 Circuit for multiplying arbitrary element in GF(2^4) by α^3.

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). Express these two elements in polynomial form:

Ex.1020 / Page 181 of 305
Sandisk Technologies, Inc. et al

$$\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3,$$
$$\gamma = c_0 + c_1\alpha + c_2\alpha^2 + c_3\alpha^3.$$

Then the product $\beta\gamma$ can be expressed in the following form:

$$\beta\gamma = (((c_3\beta)\alpha + c_2\beta)\alpha + c_1\beta)\alpha + c_0\beta \tag{6.30}$$

This product can be carried out 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 (6.30). 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, the contents of register A form 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

$$\beta\gamma = (((c_0\beta) + c_1\beta\alpha) + c_2\beta\alpha^2) + c_3\beta\alpha^3,$$

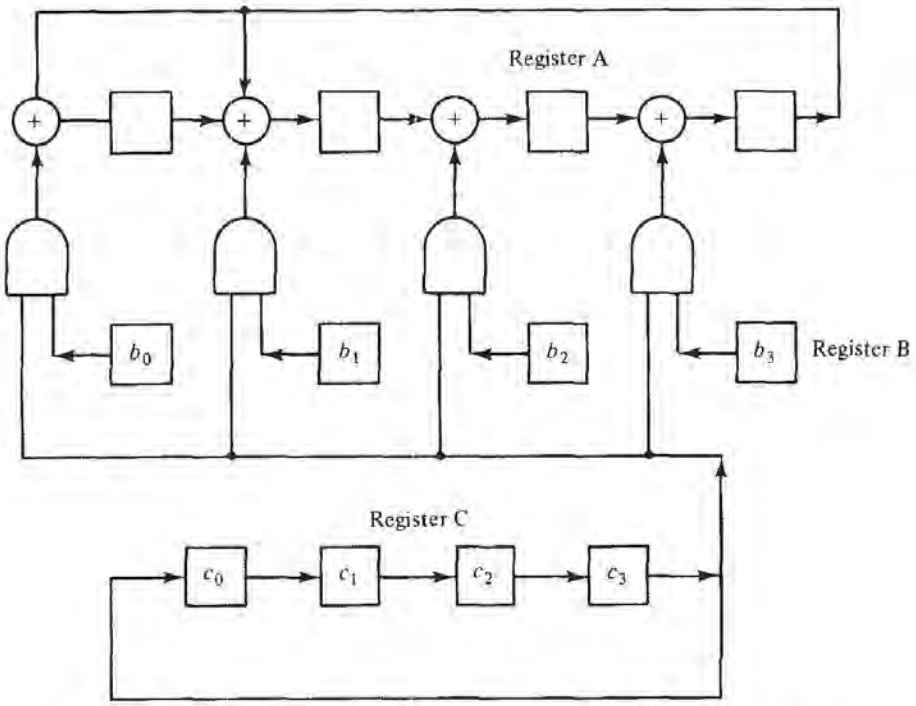


Figure 6.5 Circuit for multiplying two elements of GF(2^4).

Ex.1020 / Page 182 of 305
Sandisk Technologies, Inc. et al

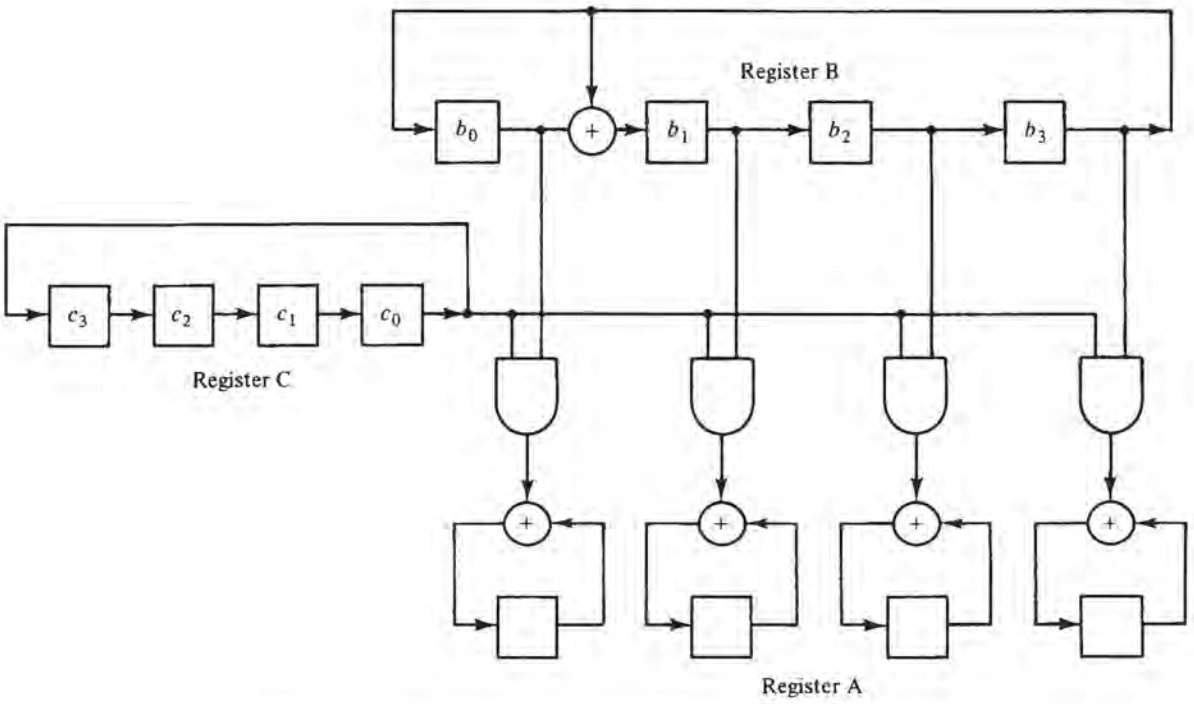


Figure 6.6 Another circuit for multiplying two elements of GF(2^4).

we obtain a different multiplication circuit as shown in Figure 6.6. To multiply, 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 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.

Multiplication of two field elements from GF(2^m) can be implemented in a combinational logic circuit with $2m$ inputs and m outputs. The advantage of this implementation is its speed. However, for $m > 7$, it becomes prohibitively complex and costly. Multiplication can also be programmed in a general-purpose computer; it would require roughly $5m$ instruction executions.

Let $\mathbf{r}(X)$ be a polynomial over GF(2). Next we consider how to compute $\mathbf{r}(\alpha^l)$. 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^l in GF(2^m). Again, we use computation over GF(2^4) for illustration. Suppose that we want to compute

$$\mathbf{r}(\alpha) = r_0 + r_1\alpha + r_2\alpha^2 + \cdots + r_{14}\alpha^{14}, \tag{6.31}$$

where α is a primitive element in GF(2^4) given by Table 2.8. The right-hand side of (6.31) can be expressed in the form

$$\mathbf{r}(\alpha) = (\cdots(((r_{14})\alpha + r_{13})\alpha + r_{12})\alpha + \cdots)\alpha + r_0.$$

Then computation of $\mathbf{r}(\alpha)$ can be accomplished 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 $(r_0, r_1, \ldots, r_{14})$ is shifted into the circuit one digit at a time. After the first

Ex.1020 / Page 183 of 305
Sandisk Technologies, Inc. et al

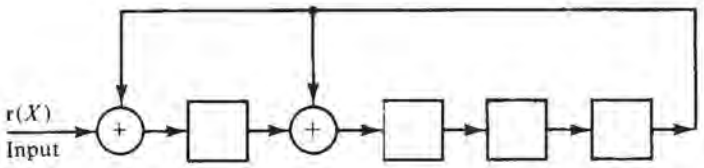


Figure 6.7 Circuit for computing $\mathbf{r}(\alpha)$.

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 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.

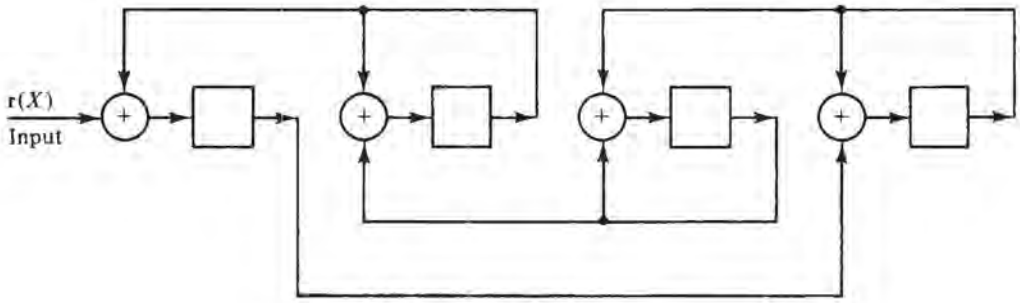


Figure 6.8 Circuit for computing $\mathbf{r}(\alpha^3)$.

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

$$\begin{aligned}\mathbf{r}(\alpha^3) &= \mathbf{b}(\alpha^3)\\ &= b_0 + b_1\alpha^3 + b_2\alpha^6 + b_3\alpha^9\\ &= b_0 + b_1\alpha^3 + b_2(\alpha^2 + \alpha^3) + b_3(\alpha + \alpha^3)\\ &= b_0 + b_3\alpha + b_2\alpha^2 + (b_1 + b_2 + b_3)\alpha^3.\end{aligned} \tag{6.32}$$

From the expression above 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 (6.32). Such a circuit is shown in Figure 6.9, where the feedback connection of the shift register is based on $\phi_3(X) = 1 + X + X^2 + X^3 + X^4$. Since 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 $\mathbf{r}(\alpha^6)$, the coefficients of $\mathbf{b}(X)$ are combined in the

Ex.1020 / Page 184 of 305
Sandisk Technologies, Inc. et al

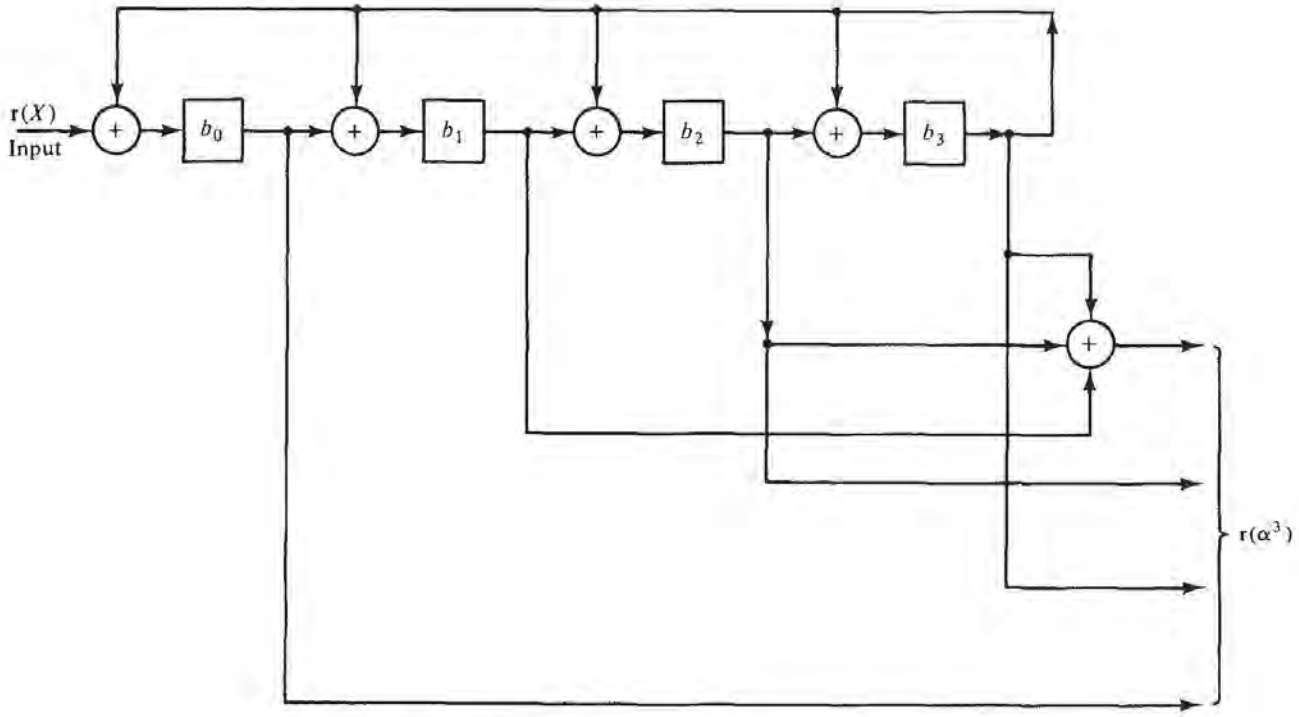


Figure 6.9 Another circuit for computing $r(\alpha^3)$ in GF(2^4).

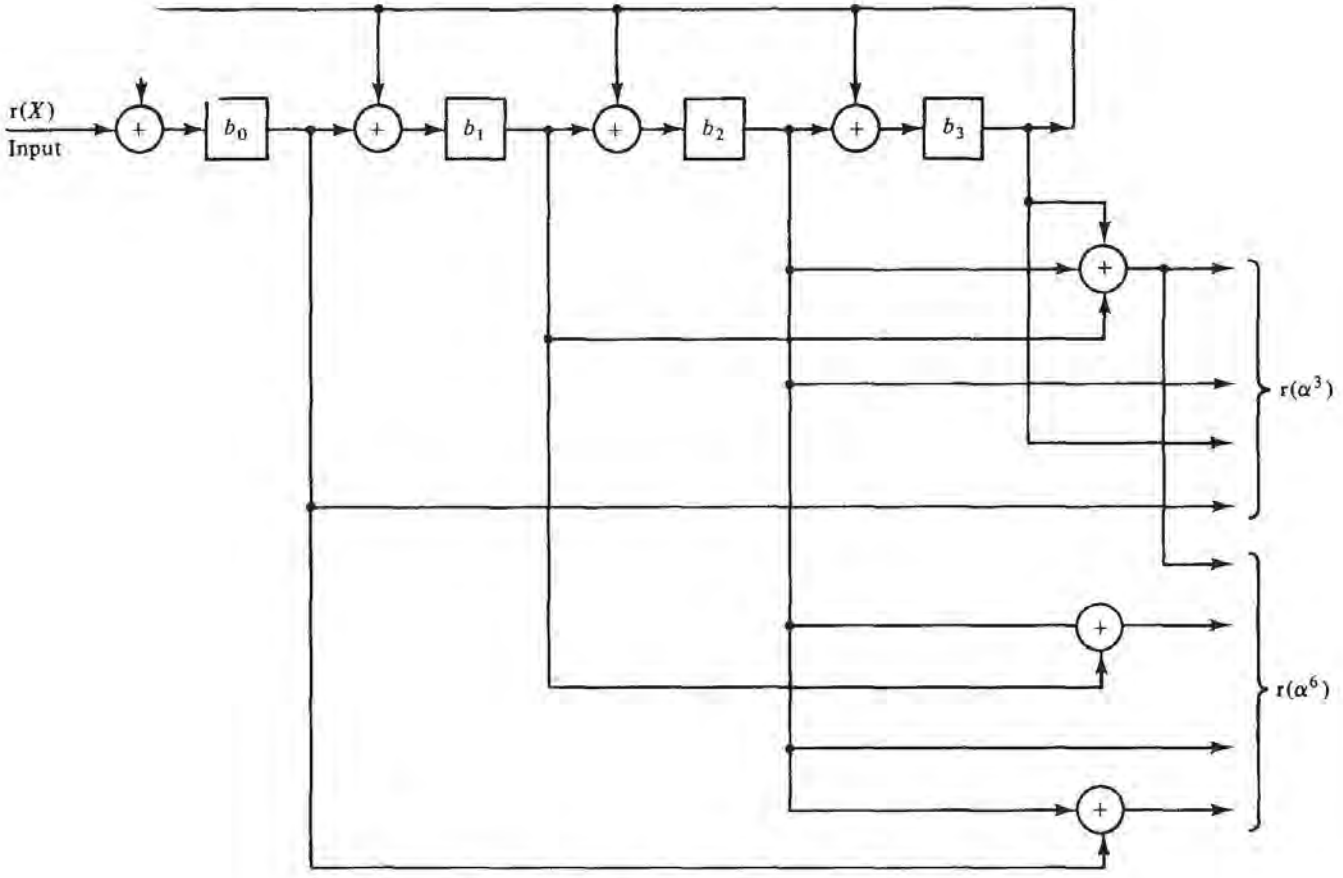


Figure 6.10 Circuit for computing $\mathbf{r}(\alpha^3)$ and $\mathbf{r}(\alpha^6)$ in GF(2^4).

Ex.1020 / Page 185 of 305
Sandisk Technologies, Inc. et al

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

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

6.4 IMPLEMENTATION OF ERROR CORRECTION

Each step in the decoding of a BCH code can be implemented either by digital hardware or by software (i.e., programmed on a general-purpose computer). Each implementation has certain advantages. We consider these implementations next.

Syndrome Computations

The first step in decoding a t-error-correction BCH code is to compute the $2t$ syndrome components $S_1, S_2, \ldots, S_{2t}$. These syndrome components may be obtained by substituting the field elements $\alpha, \alpha^2, \ldots, \alpha^{2t}$ into the received polynomial $\mathbf{r}(X)$. For software implementation, substituting α^i into $\mathbf{r}(X)$ is best accomplished in the following manner:

$$\begin{aligned}S_i = \mathbf{r}(\alpha^i) &= r_{n-1}(\alpha^i)^{n-1} + r_{n-2}(\alpha^i)^{n-2} + \cdots + r_1\alpha^i + r_0\\ &= (\cdots((r_{n-1}\alpha^i + r_{n-2})\alpha^i + r_{n-3})\alpha^i + \cdots + r_1)\alpha^i + r_0.\end{aligned}$$

This computation takes $n - 1$ additions and $n - 1$ multiplications. For binary BCH codes, it can be shown that $S_{2i} = S_i^2$ (see Problem 6.5). 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 6.3. 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.

Ex.1020 / Page 186 of 305
Sandisk Technologies, Inc. et al

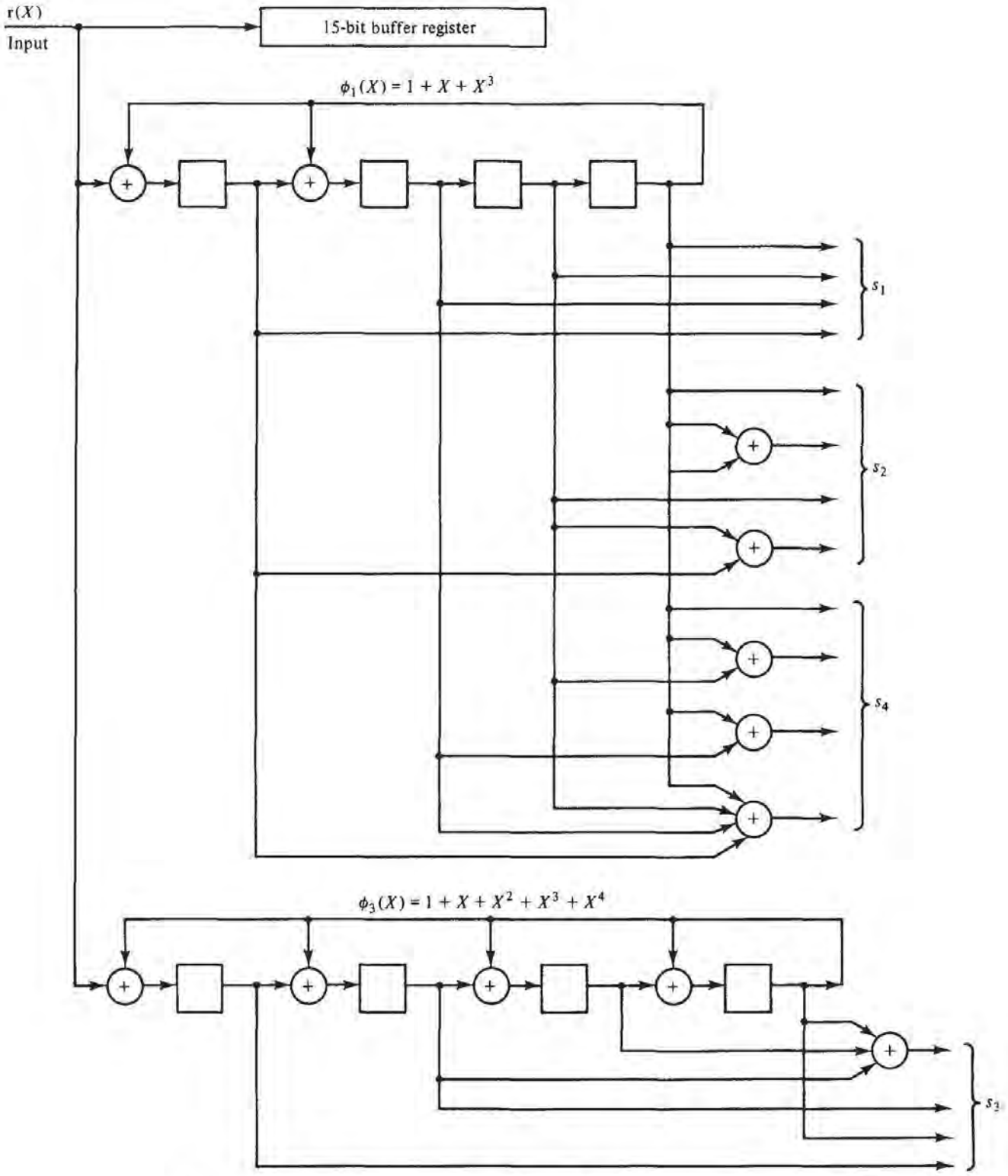


Figure 6.11 Syndrome computation circuit for the double-error-correcting (15, 7) BCH code.

Ex.1020 / Page 187 of 305
Sandisk Technologies, Inc. et al

The advantage of hardware implementation of syndrome computation is speed; however, software implementation is less expensive.

Finding the Error-Location Polynomial $\sigma(X)$

For this step the software computation requires somewhat less 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 would depend 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 multiplication. 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.

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 can also be done in hardware using Chien's searching circuit shown in Figure 6.1. Chien's searching circuit requires t multipliers for multiplying by $\alpha, \alpha^2, \ldots, \alpha^t$, respectively. These multipliers may be the type of circuits shown in Figures 6.3 and 6.4. Initially, $\sigma_1, \sigma_2, \ldots, \sigma_t$ found in step 2 are loaded into the registers of the t multipliers. Then these multipliers are shifted n times. At the end the lth shift, the t registers contain $\sigma_1\alpha^l, \sigma_2\alpha^{2l}, \ldots, \sigma_t\alpha^{tl}$. Then the sum

$$1 + \sigma_1\alpha^l + \sigma_2\alpha^{2l} + \cdots + \sigma_t\alpha^{tl}$$

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. A m-input OR gate is used to test whether the sum is zero. It takes n clock cycles to complete this step. If we only want to correct the message digits, only k clock cycles are needed. A Chien's searching circuit for the double-error-correcting (15, 7) BCH code is shown in Figure 6.12.

For large t and m, the cost for building t wired multipliers for multiplying α, $\alpha^2, \ldots, \alpha^t$ in one clock cycle becomes substantial. For more economical but slower multipliers, we may use the type of circuit shown in Figure 6.5 (or shown in Fig. 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 multipliers, 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. Since 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, the hardware implementation of these steps is essential if high decoding speed is needed. With hardware implementation, step 1

Ex.1020 / Page 188 of 305
Sandisk Technologies, Inc. et al

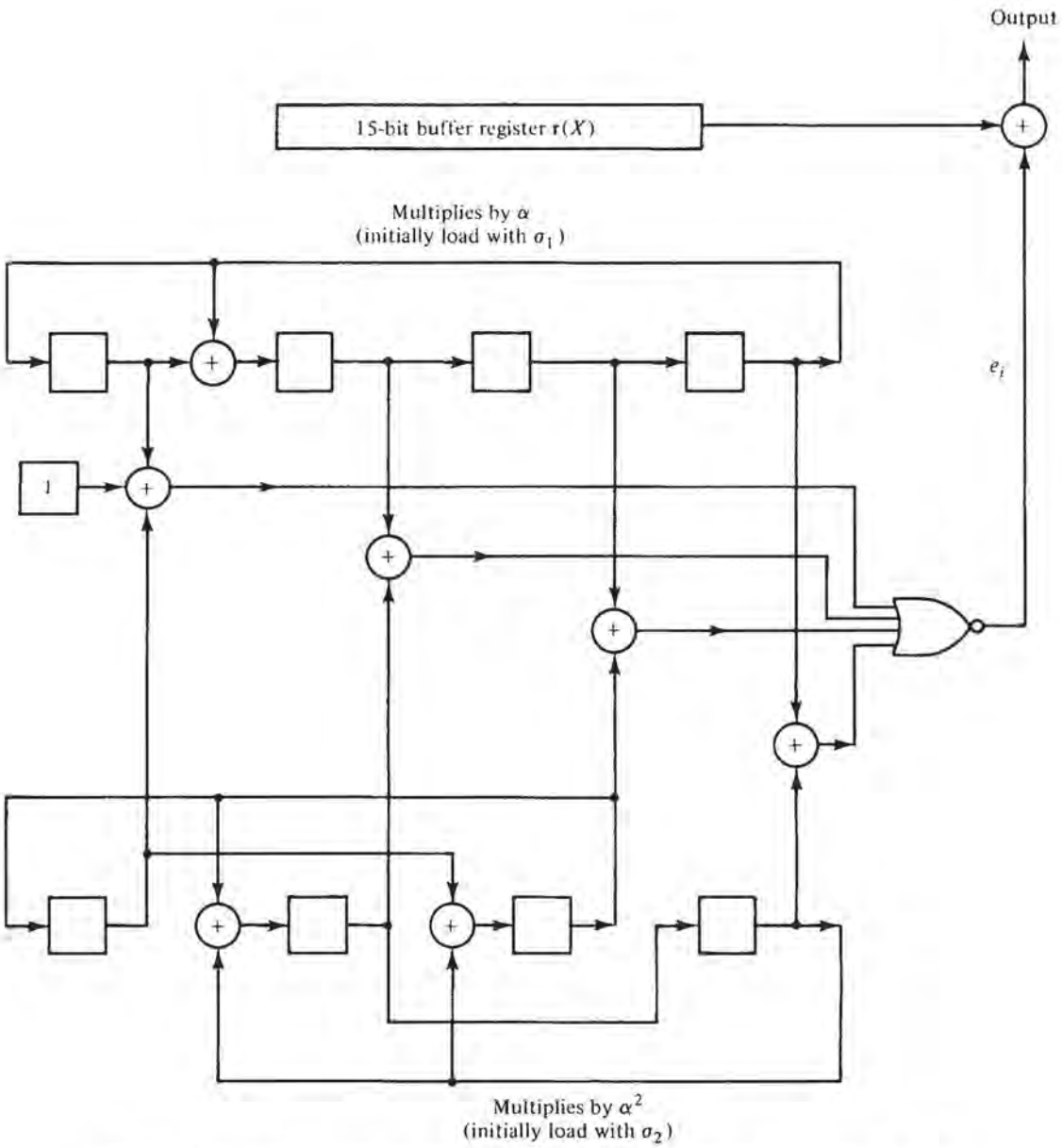


Figure 6.12 Chien's searching circuit for the double-error-correcting (15, 7) BCH code.

can be done as the received polynomial $\mathbf{r}(X)$ is read in and step 3 can be accomplished as $\mathbf{r}(X)$ is read out, and in this case the computation time required in steps 1 and 3 is essentially negligible.

6.5 NONBINARY BCH CODES AND REED–SOLOMON CODES

In addition to the binary codes, there are nonbinary codes. In fact, if p is a prime number and q is any power of p, there are codes with symbols from the Galois field GF(q). These codes are called *q-ary codes*. The concepts and properties developed

Ex.1020 / Page 189 of 305
Sandisk Technologies, Inc. et al

for the binary codes in the previous chapters apply to q-ary codes with little modification. An (n, k) linear code with symbols from GF(q) is a k-dimensional subspace of the vector space of all n-tuples over GF(q). A q-ary (n, k) cyclic code is generated by a polynomial of degree $n - k$ with coefficients from GF(q), which is a factor of $X^n - 1$. Encoding and decoding of q-ary codes are similar to that of binary codes.

The binary BCH codes defined in Section 6.1 can be generalized to nonbinary codes in a straightforward manner. For any choice of positive integers s and t, there exists a q-ary BCH code of length $n = q^s - 1$, which is capable of correcting any combination of t or fewer errors and requires no more than $2st$ parity-check digits. Let α be a primitive element in the Galois field GF(q^s). The generator polynomial $\mathbf{g}(X)$ of a t-error-correcting q-ary BCH is the polynomial of lowest degree with coefficients from GF(q) for which $\alpha, \alpha^2, \ldots, \alpha^{2t}$ are roots. Let $\phi_i(X)$ be the minimal polynomial of α^i. Then

$$\mathbf{g}(X) = \text{LCM}\,\{\phi_1(X), \phi_2(X), \ldots, \phi_{2t}(X)\}.$$

The degree of each minimal polynomial is s or less. Therefore, the degree of $\mathbf{g}(X)$ is at most $2st$, and hence the number of parity-check digits of the code generated by $\mathbf{g}(X)$ is no more than $2st$. For $q = 2$, we obtain the binary BCH codes. In this section we study only a special subclass of q-ary BCH codes. For details of q-ary BCH codes, the reader is referred to References 4, 7, 9, and 13 to 15.

The special subclass of q-ary BCH codes for which $s = 1$ is the most important subclass of q-ary BCH codes. The codes of this subclass are usually called the *Reed–Solomon* codes in honor of their discoverers [5]. A t-error-correcting Reed–Solomon code with symbols from GF(q) has the following parameters:

Block length:	$n = q - 1$
Number of parity-check digits:	$n - k = 2t$,
Minimum distance:	$d_{\min} = 2t + 1$.

We see that the length of the code is one less than the size of code symbols and the minimum distance is one greater than the number of parity-check digits. In what follows we consider Reed–Solomon codes with code symbols from the Galois field GF(2^m) (i.e., $q = 2^m$). Let α be a primitive element in GF(2^m). The generator polynomial of a primitive t-error-correcting Reed–Solomon code of length $2^m - 1$ is

$$\begin{aligned}\mathbf{g}(X) &= (X + \alpha)(X + \alpha^2)\cdots(X + \alpha^{2t}) \\ &= g_0 + g_1X + g_2X^2 + \cdots + g_{2t-1}X^{2t-1} + X^{2t}.\end{aligned}$$

Clearly, $\mathbf{g}(X)$ has $\alpha, \alpha^2, \ldots, \alpha^{2t}$ as all its roots and has coefficients from GF(2^m). The code generated by $\mathbf{g}(X)$ is an $(n, n - 2t)$ cyclic code which consists of those polynomials of degree $n - 1$ or less with coefficients from GF(2^m) that are multiples of $\mathbf{g}(X)$. Encoding of this code is similar to 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 digits 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 $\mathbf{g}(X)$. In hardware implementation, this is accomplished by using a

Ex.1020 / Page 190 of 305
Sandisk Technologies, Inc. et al

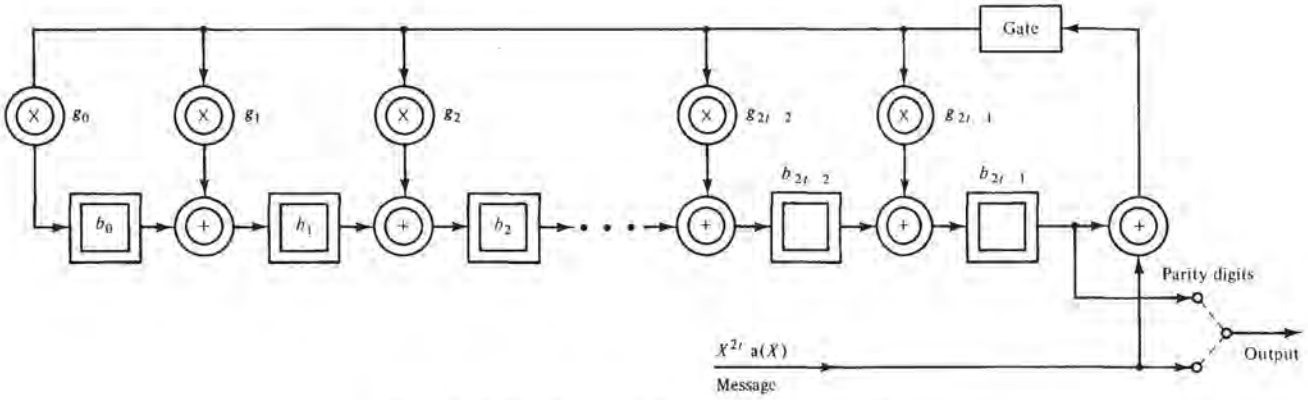
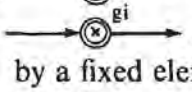


Figure 6.13 Encoding circuit for a nonbinary cyclic code.

division circuit as shown in Figure 6.13, where

1.  denotes an adder that adds two elements from GF(2^m).
2.  denotes a multiplier that multiplies a field element from GF(2^m) by a fixed element g_i from the same field.
3.  denotes a storage device that is capable of storing a field element b_i from GF(2^m).

As soon as the message $\mathbf{a}(X)$ has entered the channel and the circuit, the parity-check digits are in the register.

Using the same argument as for the binary BCH codes, we can show that the t-error-correcting Reed–Solomon code has minimum distance at least $2t + 1$. In fact we can prove that the minimum distance is exactly $2t + 1$ (see Problem 6.7).

Let

$$\mathbf{v}(X) = v_0 + v_1 X + \cdots + v_{n-1} X^{n-1}$$

be the transmitted code vector and let

$$\mathbf{r}(X) = r_0 + r_1 X + \cdots + r_{n-1} X^{n-1}$$

be the corresponding received vector. 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 from GF(2^m). Suppose that the error pattern $\mathbf{e}(X)$ contains ν errors (nonzero components) at location $X^{j_1}, X^{j_2}, \ldots, X^{j_\nu}$ where $0 \leq j_1 < j_2 < \cdots < j_\nu \leq n - 1$. Then

$$\mathbf{e}(X) = e_{j_1} X^{j_1} + e_{j_2} X^{j_2} + \cdots + e_{j_\nu} X^{j_\nu}.$$

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 ν pairs (X^{j_i}, e_{j_i})'s].

As with binary BCH codes, we define

$$\beta_l = \alpha^{j_l} \qquad \text{for } l = 1, 2, \ldots, \nu$$

as error-location numbers. In decoding a Reed–Solomon code (or any q-ary BCH code), the same three steps used for decoding a binary BCH code are required; in addition, a fourth step involving calculation of the error values is required. The $2t$ syndrome components are obtained by substituting α^i into the received polynomial $\mathbf{r}(X)$ for $i = 1, 2, \ldots, 2t$. Thus, we have

$$\begin{aligned} S_1 &= \mathbf{r}(\alpha) = e_{j_1}\beta_1 + e_{j_2}\beta_2 + \cdots + e_{j_\nu}\beta_\nu \\ S_2 &= \mathbf{r}(\alpha^2) = e_{j_1}\beta_1^2 + e_{j_2}\beta_2^2 + \cdots + e_{j_\nu}\beta_\nu^2 \\ &\vdots \\ S_{2t} &= \mathbf{r}(\alpha^{2t}) = e_{j_1}\beta_1^{2t} + e_{j_2}\beta_2^{2t} + \cdots + e_{j_\nu}\beta_\nu^{2t}. \end{aligned}$$

The syndrome component S_i can also be computed by dividing $\mathbf{r}(X)$ by $X + \alpha^i$. The division results in the equality

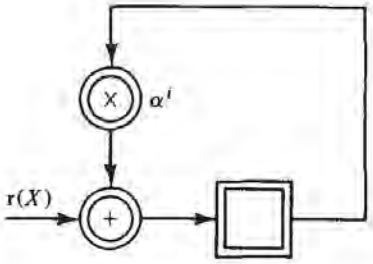
$$\mathbf{r}(X) = \mathbf{c}_i(X)(X + \alpha^i) + b_i, \tag{6.33}$$

Ex.1020 / Page 192 of 305
Sandisk Technologies, Inc. et al

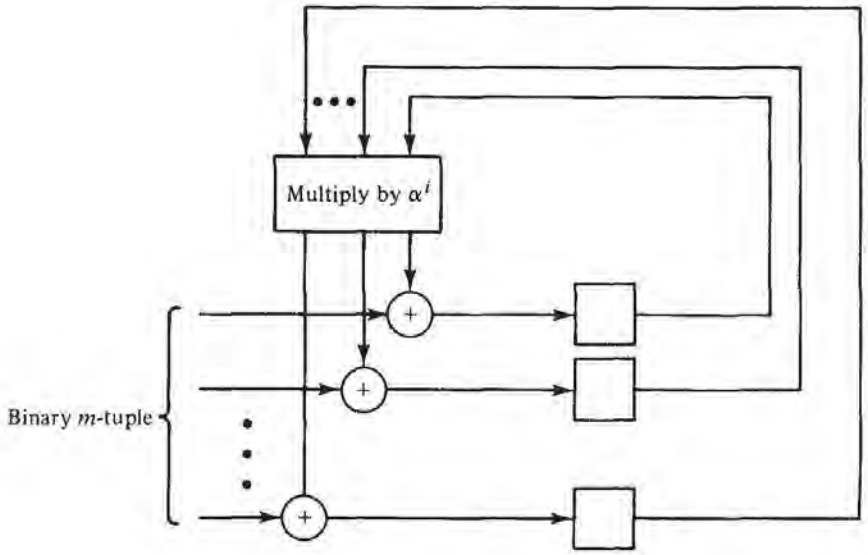
where the remainder b_i is a constant in GF(2^m). Substituting α^i in both sides of (6.33), we have

$$S_i = b_i.$$

This computation can be accomplished with a division circuit shown in Figure 6.14.



(a)



(b)

Figure 6.14 Syndrome computation circuits for Reed–Solomon codes: (a) over GF(2^m); (b) in binary form.

To find the error-location polynomial

$$\begin{aligned}\sigma(X) &= (1 + \beta_1 X)(1 + \beta_2 X)\cdots(1 + \beta_\nu X) \\ &= 1 + \sigma_1 X + \cdots + \sigma_\nu X^\nu\end{aligned}$$

with Berlekamp's iterative algorithm, we fill out Table 6.5. Once $\sigma(X)$ is found, we can determine the error values. Let

$$\begin{aligned}Z(X) = 1 + (S_1 + \sigma_1)X + (S_2 + \sigma_1 S_1 + \sigma_2)X^2 + \cdots \\ + (S_\nu + \sigma_1 S_{\nu-1} + \sigma_2 S_{\nu-2} + \cdots + \sigma_\nu)X^\nu\end{aligned} \tag{6.34}$$

Ex.1020 / Page 193 of 305
Sandisk Technologies, Inc. et al

Then the error value at location $\beta_l = \alpha^{j_l}$ is given by [9]

$$e_{j_l} = \frac{Z(\beta_l^{-1})}{\prod_{\substack{i=1 \\ i \neq l}}^{v} (1 + \beta_i \beta_l^{-1})}. \tag{6.35}$$

The decoding computation for a Reed–Solomon code is best explained by an example. Consider a triple-error-correcting Reed–Solomon code with symbols from GF(2^4). The generator polynomial of this code is

$$\begin{aligned} \mathbf{g}(X) &= (X + \alpha)(X + \alpha^2)(X + \alpha^3)(X + \alpha^4)(X + \alpha^5)(X + \alpha^6) \\ &= \alpha^6 + \alpha^9 X + \alpha^6 X^2 + \alpha^4 X^3 + \alpha^{14} X^4 + \alpha^{10} X^5 + X^6. \end{aligned}$$

Let the all-zero vector be the transmitted code vector and let $\mathbf{r} = (0\ 0\ 0\ \alpha^7\ 0\ 0\ \alpha^3\ 0\ 0\ 0\ 0\ 0\ \alpha^4\ 0\ 0)$ be the received vector. Thus, $\mathbf{r}(X) = \alpha^7 X^3 + \alpha^3 X^6 + \alpha^4 X^{12}$.

Step 1. The syndrome components are computed as follows (use Table 2.8):

$$\begin{aligned} S_1 &= \mathbf{r}(\alpha) = \alpha^{10} + \alpha^9 + \alpha = \alpha^{12} \\ S_2 &= \mathbf{r}(\alpha^2) = \alpha^{13} + 1 + \alpha^{13} = 1 \\ S_3 &= \mathbf{r}(\alpha^3) = \alpha + \alpha^6 + \alpha^{10} = \alpha^{14} \\ S_4 &= \mathbf{r}(\alpha^4) = \alpha^4 + \alpha^{12} + \alpha^7 = \alpha^{10} \\ S_5 &= \mathbf{r}(\alpha^5) = \alpha^7 + \alpha^3 + \alpha^4 = 0 \\ S_6 &= \mathbf{r}(\alpha^6) = \alpha^{10} + \alpha^9 + \alpha = \alpha^{12}. \end{aligned}$$

Step 2. To find the error-location polynomial $\sigma(X)$, we fill out Table 6.5. Thus, $\sigma(X) = 1 + \alpha^7 X + \alpha^4 X^2 + \alpha^6 X^3$.

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	$1 + \alpha^3 X$	1	1	1	(take $\rho = 0$)
3	$1 + \alpha^3 X + \alpha^3 X^2$	α^7	2	1	(take $\rho = 0$)
4	$1 + \alpha^4 X + \alpha^{12} X^2$	α^{10}	2	2	(take $\rho = 2$)
5	$1 + \alpha^7 X + \alpha^4 X^2 + \alpha^5 X^3$	0	3	2	(take $\rho = 3$)
6	$1 + \alpha^7 X + \alpha^4 X^2 + \alpha^6 X^3$	—	—	—	

Step 3. By substituting $1, \alpha, \alpha^2, \ldots, \alpha^{14}$ 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}.

Step 4. From (6.34) we find that

$$Z(X) = 1 + \alpha^2 X + X^2 + \alpha^6 X^3.$$

Ex.1020 / Page 194 of 305
Sandisk Technologies, Inc. et al

Using (6.35), we obtain the error values at locations X^3, X^6, and X^{12}:

$$e_3 = \frac{1 + \alpha^2\alpha^{-3} + \alpha^{-6} + \alpha^6\alpha^{-9}}{(1 + \alpha^6\alpha^{-3})(1 + \alpha^{12}\alpha^{-3})} = \frac{1 + \alpha^{14} + \alpha^9 + \alpha^{12}}{\alpha^{14}\alpha^7} = \frac{\alpha^{13}}{\alpha^6} = \alpha^7,$$

$$e_6 = \frac{1 + \alpha^2\alpha^{-6} + \alpha^{-12} + \alpha^6\alpha^{-18}}{(1 + \alpha^3\alpha^{-6})(1 + \alpha^{12}\alpha^{-6})} = \frac{1 + \alpha^{11} + \alpha^3 + \alpha^3}{\alpha^9} = \frac{\alpha^{12}}{\alpha^9} = \alpha^3,$$

$$e_{12} = \frac{1 + \alpha^2\alpha^{-12} + \alpha^{-24} + \alpha^6\alpha^{-36}}{(1 + \alpha^3\alpha^{-12})(1 + \alpha^6\alpha^{-12})} = \frac{1 + \alpha^5 + \alpha^6 + 1}{\alpha^5} = \frac{\alpha^9}{\alpha^5} = \alpha^4.$$

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 $\mathbf{r}(X) - \mathbf{e}(X)$.

If β is not a primitive element of GF(2^m), then the 2^m-ary code generated by

$$\mathbf{g}(X) = (X + \beta)(X + \beta^2)\cdots(X + \beta^{2t})$$

is a nonprimitive t-error-correcting Reed–Solomon code. The length n of this code is simply the order of β. Decoding of a nonprimitive Reed–Solomon code is identical to the decoding of a primitive Reed–Solomon code.

Reed–Solomon codes are very effective for correcting multiple bursts of errors. This is discussed in Chapter 9.

Two information symbols can be added to a Reed–Solomon code of length n *without reducing its minimum distance*. The extended Reed–Solomon code has length $n + 2$ and the same number of parity-check symbols as the original code. For a t-error-correcting Reed–Solomon, the parity-check matrix may take the form

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & \cdots & (\alpha^2)^{n-1} \\ \cdot & & & & \\ \cdot & & & & \\ \cdot & & & & \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & \cdots & (\alpha^{2t})^{n-1} \end{bmatrix}.$$

Then the parity-check matrix of the extended Reed–Solomon code is

$$\mathbf{H}_1 = \begin{bmatrix} 0 & 1 & \\ 0 & 0 & \\ \cdot & \cdot & \\ \cdot & \cdot & \mathbf{H} \\ \cdot & \cdot & \\ 0 & 0 & \\ 1 & 0 & \end{bmatrix}.$$

The result above was first obtained by Kasami, Lin, and Peterson [18,19] and later independently by Wolf [20].

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; however, these good codes have not been explicitly identified. Goppa codes are in general noncyclic (except the BCH codes), and they can be decoded in a

Ex.1020 / Page 195 of 305
Sandisk Technologies, Inc. et al

manner very similar to the decoding of 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 [20]. 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 detail of Goppa codes, the reader is referred to References 14, 15, and 21 to 25.

Our presentation of BCH codes and their implementation is given in time domain. BCH codes can also be defined and implemented in frequency domain using Fourier transforms over Galois fields. Decoding BCH codes in frequency domain sometimes offer computational or implementation advantages. Reference 26 is an excellent source to find transform techniques for decoding BCH codes.

6.6 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, for the other BCH codes, their weight distributions are still unknown. Computation of the weight distribution of a double-error-correcting or a triple-error-correcting primitive BCH code can be achieved 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 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 [27]. For more on the weight distribution of primitive binary BCH codes, the reader is referred to References 9 and 27.

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

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}	$(2^m - 2^{m-1} + 1)(2^m - 1)$
$2^{m-1} + 2^{(m+1)/2-1}$	$[2^{m-2} - 2^{(m-1)/2-1}](2^m - 1)$

Ex.1020 / Page 196 of 305
Sandisk Technologies, Inc. et al

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}	$(29 \cdot 2^{2m} - 4 \cdot 2^m + 64)(2^m - 1)/64$
$2^{m-1} + 2^{m/2-1}$	$2(2^{m-1} - 2^{m/2-1})(3 \cdot 2^m + 8)(2^m - 1)/15$
$2^{m-1} + 2^{(m+2)/2-1}$	$7[2^{m-1} - 2^{(m+2)/2-1}]2^m(2^m - 1)/48$
$2^{m-1} + 2^{(m+4)/2-1}$	$[2^{m-1} - 2^{(m+4)/2-1}](2^m - 4)(2^m - 1)/960$

Ex.1020 / Page 197 of 305
Sandisk Technologies, Inc. et al

tables, Tables 6.9 to 6.12. It has been proved [28] 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 of the code. It is unknown whether the probability of an undetected error for a triple-error-correcting primitive BCH code of length $2^m - 1$ satisfies the upper bound 2^{-3m}, where $3m$ is the number of parity-check digits of the code.

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 [29] 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 certain constant $m_0(t)$, the number of code vectors of weight i satisfies the following equalities:

$$A_i = \begin{cases} 0 & \text{for } 0 < i \leq 2t \\ (1 + \lambda_0 \cdot n^{-1/10})\binom{n}{i} 2^{-(n-k)} & \text{for } i > 2t, \end{cases} \tag{6.36}$$

where $n = 2^m - 1$ and λ_0 is upper bounded by a constant. From (3.19) and (6.36), we obtain the following expression for the probability of an undetected error:

$$P_u(E) = (1 + \lambda_0 \cdot n^{-1/10}) 2^{-(n-k)} \sum_{i=2t+1}^{n} \binom{n}{i} p^i (1-p)^{n-i}. \tag{6.37}$$

Let $\epsilon = (2t + 1)/n$. Then the summation of (6.37) can be upper bounded as follows [13]:

$$\sum_{i=n\epsilon}^{n} \binom{n}{i} p^i (1-p)^{n-i} \leq 2^{-nE(\epsilon, p)} \tag{6.38}$$

provided that $p < \epsilon$, where

$$E(\epsilon, p) = H(p) - (\epsilon - p)H'(p) - H(\epsilon)$$
$$H(x) = -x \log_2 x - (1 - x) \log_2 (1 - x),$$

and

$$H'(x) = \log_2 \frac{1 - x}{x}.$$

$E(\epsilon, p)$ is positive for $\epsilon > p$. Combining (6.37) and (6.38), we obtain the following upper bound on $P_u(E)$ for $\epsilon > p$:

$$P_u(E) \leq (1 + \lambda_0 \cdot n^{-1/10}) 2^{-nE(\epsilon, p)} 2^{-(n-k)} \tag{6.39}$$

For $p < \epsilon$ and sufficient large n, $P_u(E)$ can be made very small. For $p \geq \epsilon$, we can also derive a bound on $P_u(E)$. It is clear from (6.37) that

$$P_u(E) \leq (1 + \lambda_0 \cdot n^{-1/10}) 2^{-(n-k)} \sum_{i=0}^{n} \binom{n}{i} p^i (1-p)^{n-i}.$$

Since

$$\sum_{i=0}^{n} \binom{n}{i} p^i (1-p)^i = 1,$$

we obtain the following upper bound on $P_u(E)$:

$$P_u(E) \leq (1 + \lambda_0 \cdot n^{-1/10}) 2^{-(n-k)}. \tag{6.40}$$

Ex.1020 / Page 198 of 305
Sandisk Technologies, Inc. et al

We see that, for $p \geq \epsilon$, $P_u(E)$ still decreases exponentially with the number of parity-check digits, $n - k$. If we use sufficient large number of parity-check digits, the probability of an undetected error $P_u(E)$ will become very small. Now, we may summarize the 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:

$$P_u(E) \leq \begin{cases} (1 + \lambda_0 \cdot n^{-1/10}) 2^{-n[1-R+E(\epsilon,p)]} & \text{for } p < \epsilon \\ (1 + \lambda_0 \cdot n^{-1/10}) 2^{-n(1-R)} & \text{for } p \geq \epsilon \end{cases} \tag{6.41}$$

where $\epsilon = (2t + 1)/n$, $R = k/n$, and λ_0 is a constant.

The analysis above indicates that primitive BCH codes are very effective for error detection on a BSC. Even though it has not been proved, we believe that the probability of an undetected error for any primitive BCH code satisfies the upper bound $2^{-(n-k)}$.

For the nonbinary case, the weight distribution of Reed–Solomon codes has been completely determined [18,30,31]. For a t-error-correcting Reed–Solomon code of length $q - 1$ with symbols from GF(q), the number of code vectors of weight j is

$$A_j = \binom{q-1}{j} \sum_{i=0}^{j-2t-1} (-1)^i \binom{j}{i} (q^{j-2t-i} - 1).$$

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) which 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 $\mathbf{g}(X)$ which 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 found 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. Find the generator polynomial of the double-error-correcting Reed–Solomon code of length $2^4 - 1$ with symbols from GF(2^4). Let α be a primitive element in GF(2^4) given by Table 2.8. Decode the received polynomial $\mathbf{r}(X) = \alpha X^3 + \alpha^{11} X^7$.

6.5. Prove that the syndrome components S_i and S_{2i} are related by $S_{2i} = S_i^2$.

6.6. 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 $2t + 1$.

Ex.1020 / Page 199 of 305
Sandisk Technologies, Inc. et al

[*Hint:* Let $n = l(2t + 1)$. Show that $(X^n + 1)/(X^l + 1)$ is a code polynomial of weight $2t + 1$.]

6.7. Show that the t-error-correcting Reed–Solomon code with symbols from GF(2^m) generated by

$$\mathbf{g}(X) = (X + \alpha)(X + \alpha^2) \cdots (X + \alpha^{2t})$$

has minimum distance exactly $2t - 1$, where α is a primitive element in GF(2^m).

6.8. 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.

6.9. 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.

6.10. 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).

6.11. Devise a syndrome computation circuit for the binary double-error-correcting (31, 21) BCH code.

6.12. Devise a Chien's searching circuit for the binary double-error-correcting (31, 21) BCH code.

6.13. Devise an encoding circuit for the single-error-correcting Reed–Solomon code of length 15 with symbols from GF(2^4). Use Table 2.8.

6.14. Devise a syndrome computation circuit for the single-error-correcting Reed–Solomon code of length 15 with symbols from GF(2^4).

6.15. Let β be any nonzero element in GF(2^m). Let l_0 be any integer. Let $\mathbf{g}(X)$ be the lowest-degree polynomial over GF(2) which has

$$\beta^{l_0}, \beta^{l_0+1}, \ldots, \beta^{l_0+d-2}$$

as its roots.

(a) Express $\mathbf{g}(X)$ in terms of the minimal polynomials of its roots.

(b) Determine the smallest integer n such that $\mathbf{g}(X)$ divides $X^n + 1$.

(c) Show that the cyclic code of length n generated by $\mathbf{g}(X)$ has minimum distance at least d.

Remark. The code defined above is a binary BCH code in general form. The code is called a BCH code with designed distance d. If $l_0 = 1$, $d = 2t + 1$, and β is a primitive element in GF(2^m), the code is a primitive binary t-error-correcting BCH code.

6.16. Consider the Galois field GF(2^6) given by Table 6.2. Let $\beta = \alpha^3$, $l_0 = 2$, and $d = 5$. Use the results of Problem 6.15 to determine the generator polynomial of the BCH code which has

$$\beta^2, \beta^3, \beta^4, \beta^5$$

as its roots. What is the length of this code?

6.17. In Problem 6.15, let $l_0 = -t$ and $d = 2t + 2$. Then we obtain a BCH code of designed distance $2t + 2$ whose generator polynomial has

$$\beta^{-t}, \ldots, \beta^{-1}, \beta^0, \beta^1, \ldots, \beta^t$$

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.]

Ex.1020 / Page 200 of 305
Sandisk Technologies, Inc. et al

REFERENCES

1. A. Hocquenghem, "Codes corecteurs d'erreurs," *Chiffres*, 2, pp. 147–156, 1959.
2. R. C. Bose and D. K. Ray-Chaudhuri, "On a Class of Error Correcting Binary Group Codes," *Inf. Control*, 3, pp. 68–79, March 1960.
3. W. W. Peterson, "Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes," *IRE Trans. Inf. Theory*, IT-6, pp. 459–470, 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, pp. 107–214, June 1961.
5. I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *J. Soc. Ind. Appl. Math.*, 8, pp. 300–304, June 1960.
6. R. T. Chien, "Cyclic Decoding Procedure for the Bose-Chaudhuri-Hocquenghem Codes," *IEEE Trans. Inf. Theory*, IT-10, pp. 357–363, October 1964.
7. G. D. Forney, "On Decoding BCH Codes," *IEEE Trans. Inf. Theory*, IT-11, pp. 549–557, October 1965.
8. E. R. Berlekamp, "On Decoding Binary Bose-Chaudhuri-Hocquenghem Codes," *IEEE Trans. Inf. Theory*, IT-11, pp. 577–580, 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. Inf. Theory*, IT-11, pp. 580–585, October 1965.
11. J. L. Massey, "Shift-Register Synthesis and BCH Decoding," *IEEE Trans. Inf. Theory*, IT-15, pp. 122–127, January 1969.
12. H. O. Burton, "Inversionless Decoding of Binary BCH Codes," *IEEE Trans. Inf. Theory*, IT-17, pp. 464–466, July 1971.
13. W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2nd ed., MIT Press, Cambridge, Mass., 1970.
14. T. Kasami, N. Tokura, Y. Iwadare, and Y. Inagaki, *Coding Theory*, Corona, Tokyo, 1974 (in Japanese).
15. F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, 1977.
16. H. B. Mann, "On the Number of Information Symbols in Bose-Chaudhuri Codes," *Inf. Control*, 5, pp. 153–162, June 1962.
17. C. L. Chen, "High-Speed Decoding of BCH Codes," *IEEE Trans. Inf. Theory*, IT-27(2), pp. 254–256, March 1981.
18. T. Kasami, S. Lin, and W. W. Peterson, "Some Results on Weight Distributions of BCH Codes," *IEEE Trans. Inf. Theory*, IT-12(2), p. 274, April 1966.
19. 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.
20. J. K. Wolf, "Adding Two Information Symbols to Certain Nonbinary BCH Codes and Some Applications," *Bell Syst. Tech. J.*, 48, pp. 2405–2424, 1969.
21. V. D. Goppa, "A New Class of Linear Codes," *Probl. Peredachi Inf.*, 6(3), pp. 24–30, September 1970.
22. V. D. Goppa, "Rational Representation of Codes and (L, g) Codes," *Probl. Peredachi Inf.*, 7(3), pp. 41–49, September 1971.

Ex.1020 / Page 201 of 305
Sandisk Technologies, Inc. et al

23. N. J. Patterson, "The Algebraic Decoding of Goppa Codes," *IEEE Trans. Inf. Theory*, IT-21, pp. 203–207, March 1975.
24. E. R. Berlekamp, "Goppa Codes," *IEEE Trans. Inf. Theory*, IT-19(5), pp. 590–592, September 1973.
25. Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A Method for Solving Key Equation for Decoding Goppa Codes," *Inf. Control*, 27, pp. 87–99, January 1975.
26. R. E. Blahut, "Transform Techniques for Error Control Codes," *IBM J. Res. Dev.*, 23(3), May 1979.
27. 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, N.C., 1968.
28. 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. Inf. Theory*, IT-25(1), pp. 110–112, January 1979.
29. V. M. Sidelńikov, "Weight Spectrum of Binary Bose-Chaudhuri-Hocquenghem Code," *Probl. Inf. Transm.*, 7(1), pp. 11–17, 1971.
30. E. F. Assmus, Jr., H. F. Mattson, Jr., and R. J. Turyn, "Cyclic Codes," Report No. AFCRL-65-332, Air Force Cambridge Research Labs., Bedford, Mass, April 1965.
31. G. D. Forney, Jr., *Concatenated Codes*, MIT Press, Cambridge, Mass, 1966.

Ex.1020 / Page 202 of 305
Sandisk Technologies, Inc. et al

7

Majority-Logic Decoding for Cyclic Codes

The majority-logic decoding presented in this and the next chapters is another 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 a class of multiple-error-correcting codes discovered by Muller [2]. Reed's algorithm was later extended and generalized by many coding investigators. The first unified formulation of majority-logic decoding algorithms was due to Massey [3].

Most majority-logic decodable codes found so far are cyclic codes. Important cyclic codes of this category are presented in this and the next chapters.

7.1 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 vector $\mathbf{v}$ in C and any vector $\mathbf{w}$ in C_d, the inner product of $\mathbf{v}$ and $\mathbf{w}$ is zero, that is,

$$\mathbf{w} \cdot \mathbf{v} = w_0 v_0 + w_1 v_1 + \cdots + w_{n-1} v_{n-1} = 0. \tag{7.1}$$

In fact, an n-tuple $\mathbf{v}$ is a code vector in C if and only if, for any vector $\mathbf{w}$ in C_d, $\mathbf{w} \cdot \mathbf{v} = 0$. The equality of (7.1) is called a *parity-check* equation. Clearly, there are $2^{(n-k)}$ such parity-check equations.

Now suppose that a code vector $\mathbf{v}$ in C is trasmitted. Let $\mathbf{e} = (e_0, e_1, \ldots, e_{n-1})$ and $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$ be the error vector and the received vector respectively. Then

Ex.1020 / Page 203 of 305
Sandisk Technologies, Inc. et al

$$\mathbf{r} = \mathbf{v} + \mathbf{e}. \tag{7.2}$$

For any vector $\mathbf{w}$ in the dual code C_d, we can form the following linear sum of the received digits:

$$A = \mathbf{w} \cdot \mathbf{r} = w_0 r_0 + w_1 r_1 + \cdots + w_{n-1} r_{n-1}, \tag{7.3}$$

which is called a *parity-check sum* or simply *check sum*. If the received vector $\mathbf{r}$ is a code vector in C, this parity-check sum, A, *must be* zero; however, if $\mathbf{r}$ is not a code vector in C, then A *may not* be zero. Combining (7.2) and (7.3) 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 $\mathbf{e}$:

$$A = w_0 e_0 + w_1 e_1 + \cdots + w_{n-1} e_{n-1}. \tag{7.4}$$

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,

$$\begin{aligned}
\mathbf{w}_1 &= (w_{10}, w_{11}, \ldots, w_{1,n-1}),\\
\mathbf{w}_2 &= (w_{20}, w_{21}, \ldots, w_{2,n-1}),\\
&\vdots\\
\mathbf{w}_J &= (w_{J0}, w_{J1}, \ldots, w_{J,n-1}),
\end{aligned}$$

which 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 ith 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, let us form J parity-check sums from these J orthogonal vectors,

$$\begin{aligned}
A_1 &= \mathbf{w}_1 \cdot \mathbf{r} = w_{10} r_0 + w_{11} r_1 + \cdots + w_{1,n-1} r_{n-1}\\
A_2 &= \mathbf{w}_2 \cdot \mathbf{r} = w_{20} r_0 + w_{21} r_1 + \cdots + w_{2,n-1} r_{n-1}\\
&\vdots\\
A_J &= \mathbf{w}_J \cdot \mathbf{r} = w_{J0} r_0 + w_{J1} r_1 + \cdots + w_{J,n-1} r_{n-1}.
\end{aligned} \tag{7.5}$$

Since $w_{1,n-1} = w_{2,n-1} = \cdots = w_{J,n-1} = 1$, these J check sums are related to the error digits in the following manner:

$$\begin{aligned}
A_1 &= w_{10} e_0 + w_{11} e_1 + \cdots + w_{1,n-2} e_{n-2} + e_{n-1}\\
A_2 &= w_{20} e_0 + w_{21} e_1 + \cdots + w_{2,n-2} e_{n-2} + e_{n-1}\\
&\vdots\\
A_J &= w_{J,0} e_0 + w_{J,1} e_1 + \cdots + w_{J,n-2} e_{n-2} + e_{n-1}.
\end{aligned} \tag{7.6}$$

Ex.1020 / Page 204 of 305
Sandisk Technologies, Inc. et al

We see that the error digit e_{n-1} is checked by all the check sums above. Because of the second property of the orthogonal vectors, $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_J$, 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 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 $\mathbf{e} = (e_0, e_1, \ldots, e_{n-1})$ (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 one-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 one-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 facts above, 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 described above is called *one-step majority-logic decoding* [3]. 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 $t_{\mathrm{ML}} = \lfloor J/2 \rfloor$ 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 $t_{\mathrm{ML}} = \lfloor J/2 \rfloor$ 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: 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, an example will be helpful in clarifying the notions developed above.

Ex.1020 / Page 205 of 305
Sandisk Technologies, Inc. et al

Example 7.1

Consider a (15, 7) cyclic code generated by the polynomial

$$g(X) = 1 + X^4 + X^6 + X^7 + X^8.$$

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 **H**:

Digit positions:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\mathbf{w}_1 = \mathbf{h}_3 =$	(0	0	0	1	0	0	0	0	0	0	0	1	1	0	1),
$\mathbf{w}_2 = \mathbf{h}_1 + \mathbf{h}_5 =$	(0	1	0	0	0	1	0	0	0	0	0	0	0	1	1),
$\mathbf{w}_3 = \mathbf{h}_0 + \mathbf{h}_2 + \mathbf{h}_6 =$	(1	0	1	0	0	0	1	0	0	0	0	0	0	0	1),
$\mathbf{w}_4 = \mathbf{h}_7 =$	(0	0	0	0	0	0	0	1	1	0	1	0	0	0	1).

We see that all these four vectors have a 1 at the 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 **r** be the received vector. The four parity-check sums formed from these orthogonal vectors are related to the error digits as follows:

$$\begin{aligned}
A_1 &= \mathbf{w}_1 \cdot \mathbf{r} = e_3 + e_{11} + e_{12} + e_{14} \\
A_2 &= \mathbf{w}_2 \cdot \mathbf{r} = e_1 + e_5 + e_{13} + e_{14} \\
A_3 &= \mathbf{w}_3 \cdot \mathbf{r} = e_0 + e_2 + e_6 + e_{14} \\
A_4 &= \mathbf{w}_4 \cdot \mathbf{r} = e_7 + e_8 + e_{10} + e_{14}.
\end{aligned}$$

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 **e**, the one-step majority-logic decoding always results in correct decoding of e_{14}. Since 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 which cannot be corrected. Consider an error pattern **e** with three errors which are 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 $A_4 = 1$. Since 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

Ex.1020 / Page 206 of 305
Sandisk Technologies, Inc. et al

is actually a BCH code with minimum distance 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 be easily implemented. First, from the null space C_d of the code, we determine a set of J vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_J$ that are orthogonal on the highest-order digit position, X^{n-1}. Then, J parity-check sums $A_1, A_2, \ldots, A_J$ orthogonal on the error digit e_{n-1} are formed from these J orthogonal vectors and the received vector $\mathbf{r}$. From (7.5), 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 one-half of 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 7.1. The error correction procedure can be described as follows:

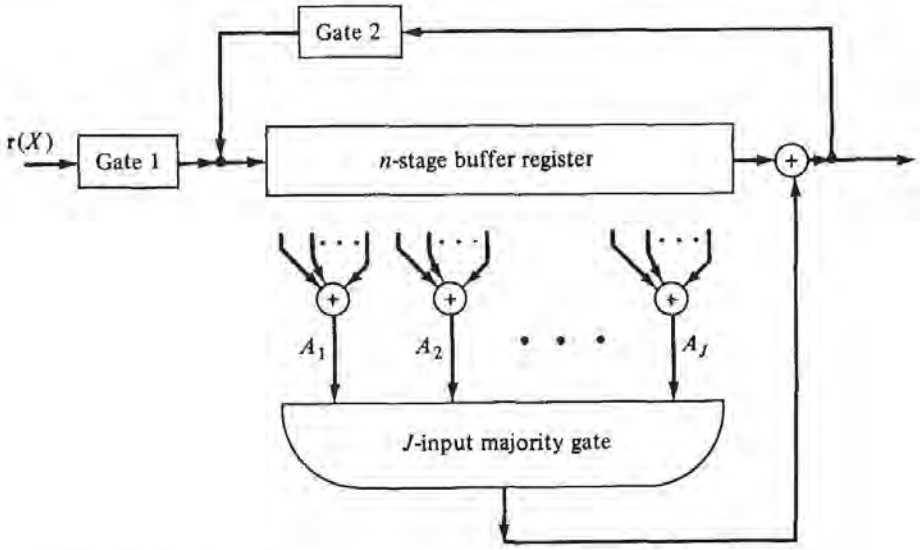


Figure 7.1 General type II one-step majority-logic decoder.

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 same manner as the first received digit was. The decoder repeats steps 2 and 3.

Ex.1020 / Page 207 of 305
Sandisk Technologies, Inc. et al

Step 5. The received vector is decoded digit by digit in the manner above 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 gate should be all 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 decoder shown in Figure 7.1 is called the type II one-step majority-logic decoder [3].

The type II one-step majority-logic decoder for the (15, 7) BCH code considered in Example 7.1 is shown in Figure 7.2.

The parity-check sums orthogonal on an error digit can also be formed from the syndrome digits. Let

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \mathbf{h}_2 \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{h}_{n-k-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & p_{00} & p_{01} & \cdots & p_{0,k-1} \\ 0 & 1 & 0 & 0 & \cdots & 0 & p_{10} & p_{11} & & p_{1,k-1} \\ 0 & 0 & 1 & 0 & \cdots & 0 & p_{20} & p_{21} & & p_{2,k-1} \\ \cdot & & & & & & \cdot & \cdot & & \cdot \\ \cdot & & & & & & \cdot & \cdot & & \cdot \\ \cdot & & & & & & \cdot & \cdot & & \cdot \\ 0 & 0 & 0 & 0 & \cdots & 1 & p_{n-k-1,0} & p_{n-k-1,1} & \cdots & p_{n-k-1,k-1} \end{bmatrix}$$

be the parity-check matrix for an (n, k) cyclic code C in systematic form. Since the orthogonal vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_J$ are vectors in the row space of $\mathbf{H}$, they are linear combinations of rows of $\mathbf{H}$. Let

$$\begin{aligned} \mathbf{w}_j &= (w_{j0}, w_{j1}, \ldots, w_{j,n-1}) \\ &= a_{j0}\mathbf{h}_0 + a_{j1}\mathbf{h}_1 + \cdots + a_{j,n-k-1}\mathbf{h}_{n-k-1}. \end{aligned}$$

Because of the systematic structure of $\mathbf{H}$, we see that

$$w_{j0} = a_{j0}, \quad w_{j1} = a_{j1}, \quad \ldots, \quad w_{j,n-k-1} = a_{j,n-k-1}. \tag{7.7}$$

Let $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$ be the received vector. Then the syndrome of $\mathbf{r}$ is

$$\mathbf{s} = (s_0, s_1, \ldots, s_{n-k-1}) = \mathbf{r} \cdot \mathbf{H}^T,$$

where the ith syndrome digit is

$$s_i = \mathbf{r} \cdot \mathbf{h}_i \tag{7.8}$$

for $0 \leq i < n - k$. Now, consider the parity-check sum

$$\begin{aligned} A_j &= \mathbf{w}_j \cdot \mathbf{r} \\ &= (a_{j0}\mathbf{h}_0 + a_{j1}\mathbf{h}_1 + \cdots + a_{j,n-k-1}\mathbf{h}_{n-k-1}) \cdot \mathbf{r} \\ &= a_{j0}\mathbf{r} \cdot \mathbf{h}_0 + a_{j1}\mathbf{r} \cdot \mathbf{h}_1 + \cdots + a_{j,n-k-1}\mathbf{r} \cdot \mathbf{h}_{n-k-1}. \end{aligned} \tag{7.9}$$

From (7.7), (7.8), and (7.9), we obtain

$$A_j = w_{j0}s_0 + w_{j1}s_1 + \cdots + w_{j,n-k-1}s_{n-k-1}. \tag{7.10}$$

Thus, the check sum A_j is simply a *linear sum of the syndrome digits with coefficients being the first $n - k$ digits of the orthogonal vector* $\mathbf{w}_j$. Based on (7.10), we obtain a different implementation of the one-step majority-logic decoding as shown in Figure 7.3 (the received vector can be shifted into the syndrome register from the *right*

Ex.1020 / Page 208 of 305
Sandisk Technologies, Inc. et al

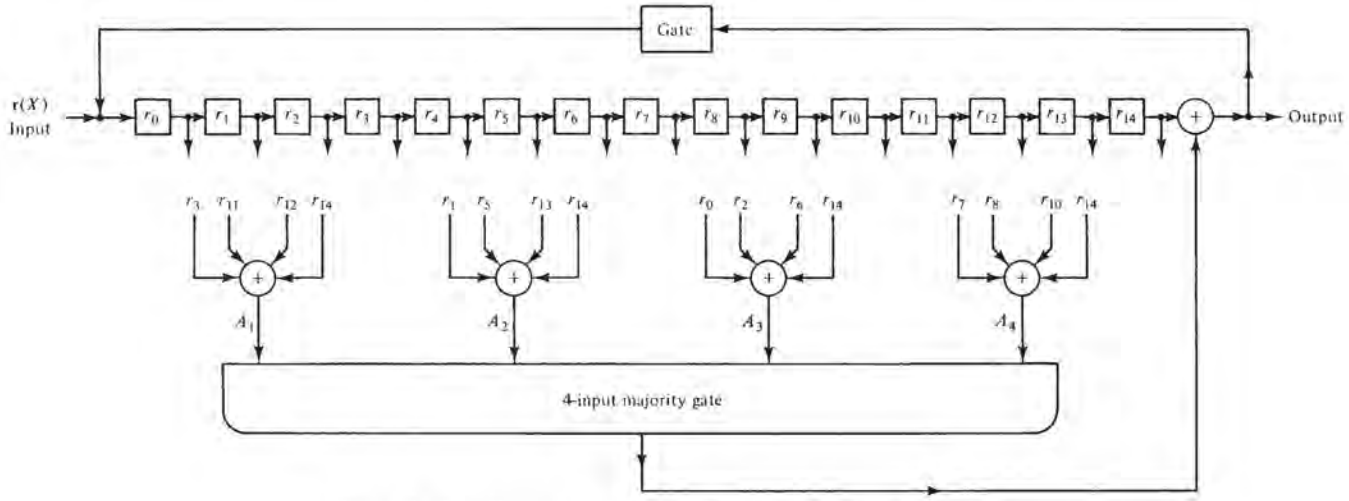


Figure 7.2 Type II one-step majority-logic decoder for the (15, 7) BCH code.

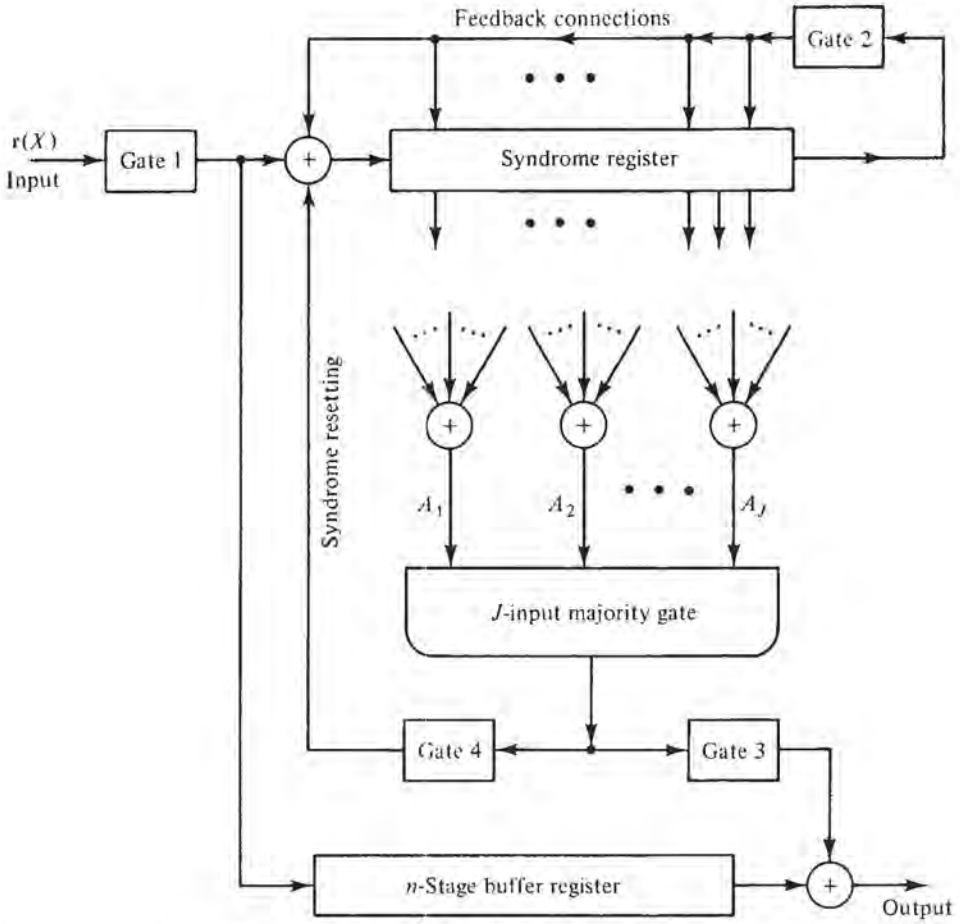


Figure 7.3 General type-I one-step majority-logic decoder.

end). This decoder is called the type I one-step majority-logic decoder [3]. The error correction procedure is described 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 of e_{n-1} on the syndrome is removed (with gate 4 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 manner above until a total of n shifts of the buffer and the syndrome registers.

Ex.1020 / Page 210 of 305
Sandisk Technologies, Inc. et al

At the completion of the decoding operation, the syndrome register should contain only zeros if the decoder ouput is a code vector. If the syndrome register does not contain all zeros at the end of the decoding, an uncorrectable error pattern has been detected. If we are interested only in decoding the received message digits but not the received parity digits, the buffer register needs only to store 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 7.2

Consider the (15, 7) BCH code given in Example 7.1. 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:

$$A_1 = s_3, \qquad A_2 = s_1 + s_5, \qquad A_3 = s_0 + s_2 + s_6, \qquad A_4 = s_7.$$

Based on these sums we construct the type I one-step majority-logic decoder for the (15, 7) BCH code as shown in Figure 7.4. Suppose that the all-zero code vector (0, 0, . . . , 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

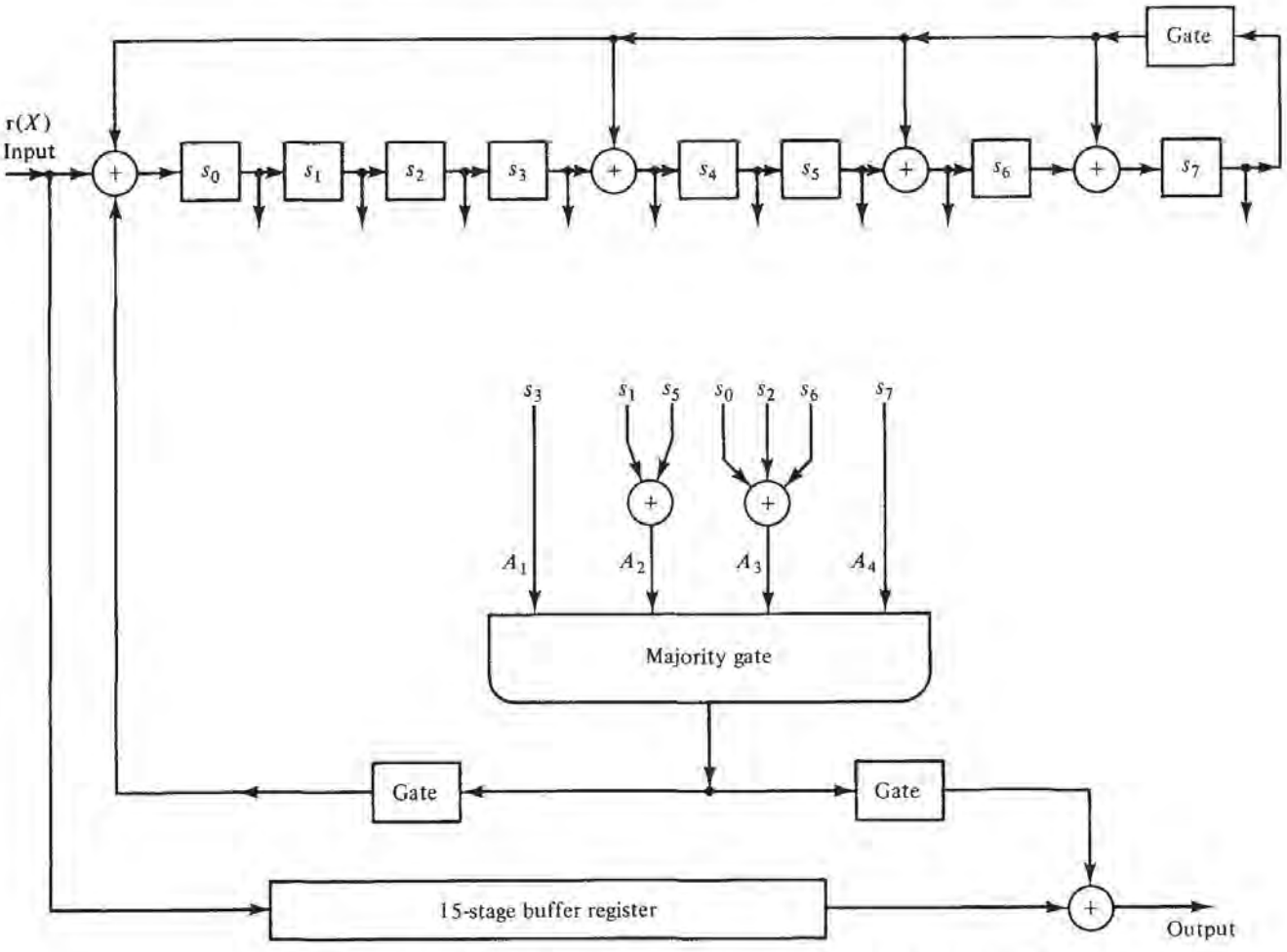


Figure 7.4 Type I one-step majority-logic decoder for (15, 7) BCH code.

Ex.1020 / Page 211 of 305
Sandisk Technologies, Inc. et al

orthogonal on e_{14} are

$$A_1 = 1, \quad A_2 = 0, \quad A_3 = 1, \quad A_4 = 1.$$

Since 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 shift the buffer and syndrome registers 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}. Shift both the buffer and syndrome registers once more; the received digit r_{13} would be corrected and the syndrome register would 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 larger J compared to $d_{\min} - 1$. When J is small compared to $d_{\min} - 1$, 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 is answered by Theorem 7.1.

Theorem 7.1. 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

$$J \leq \left\lfloor \frac{n-1}{\delta - 1} \right\rfloor. \tag{7.11}$$

Proof. Suppose that there exist J vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_J$ in the dual code of C which are orthogonal on the highest-order digit position, X^{n-1}. Since each of these J vectors has weight 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 $J \leq (n - 1)/(\delta - 1)$. Since 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 minimum distance 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 7.1 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, the code has minimum distance at least $J + 1$. The proof of this is left as a problem.

As we pointed out earlier in this section, one-step majority-logic decoding is most effective for cyclic codes which are completely orthogonalizable. Unfortunately, there exist very few good cyclic codes in this category. The double-error-correcting (15, 7) code considered in Example 7.1 is the only known BCH code that is completely

Ex.1020 / Page 212 of 305
Sandisk Technologies, Inc. et al

orthogonalizable in one step. In the next two sections, several small classes of one-step majority-logic decodable cyclic codes will be presented. Two of the classes are proved to be completely orthogonalizable.

7.2 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 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 $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ 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., $v_\infty = v_0 + v_1 + \cdots + v_{n-1}$). Adding v_∞ to $\mathbf{v}$ results in the following vector of $n + 1 = 2^m$ components:

$$\mathbf{v}_e = (v_\infty, v_0, v_1, \ldots, v_{n-1}).$$

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 code vectors 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 $\mathbf{v}_e = (v_\infty, v_0, v_1, \ldots, v_{2^m-2})$ 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 GF(2^m) and $a \neq 0$. This permutation is called an *affine permutation*. Application of an affine permutation to a vector of 2^m components results in another vector of 2^m components.

Example 7.3

Consider the following vector of 16 components, which are numbered with the elements of GF(2^4) (using Table 2.8):

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}
(1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0).

Now, we apply the affine permutation

$$Z = \alpha Y + \alpha^{14}$$

to the components of the vector above. The resultant vector is

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}
(0	0	1	1	0	0	0	0	1	0	1	0	1	0	0	1).

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 code vector in C_e

Ex.1020 / Page 213 of 305
Sandisk Technologies, Inc. et al

into another code vector in C_e. In the following 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 nonegative 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 7.4

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:

$$20 = 0 + 0 \cdot 2 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4,$$
$$17 = 1 + 0 \cdot 2 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4,$$
$$16 = 0 + 0 \cdot 2 + 0 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4,$$
$$5 = 1 + 0 \cdot 2 + 1 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4,$$
$$4 = 0 + 0 \cdot 2 + 1 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4,$$
$$1 = 1 + 0 \cdot 2 + 0 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4,$$
$$0 = 0 + 0 \cdot 2 + 0 \cdot 2^2 + 0 \cdot 2^3 + 0 \cdot 2^4.$$

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 7.2. 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 References 4 and 5. A cyclic code of length $2^m - 1$ whose generator polynomial satisfies the conditions given in Theorem 7.2 is said to have the *doubly transitive invariant (DTI) property*.

Given a code C_e of length $n = 2^m$ which 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 permuation $Z = \alpha Y$ to a vector $(v_\infty, v_0, v_1, \ldots, v_{2^m-2})$ 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

$$(v_\infty, v_{2^m-2}, v_0, v_1, \ldots, v_{2^m-3}),$$

Ex.1020 / Page 214 of 305
Sandisk Technologies, Inc. et al

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:

$$X^{2^m-1} + 1 = (1 + X^J)(1 + X^J + X^{2J} + \cdots + X^{(L-1)J}).$$

Let

$$\pi(X) = 1 + X^J + X^{2J} + \cdots + X^{(L-1)J}. \tag{7.12}$$

From Theorem 2.8 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). Since $(\alpha^L)^J = \alpha^{2^m-1} = 1$, the polynomial $X^J + 1$ has $\alpha^0 = 1, \alpha^L, \alpha^{2L}, \ldots, \alpha^{(J-1)L}$ 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 following: $H(X)$ 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

$$H(X) = \text{LCM } \{\text{minimal polynomials } \phi_i(X) \text{ of the roots of } H(X)\}. \tag{7.13}$$

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 7.2 that C' has the doubly transitive invariant 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

$$X^{2^m-1} + 1 = G(X)H(X).$$

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

$$\mathbf{g}(X) = X^{2^m-k-1}G(X^{-1}), \tag{7.14}$$

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}. Since $\pi(X)$ 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 $X\pi(X), X^2\pi(X), \ldots, X^{J-1}\pi(X)$ are also code polynomials in C'. From (7.12), we see that, for $i \neq j$, $X^i\pi(X)$ and $X^j\pi(X)$ do not have any common term. Let $\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_{J-1}$ be the J corresponding code vectors of $\pi(X), X\pi(X), \ldots, X^{J-1}\pi(X)$. The weight of each of these vectors is L. Adding an overall parity-check digit to each of these vectors, we obtain J vectors $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{J-1}$ of length 2^m which are code vectors 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 $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{J-1}$ have the following properties:

Ex.1020 / Page 215 of 305
Sandisk Technologies, Inc. et al

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 $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{J-1}$. This permutation carries $\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{J-1}$ into J vectors $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{J-1}$ 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 the location α^{2^m-2}. Thus, the vectors $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{J-1}$ are orthogonal on the digit at location α^{2^m-2}. Deleting the digit at location α^{∞} from $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{J-1}$, we obtain J vectors $\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_{J-1}$ of length $2^m - 1$, which are vectors in C' and are orthogonal on the digit at the 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 $t_{\mathrm{ML}} = \lfloor J/2 \rfloor$ of fewer errors. For convenience, we call this code a *type* 0 one-step majority-logic decodable DTI code.

Example 7.5

Let $m = 5$. The polynomial $X^{2^4-1} + 1 = X^{15} + 1$ can be factored 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 $\phi_1(X) = 1 + X + X^4$. Since $\alpha^{15} = 1$, the polynomial $1 + X^5$ has 1, α^3, α^6, α^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 minimum polynomials $\phi_1(X) = 1 + X + X^4$. The roots α^5 and α^{10} are conjugates and they have $\phi_5(X) = 1 + X + X^2$ as their minimal polynomial. Hence,

$$\begin{aligned} H(X) &= \phi_1(X)\phi_5(X) = (1 + X + X^4)(1 + X + X^2) \\ &= 1 + X^3 + X^4 + X^5 + X^6. \end{aligned}$$

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

$$\mathbf{g}(X) = X^9G(X^{-1}) = 1 + X + X^4 + X^5 + X^6 + X^9.$$

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

Ex.1020 / Page 216 of 305
Sandisk Technologies, Inc. et al

	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}
$\mathbf{v}_0 = (1$	0	0	0	0	1	0	0	0	0	1	0	0	0	0)
$\mathbf{v}_1 = (0$	1	0	0	0	0	1	0	0	0	0	1	0	0	0)
$\mathbf{v}_2 = (0$	0	1	0	0	0	0	1	0	0	0	0	1	0	0)
$\mathbf{v}_3 = (0$	0	0	1	0	0	0	0	1	0	0	0	0	1	0)
$\mathbf{v}_4 = (0$	0	0	0	1	0	0	0	0	1	0	0	0	0	1)

which are code vectors 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	0)
$\mathbf{u}_1 = (1$	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0)
$\mathbf{u}_2 = (1$	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0)
$\mathbf{u}_3 = (1$	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0)
$\mathbf{u}_4 = (1$	0	0	0	0	1	0	0	0	0	1	0	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	1)
$\mathbf{z}_1 = (0$	0	1	0	0	0	1	0	0	0	0	0	0	0	1	1)
$\mathbf{z}_2 = (0$	1	0	1	0	0	0	1	0	0	0	0	0	0	0	1)
$\mathbf{z}_3 = (1$	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1)
$\mathbf{z}_4 = (0$	0	0	0	1	0	0	0	0	0	0	0	1	1	0	1)

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	0	1	1	0	1	0	0	0	1)
$\mathbf{w}_1 = (0$	1	0	0	0	1	0	0	0	0	0	0	0	1	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

Ex.1020 / Page 217 of 305
Sandisk Technologies, Inc. et al

$$A_0 = \mathbf{r} \cdot \mathbf{w}_0 = r_7 + r_8 + r_{10} + r_{14},$$
$$A_1 = \mathbf{r} \cdot \mathbf{w}_1 = r_1 + r_5 + r_{13} + r_{14},$$
$$A_2 = \mathbf{r} \cdot \mathbf{w}_2 = r_0 + r_2 + r_6 + r_{14},$$
$$A_3 = \mathbf{r} \cdot \mathbf{w}_3 = r_4 + r_9 + r_{14},$$
$$A_4 = \mathbf{r} \cdot \mathbf{w}_4 = r_3 + r_{11} + r_{12} + r_{14}.$$

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. The code has minimum distance at least $J + 1 = 5 + 1 = 6$. However, the generator polynomial has weight exactly 6. Thus, 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: $\alpha, \alpha^2, \ldots, \alpha^{L-1}$. Since 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 $\alpha, \alpha^2, \ldots, \alpha^{L-1}$ 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 7.1 that the number of parity-check sums orthogonal on an error digit that can be formed for C is upper bounded by

$$\left\lfloor \frac{2^m - 2}{L - 1} \right\rfloor. \tag{7.15}$$

However, $J = (2^m - 1)/L$. Therefore, for large L, J is either equal to or close to the upper bound of (7.15).

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

$$H_1(X) = (X + 1)H(X). \tag{7.16}$$

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 code vectors. Recall that the J orthogonal vectors $\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_{J-1}$ in C' are obtained from the vectors $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{J-1}$ by deleting the digit at the location α^∞. Since $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{J-1}$ are orthogonal on the digit at the location α^{2^m-2}, there is one and only one vector $\mathbf{z}_i$ that has a "1" at the location α^∞. Since $\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_{J-1}$ all have weight $L + 1$ which is even, all but one of the orthogonal vectors $\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_{J-1}$ 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

Ex.1020 / Page 218 of 305
Sandisk Technologies, Inc. et al

orthogonal on an error digit can be formed. Let

$$G_1(X) = \frac{G(X)}{X+1}. \tag{7.17}$$

Then the generator polynomial for C_1 is

$$\mathbf{g}_1(X) = X^{2^m-k-2}G_1(X^{-1}) = \frac{\mathbf{g}(X)}{X+1}, \tag{7.18}$$

where $\mathbf{g}(X)$ is given by (7.14). 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 7.6

For $m = 4$ and $J = 5$, the type 1 DTI code C_1 that corresponds to the type 0 DTI code given in Example 7.5 is generated by

$$\mathbf{g}_1(X) = \frac{1 + X + X^4 + X^5 + X^6 + X^9}{1 + X}$$
$$= 1 + X^4 + X^6 + X^7 + X^8.$$

It is interesting to note that this code is the (15, 7) BCH code. From Example 7.5 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

$$\mathbf{w}_0 = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1),$$
$$\mathbf{w}_1 = (0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1),$$
$$\mathbf{w}_2 = (1\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1),$$
$$\mathbf{w}_4 = (0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1),$$

which are the same four orthogonal vectors given in Example 7.1.

Since the dual code of type 1 DTI code C_1 has minimum distance $L + 1$, 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

$$n - k = 2^m - (2^\lambda - 1)^l - 1.$$

Ex.1020 / Page 219 of 305
Sandisk Technologies, Inc. et al

TABLE 7.1 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			

A list of one-step majority-logic decodable type 1 DTI codes is given in Table 7.1.

For 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 which 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 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 the (63, 37) DTI code. For large block length, the DTI codes are much less efficient than the BCH codes of the same length and the same error-correcting capability.

7.3 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.

Maximum-Length Codes

For any integer $m \geq 3$, there exists a nontrivial maximum-length code with the following parameters:

Block length: $n = 2^m - 1$
Number of information digits: $k = m$
Minimum distance: $d = 2^{m-1}$.

Ex.1020 / Page 220 of 305
Sandisk Technologies, Inc. et al

The generator polynomial of this code is

$$\mathbf{g}(X) = \frac{X^n + 1}{\mathbf{p}(X)} \tag{7.19}$$

where $\mathbf{p}(X)$ is a primitive polynomial of degree m. This code consists of the all-zero code vector and $2^m - 1$ code vectors of weight 2^{m-1} (see Problem 7.11). 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 $\mathbf{p}(X)$,

$$\mathbf{p}^*(X) = X^m \mathbf{p}(X^{-1}).$$

Since $\mathbf{p}^*(X)$ 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:

$$Q = \{\mathbf{w}(X) = X^i + X^j + X^{n-1} \mid 0 \leq i < j \leq n - 1\} \tag{7.20}$$

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 $\mathbf{p}^*(X)$. This can be carried out 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 start 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. Since the maximum-length code generated by $\mathbf{g}(X)$ of (7.19) has minimum distance exactly 2^{m-1}, it is completely orthogonalizable. The code is capable of correcting $t_{\mathrm{ML}} = 2^{m-2} - 1$ or fewer errors with one-step majority-logic decoding.

Example 7.7

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

$$\begin{aligned}\mathbf{g}(X) &= \frac{X^{15} + 1}{\mathbf{p}(X)} \\ &= 1 + X + X^2 + X^3 + X^5 + X^7 + X^8 + X^{11}.\end{aligned}$$

The null space of this code is generated by

$$\mathbf{p}^*(X) = X^4 \mathbf{p}(X^{-1}) = X^4 + X^3 + 1.$$

Divide $X^{14} + X^{13}$ by $\mathbf{p}^*(X) = X^4 + X^3 + 1$ with long division as shown in the following:

Ex.1020 / Page 221 of 305
Sandisk Technologies, Inc. et al

$$
\begin{array}{r}
X^{10} \\
X^4 + X^3 + 1 \overline{)\, X^{14} + X^{13}} \\
X^{14} + X^{13} + X^{10} \\
\hline
X^{10}
\end{array}
$$

(stop).

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 divide $X^{14} + X^{12}$ by $\mathbf{p}^*(X)$:

$$
\begin{array}{r}
X^{10} + X^9 \quad + X^6 \\
X^4 + X^3 + 1 \overline{)\, X^{14} \qquad + X^{12}} \\
X^{14} + X^{13} \qquad + X^{10} \\
\hline
X^{13} + X^{12} + X^{10} \\
X^{13} + X^{12} \qquad + X^9 \\
\hline
X^{10} + X^9 \\
X^{10} + X^9 + X^6 \\
\hline
X^6
\end{array}
$$

(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

$$\mathbf{w}_3(X) = 1 + X^{11} + X^{14}, \qquad \mathbf{w}_4(X) = X^4 + X^9 + X^{14},$$
$$\mathbf{w}_5(X) = X + X^8 + X^{14}, \qquad \mathbf{w}_6(X) = X^5 + X^7 + X^{14},$$
$$\mathbf{w}_7(X) = X^2 + X^3 + X^{14}.$$

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. The type I and type II one-step majority-logic decoders for this code are shown in Figures 7.5 and 7.6, respectively.

Difference-Set Codes

The formulation of difference-set codes is based on the construction of a *perfect difference set*. Let $P = \{l_0, l_1, l_2, \ldots, l_q\}$ be a set of $q + 1$ nonnegative integers such that

$$0 \leq l_0 < l_1 < l_2 < \cdots < l_q \leq q(q + 1).$$

From this set of integers, it is possible to form $q(q + 1)$ *ordered differences* as follows:

$$D = \{l_j - l_i \mid j \neq i\}.$$

Ex.1020 / Page 222 of 305
Sandisk Technologies, Inc. et al

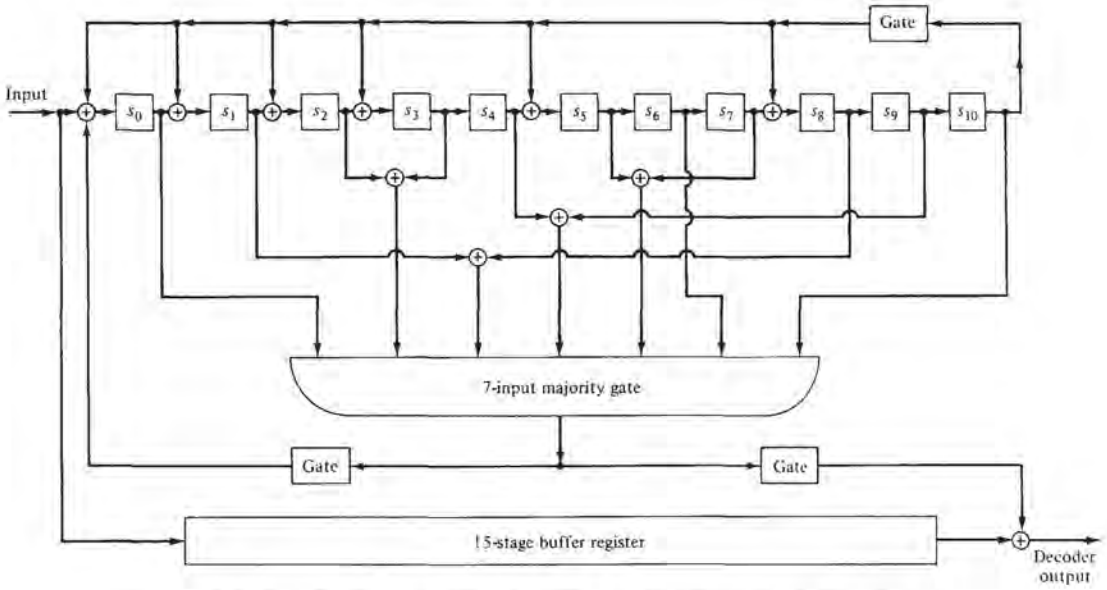


Figure 7.5 Type I majority-logic decoder for the (15, 4) maximum-length code.

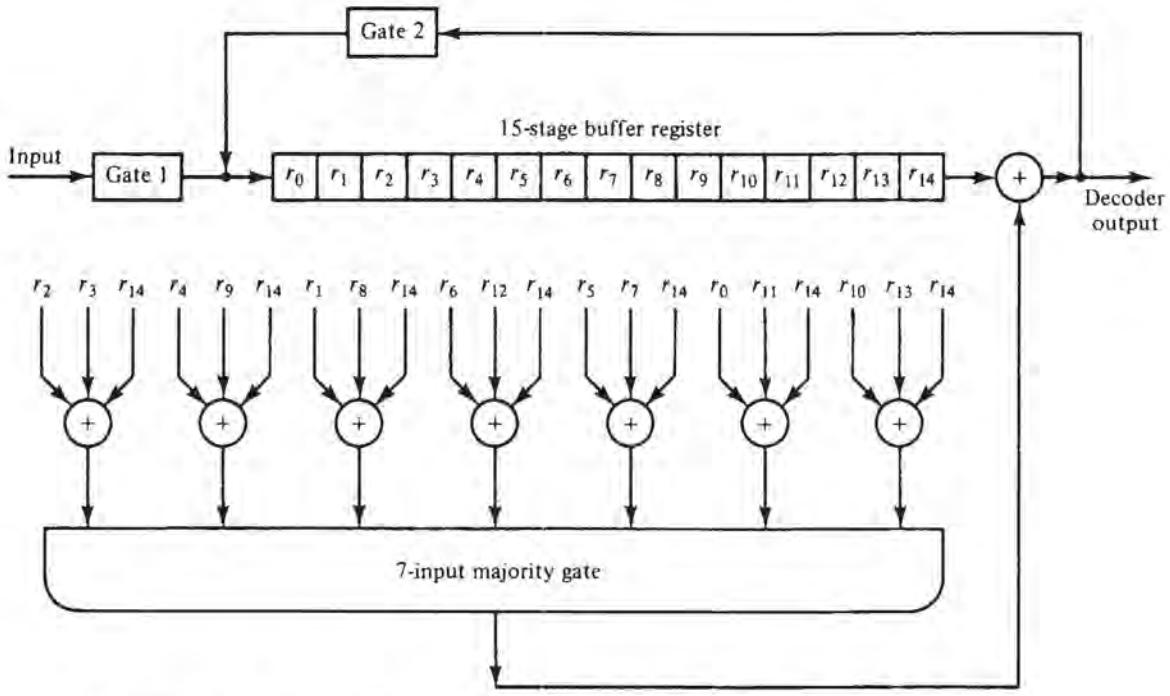


Figure 7.6 Type II majority-logic decoder for the (15, 4) maximum-length code.

Obviously, half of 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 $P' = \{0, l_1 - l_0, l_2 - l_0, \ldots, l_q - l_0\}$ is also a perfect simple difference set.

Example 7.8

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 Reference 10). In what follows we shall only be concerned with $q = 2^s$.

Let $P = \{l_0 = 0, l_1, l_2, \ldots, l_{2^s}\}$ be a perfect simple difference set of order 2^s. Define the polynomial

$$\mathbf{z}(X) = 1 + X^{l_1} + X^{l_2} + \cdots + X^{l_{2^s}}. \tag{7.21}$$

Ex.1020 / Page 224 of 305
Sandisk Technologies, Inc. et al

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,

$$\begin{aligned}\mathbf{h}(X) &= \text{GCD}\,\{\mathbf{z}(X), X^n + 1\} \\ &= 1 + h_1X + h_2X^2 + \cdots + h_{k-1}X^{k-1} + X^k.\end{aligned} \tag{7.22}$$

Then a difference-set code of length n is defined as the cyclic code generated by

$$\begin{aligned}\mathbf{g}(X) &= \frac{X^n + 1}{\mathbf{h}(X)} \\ &= 1 + g_1X + g_2X^2 + \cdots + X^{n-k}.\end{aligned} \tag{7.23}$$

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 7.9

In Example 7.8 we have shown 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 (7.23). Let

$$\begin{aligned}\mathbf{z}^*(X) &= X^{l_{2s}}\mathbf{z}(X^{-1}) \\ &= 1 + \cdots + X^{l_{2s}-l_2} + X^{l_{2s}-l_1} + X^{l_{2s}}.\end{aligned} \tag{7.24}$$

Since $\mathbf{z}(X)$ 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 (7.23). Let

$$\begin{aligned}\mathbf{w}_0(X) &= X^{n-1-l_{2s}}\mathbf{z}^*(X) \\ &= X^{n-1-l_{2s}} + \cdots + X^{n-1-l_2} + X^{n-1-l_1} + X^{n-1}.\end{aligned}$$

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 (7.23). Now let

Ex.1020 / Page 225 of 305
Sandisk Technologies, Inc. et al

$$\mathbf{w}_i(X) = X^{l_i - l_{i-1} - 1} + X^{l_i - l_{i-2} - 1} + \cdots + X^{l_i - l_1 - 1} + X^{l_i - 1} + X^{n-1-l_{2s}+l_i} + X^{n-1-l_{2s-1}+l_i} + \cdots + X^{n-1} \tag{7.25}$$

be the vector obtained by shifting $\mathbf{w}_0(X)$ cyclically to the right l_i times. Since $\{l_0 = 0, l_1, l_2, \ldots, l_{2^s}\}$ 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, $\mathbf{w}_0(X), \mathbf{w}_1(X), \ldots, \mathbf{w}_{2^s}(X)$ form a set of $J = 2^s + 1$ polynomials orthogonal on the digit at the position X^{n-1}. Since the code generated by $\mathbf{g}(X)$ of (7.23) is proved to have minimum distance $2^s + 2$, it is completely orthogonalizable and is capable of correcting $t_{\mathrm{ML}} = 2^{s-1}$ or fewer errors.

Example 7.10

Consider the code given in Example 7.9, which is specified by the perfect difference set $P = \{0, 2, 7, 8, 11\}$ of order 2^2. Thus, we have

$$\mathbf{z}^*(X) = X^{11}\mathbf{z}(X^{-1}) = 1 + X^3 + X^4 + X^9 + X^{11}$$

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 7.7. 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 polynomials and their corresponding perfect simple difference sets is given in Table 7.2.

There are other one-step majority-logic decodable cyclic codes which we present in Chapter 8.

Ex.1020 / Page 226 of 305
Sandisk Technologies, Inc. et al

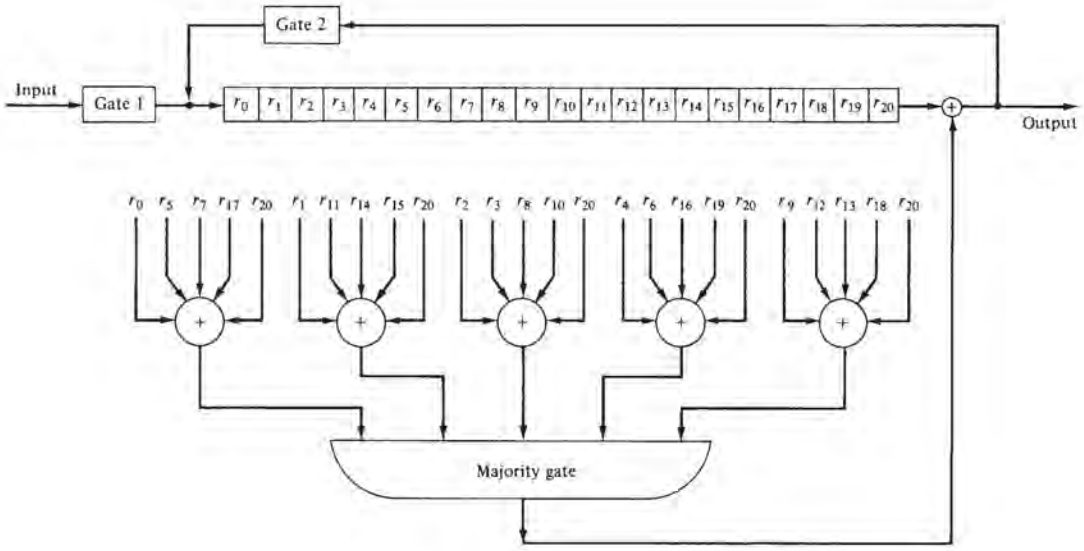


Figure 7.7 Type II majority-logic decoder for the (21, 11) difference-set code.

TABLE 7.2 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, 96, 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 $g(X) = 1 + X^2 + X^3 + X^4$.

7.4 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 to their minimum distance $d_{\min}$. Unfortunately, there are only several small classes of cyclic codes 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 $E = \{e_{i_1}, e_{i_2}, \ldots, e_{i_M}\}$ be a set of M error digits where $0 \leq i_1 < i_2 < \cdots < i_M < n$. The integer M is called the *size* of E.

Definition 7.2. A set of J parity-check sums $A_1, A_2, \ldots, A_J$ 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.

Ex.1020 / Page 228 of 305
Sandisk Technologies, Inc. et al

For example, the following four parity-check sums are orthogonal on the set $E = \{e_6, e_8\}$:

$$\begin{aligned} A_1 &= e_0 + e_2 + e_6 + e_8, \\ A_2 &= e_3 + e_4 + e_6 + e_8, \\ A_3 &= e_1 + e_6 + e_7 + e_8, \\ A_4 &= e_5 + e_6 + e_8. \end{aligned}$$

Following the same argument employed for one-step majority-logic decoding, the sum of error digits in E, $e_{i_1} + e_{i_2} + \cdots + e_{i_M}$, can be determined correctly from the check sums, $A_1, A_2, \ldots, A_J$, orthogonal on E provided that there are $\lfloor J/2 \rfloor$ or fewer errors in the error pattern $\mathbf{e}$. 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 which is used for error control purpose in a communication system. Let $\mathbf{e} = (e_0, e_1, \ldots, e_{n-1})$ denote the error vector that occurs during the transmission of a code vector $\mathbf{v}$ in C. Let $E_1^1, E_2^1, \ldots, E_i^1, \ldots$ 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 one-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, $S(E_1^1), S(E_2^1), \ldots, S(E_i^1), \ldots$ (possibly together with other check sums) are then used to estimate the sums of error digits in the second selected sets, $E_1^2, E_2^2, \ldots, E_i^2, \ldots$ with size 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, $S(E_1^2), S(E_2^2), \ldots, S(E_i^2), \ldots$, are determined, they (may be together with other check sums) are used to estimate the sums of error digits in the third selected sets, $E_1^3, E_2^3, \ldots, E_i^3, \ldots$, with size smaller than that of the second selected sets. The process of estimating check sums from known check sums is called *orthogonalization* [3]. 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 one less than the minimum distance of the code (i.e., $J = d_{\min} - 1$). Since 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.

In the following two examples are used to illustrate the notions of multiple-step majority-logic decoding.

Ex.1020 / Page 229 of 305
Sandisk Technologies, Inc. et al

Example 7.11

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

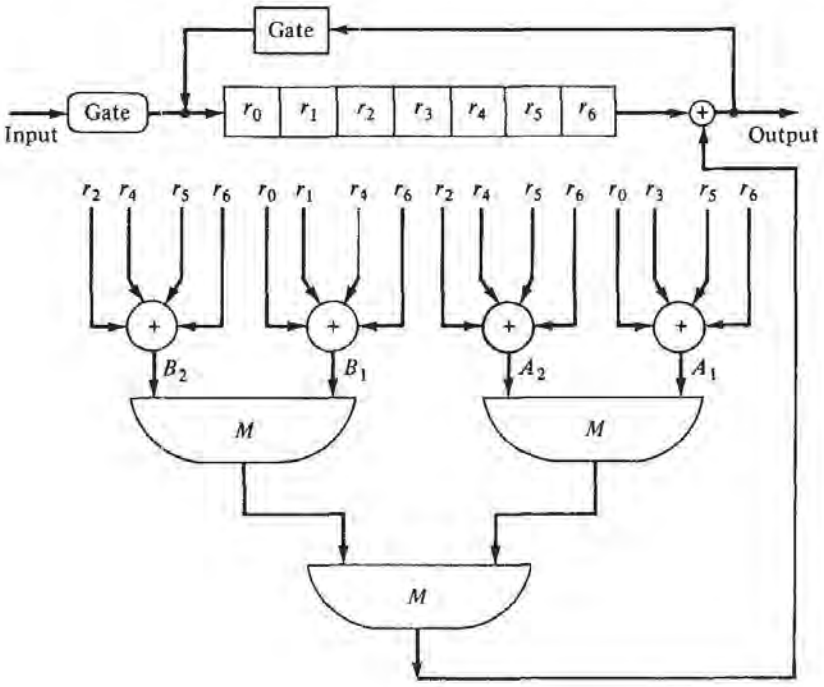
$$A_1 = \mathbf{r} \cdot \mathbf{h}_0 = e_0 + e_3 + e_5 + e_6$$
$$A_2 = \mathbf{r} \cdot \mathbf{h}_2 = e_2 + e_4 + e_5 + e_6$$

and the parity-check sums formed from $\mathbf{h}_0 + \mathbf{h}_1$ and $\mathbf{h}_2$ are

$$B_1 = \mathbf{r} \cdot (\mathbf{h}_0 + \mathbf{h}_1) = e_0 + e_1 + e_4 + e_6$$
$$B_2 = \mathbf{r} \cdot \mathbf{h}_2 = e_2 + e_4 + e_5 + e_6.$$

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 $S(E_2^1)$ would 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 $S(E_2^1)$ are orthogonal on e_6. Hence, e_6 can be estimated from $S(E_1^1)$ and $S(E_2^1)$. The value of e_6 will be estimated correctly provided that there are 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. Since 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 7.8.

Figure 7.8 Type II two-step majority-logic decoder for the (7, 4) Hamming code.



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:

$$A_1 = s_0, \qquad A_2 = s_2,$$
$$B_1 = s_0 + s_1, \qquad B_2 = s_2.$$

Based on these check sums, one may construct a type I majority-logic decoder for the (7, 4) Hamming code.

Example 7.12

Consider the triple-error-correcting (15, 5) BCH code whose generator polynomial is

$$g(X) = 1 + X + X^2 + X^4 + X^5 + X^8 + X^{10}.$$

The parity-check matrix (in systematic form) is

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \\ \mathbf{h}_4 \\ \mathbf{h}_5 \\ \mathbf{h}_6 \\ \mathbf{h}_7 \\ \mathbf{h}_8 \\ \mathbf{h}_9 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}.$$

Let

$$E_1^1 = \{e_{13}, e_{14}\}, \qquad E_2^1 = \{e_{12}, e_{14}\},$$
$$E_3^1 = \{e_{11}, e_{14}\}, \qquad E_4^1 = \{e_{10}, e_{14}\},$$
$$E_5^1 = \{e_5, e_{14}\}, \qquad E_6^1 = \{e_2, e_{14}\}$$

be six selected sets of error digits. For each of the sets above, 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 of $\mathbf{H}$, 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 $E_1^1 = \{e_{13}, e_{14}\}$:

$$A_{11} = \mathbf{r} \cdot \mathbf{h}_4 = e_4 + e_{10} + e_{13} + e_{14}$$
$$A_{12} = \mathbf{r} \cdot \mathbf{h}_7 = e_7 + e_{12} + e_{13} + e_{14}$$
$$A_{13} = \mathbf{r} \cdot \mathbf{h}_9 = e_9 + e_{11} + e_{13} + e_{14}$$
$$A_{14} = \mathbf{r} \cdot (\mathbf{h}_0 + \mathbf{h}_8) = e_0 + e_8 + e_{13} + e_{14}$$
$$A_{15} = \mathbf{r} \cdot (\mathbf{h}_1 + \mathbf{h}_5) = e_1 + e_5 + e_{13} + e_{14}$$
$$A_{16} = \mathbf{r} \cdot (\mathbf{h}_3 + \mathbf{h}_6) = e_3 + e_6 + e_{13} + e_{14}.$$

2. Check sums orthogonal on $E_2^1 = \{e_{12}, e_{14}\}$:

$$A_{21} = \mathbf{r} \cdot \mathbf{h}_0 = e_0 + e_{10} + e_{12} + e_{14}$$
$$A_{22} = \mathbf{r} \cdot \mathbf{h}_3 = e_3 + e_{11} + e_{12} + e_{14}$$
$$A_{23} = \mathbf{r} \cdot \mathbf{h}_7 = e_7 + e_{13} + e_{12} + e_{14}$$

Ex.1020 / Page 231 of 305
Sandisk Technologies, Inc. et al

$$A_{24} = \mathbf{r} \cdot (\mathbf{h}_1 + \mathbf{h}_2) = e_1 + e_2 + e_{12} + e_{14}$$
$$A_{25} = \mathbf{r} \cdot (\mathbf{h}_4 + \mathbf{h}_8) = e_4 + e_8 + e_{12} + e_{14}$$
$$A_{26} = \mathbf{r} \cdot (\mathbf{h}_6 + \mathbf{h}_9) = e_6 + e_9 + e_{12} + e_{14}.$$

3. Check sums orthogonal on $E_3^1 = \{e_{11}, e_{14}\}$:

$$A_{31} = \mathbf{r} \cdot \mathbf{h}_3 = e_3 + e_{12} + e_{11} + e_{14}$$
$$A_{32} = \mathbf{r} \cdot \mathbf{h}_9 = e_9 + e_{13} + e_{11} + e_{14}$$
$$A_{33} = \mathbf{r} \cdot (\mathbf{h}_0 + \mathbf{h}_5) = e_0 + e_5 + e_{11} + e_{14}$$
$$A_{34} = \mathbf{r} \cdot (\mathbf{h}_1 + \mathbf{h}_8) = e_1 + e_8 + e_{11} + e_{14}$$
$$A_{35} = \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_4) = e_2 + e_4 + e_{11} + e_{14}$$
$$A_{36} = \mathbf{r} \cdot (\mathbf{h}_6 + \mathbf{h}_7) = e_6 + e_7 + e_{11} + e_{14}.$$

4. Check sums orthogonal on $E_4^1 = \{e_{10}, e_{14}\}$:

$$A_{41} = \mathbf{r} \cdot \mathbf{h}_0 = e_0 + e_{12} + e_{10} + e_{14}$$
$$A_{42} = \mathbf{r} \cdot \mathbf{h}_4 = e_4 + e_{13} + e_{10} + e_{14}$$
$$A_{43} = \mathbf{r} \cdot (\mathbf{h}_1 + \mathbf{h}_6) = e_1 + e_6 + e_{10} + e_{14}$$
$$A_{44} = \mathbf{r} \cdot (\mathbf{h}_3 + \mathbf{h}_5) = e_3 + e_5 + e_{10} + e_{14}$$
$$A_{45} = \mathbf{r} \cdot (\mathbf{h}_7 + \mathbf{h}_8) = e_7 + e_8 + e_{10} + e_{14}$$
$$A_{46} = \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_9) = e_2 + e_9 + e_{10} + e_{14}.$$

5. Check sums orthogonal on $E_5^1 = \{e_5, e_{14}\}$:

$$A_{51} = \mathbf{r} \cdot (\mathbf{h}_0 + \mathbf{h}_5) = e_0 + e_{11} + e_5 + e_{14}$$
$$A_{52} = \mathbf{r} \cdot (\mathbf{h}_1 + \mathbf{h}_5) = e_1 + e_{13} + e_5 + e_{14}$$
$$A_{53} = \mathbf{r} \cdot (\mathbf{h}_3 + \mathbf{h}_5) = e_3 + e_{10} + e_5 + e_{14}$$
$$A_{54} = \mathbf{r} \cdot (\mathbf{h}_4 + \mathbf{h}_5 + \mathbf{h}_6) = e_4 + e_6 + e_5 + e_{14}$$
$$A_{55} = \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_5 + \mathbf{h}_7) = e_2 + e_7 + e_5 + e_{14}$$
$$A_{56} = \mathbf{r} \cdot (\mathbf{h}_5 + \mathbf{h}_8 + \mathbf{h}_9) = e_8 + e_9 + e_5 + e_{14}.$$

6. Check sums orthogonal on $E_6^1 = \{e_2, e_{14}\}$:

$$A_{61} = \mathbf{r} \cdot (\mathbf{h}_1 + \mathbf{h}_2) = e_1 + e_{12} + e_2 + e_{14}$$
$$A_{62} = \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_4) = e_4 + e_{11} + e_2 + e_{14}$$
$$A_{63} = \mathbf{r} \cdot (\mathbf{h}_0 + \mathbf{h}_2 + \mathbf{h}_6) = e_0 + e_6 + e_2 + e_{14}$$
$$A_{64} = \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_3 + \mathbf{h}_8) = e_3 + e_8 + e_2 + e_{14}$$
$$A_{65} = \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_5 + \mathbf{h}_7) = e_5 + e_7 + e_2 + e_{14}$$
$$A_{66} = \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_9) = e_9 + e_{10} + e_2 + e_{14}.$$

From the orthogonal check sums above, 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. Since $J = 6$, it is capable of correcting three or fewer errors with two-step majority-logic decoding.

Ex.1020 / Page 232 of 305
Sandisk Technologies, Inc. et al

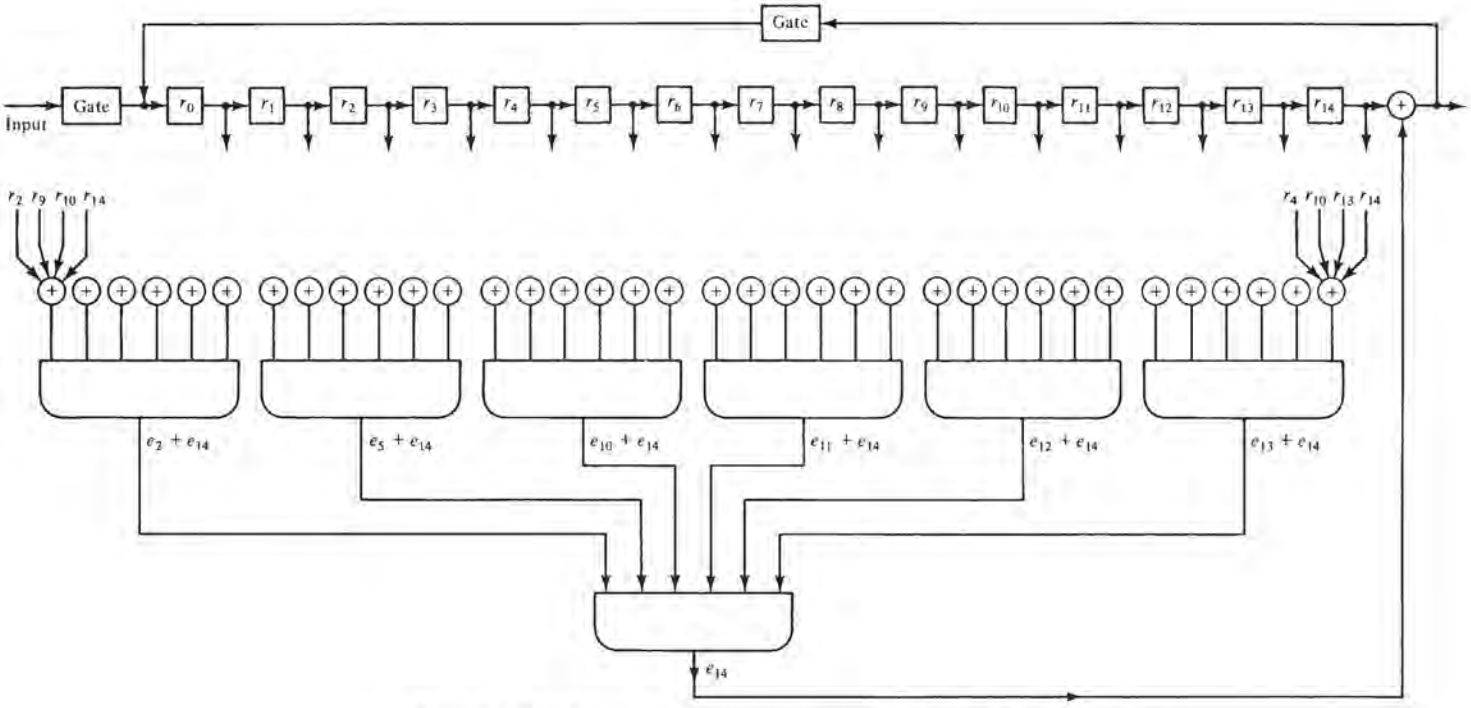


Figure 7.9 Type II two-step majority-logic decoder for the (15, 5) BCH code.

It is known that the code has minimum distance exactly 7. Hence, it is two-step completely orthogonalizable.

The type II decoder for the (15, 5) BCH code is shown in Figure 7.9, where seven six-input majority-logic gates (connected in a tree form) are used. If we examine the parity-check matrix carefully and are willing to trade decoding speed with decoding complexity, we will be able to reduce the number of majority-logic gates. Suppose that we choose the following sets:

$$Q_1^1 = \{e_{12}, e_{13}\}, \qquad Q_2^1 = \{e_{10}, e_{13}\}, \qquad Q_3^1 = \{e_{11}, e_{12}\}.$$

For each of these sets, we are able to form six parity-check sums orthogonal on it as follows:

1. Check sums orthogonal on $Q_1^1 = \{e_{12}, e_{13}\}$:

$$\begin{aligned}
B_{11} &= \mathbf{r} \cdot (\mathbf{h}_3 + \mathbf{h}_9) = e_3 + e_9 + e_{12} + e_{13} \\
B_{12} &= \mathbf{r} \cdot \mathbf{h}_6 = e_6 + e_{11} + e_{12} + e_{13} \\
B_{13} &= \mathbf{r} \cdot \mathbf{h}_8 = e_8 + e_{10} + e_{12} + e_{13} \\
B_{14} &= \mathbf{r} \cdot \mathbf{h}_7 = e_7 + e_{14} + e_{12} + e_{13} \\
B_{15} &= \mathbf{r} \cdot (\mathbf{h}_0 + \mathbf{h}_4) = e_0 + e_4 + e_{12} + e_{13} \\
B_{16} &= \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_5) = e_2 + e_5 + e_{12} + e_{13}.
\end{aligned}$$

2. Check sums orthogonal on $Q_2^1 = \{e_{10}, e_{13}\}$:

$$\begin{aligned}
B_{21} &= \mathbf{r} \cdot \mathbf{h}_2 = e_2 + e_{11} + e_{10} + e_{13} \\
B_{22} &= \mathbf{r} \cdot \mathbf{h}_8 = e_8 + e_{12} + e_{10} + e_{13} \\
B_{23} &= \mathbf{r} \cdot \mathbf{h}_4 = e_4 + e_{14} + e_{10} + e_{13} \\
B_{24} &= \mathbf{r} \cdot (\mathbf{h}_0 + \mathbf{h}_7) = e_0 + e_7 + e_{10} + e_{13} \\
B_{25} &= \mathbf{r} \cdot (\mathbf{h}_1 + \mathbf{h}_3) = e_1 + e_3 + e_{10} + e_{13} \\
B_{26} &= \mathbf{r} \cdot (\mathbf{h}_5 + \mathbf{h}_6) = e_5 + e_6 + e_{10} + e_{13}.
\end{aligned}$$

3. Check sums orthogonal on $Q_3^1 = \{e_{11}, e_{12}\}$:

$$\begin{aligned}
B_{31} &= \mathbf{r} \cdot (\mathbf{h}_2 + \mathbf{h}_8) = e_2 + e_8 + e_{11} + e_{12} \\
B_{32} &= \mathbf{r} \cdot \mathbf{h}_5 = e_5 + e_{10} + e_{11} + e_{12} \\
B_{33} &= \mathbf{r} \cdot (\mathbf{h}_7 + \mathbf{h}_9) = e_7 + e_9 + e_{11} + e_{12} \\
B_{34} &= \mathbf{r} \cdot \mathbf{h}_6 = e_6 + e_{13} + e_{11} + e_{12} \\
B_{35} &= \mathbf{r} \cdot \mathbf{h}_3 = e_3 + e_{14} + e_{11} + e_{12} \\
B_{36} &= \mathbf{r} \cdot (\mathbf{h}_1 + \mathbf{h}_4) = e_1 + e_4 + e_{11} + e_{12}.
\end{aligned}$$

Based on the orthogonal sums above, we can estimate the sums, $S(Q_1^1) = e_{12} + e_{13}$, $S(Q_2^1) = e_{10} + e_{13}$, and $S(Q_3^1) = e_{11} + e_{12}$. Next, we combine the sums $S(Q_1^1)$, $S(Q_2^1)$, and $S(Q_3^1)$ with *other check sums* to form six check sums orthogonal on $Q_1^2 = \{e_{14}\}$. By examining the rows of $\mathbf{H}$ carefully, we obtain the following check sums orthogonal on e_{14}:

$$\begin{aligned}
D_1 &= S(Q_1^1) + \mathbf{r} \cdot \mathbf{h}_7 = e_7 + e_{14}, \\
D_2 &= S(Q_2^1) + \mathbf{r} \cdot \mathbf{h}_4 = e_4 + e_{14}, \\
D_3 &= S(Q_3^1) + \mathbf{r} \cdot \mathbf{h}_3 = e_3 + e_{14},
\end{aligned}$$

Ex.1020 / Page 234 of 305
Sandisk Technologies, Inc. et al

$$D_4 = S(Q_1^1) + S(Q_2^1) + \mathbf{r} \cdot \mathbf{h}_0 = e_0 + e_{14},$$
$$D_5 = S(Q_1^1) + S(Q_3^1) + \mathbf{r} \cdot \mathbf{h}_9 = e_9 + e_{14},$$
$$D_6 = S(Q_2^1) + S(Q_3^1) + \mathbf{r} \cdot \mathbf{h}_1 = e_1 + e_{14}.$$

From these check sums, we can determine e_{14}. At the first level of decoding, we need three six-input majority-logic gates to determine the sums $S(Q_1^1)$, $S(Q_2^1)$, and $S(Q_3^1)$. Once these sums are formed, we can form the sums D_1 to D_6 by using six modulo-2 adders. Then it comes to the last level of decoding, where another six-input majority-logic gate is needed to determine e_{14} from the sums D_1 to D_6. Forming orthogonal parity-check sums as described above, we obtain a simpler type II majority-logic decoder for the (15, 5) BCH code as shown in Figure 7.10. Comparing the two decoding circuits for the (15, 5) BCH code, we see that the first circuit (Figure 7.9) requires 7 six-input majority-logic gates and 26 modulo-2 adders to form the orthogonal sums and to make the decoding decision, whereas the second circuit (Figure 7.10) requires only 4 six-input majority-logic gates and 22 modulo-2 adders. Therefore, there is a reduction in decoding complexity by using the second method of forming the orthogonal check sums. However, this reduction in complexity is achieved at the expense of extra decoding delay caused by the layer of modulo-2 adders between the two levels of majority-logic gates. Construction of a type I majority-logic decoder for the (15, 5) BCH code is left as an exercise (see Problem 7.12).

A general type II L-step majority-logic decoder is shown in Figure 7.11. The error correction procedure is described 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, 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 continues until the last level is reached; there is only one gate at the last level. The J inputs to this 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 the highest-order received digit r_{n-1} was. The decoder repeats steps 2 and 3.

Step 5. The received vector is decoded digit by digit in the manner described above until a total of n shifts.

A general type I decoder for a L-step majority-logic decoder code is shown in Figure 7.12. Its decoding operation is identical to that of the type I decoder for a

Ex.1020 / Page 235 of 305
Sandisk Technologies, Inc. et al

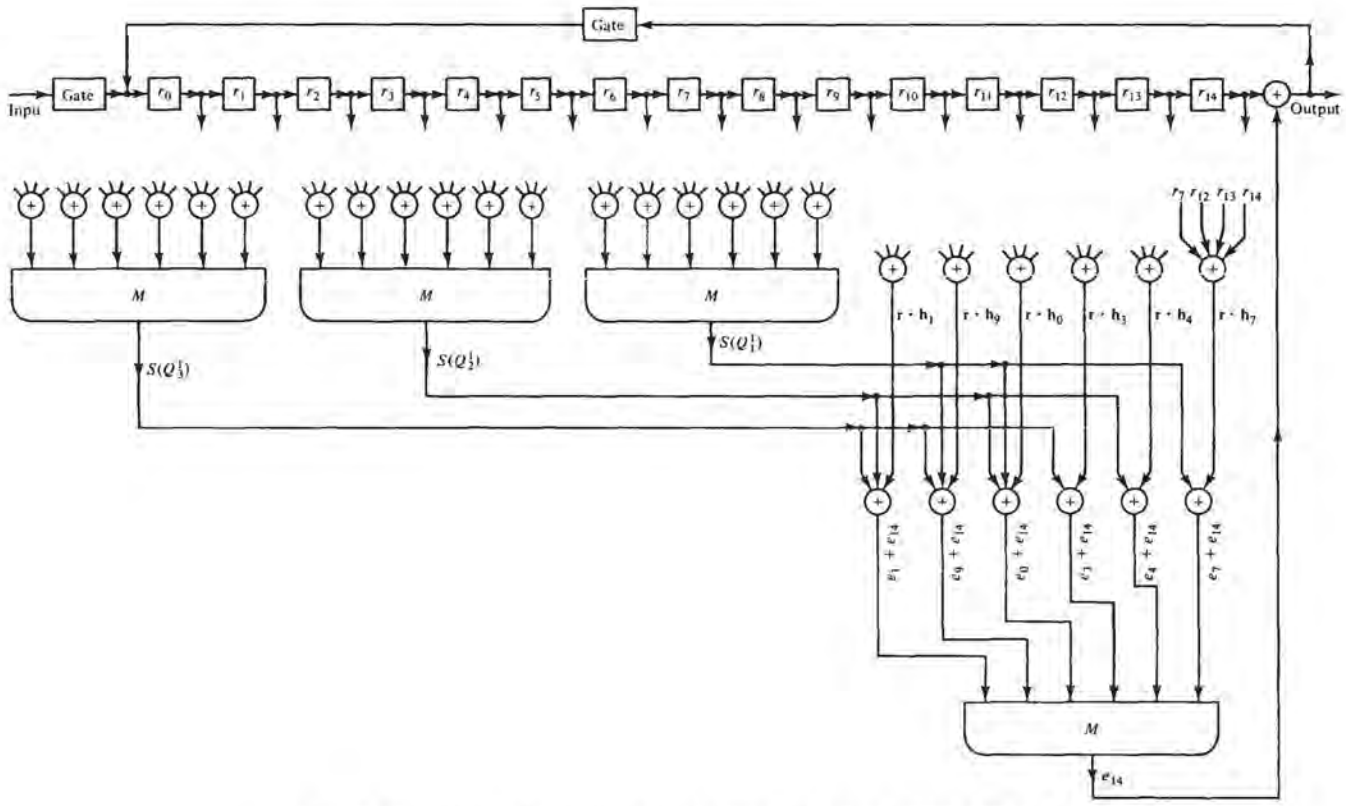


Figure 7.10 Another type II one-step majority logic decoder for the (15, 5) BCH code.

Ex.1020 / Page 236 of 305
Sandisk Technologies, Inc. et al

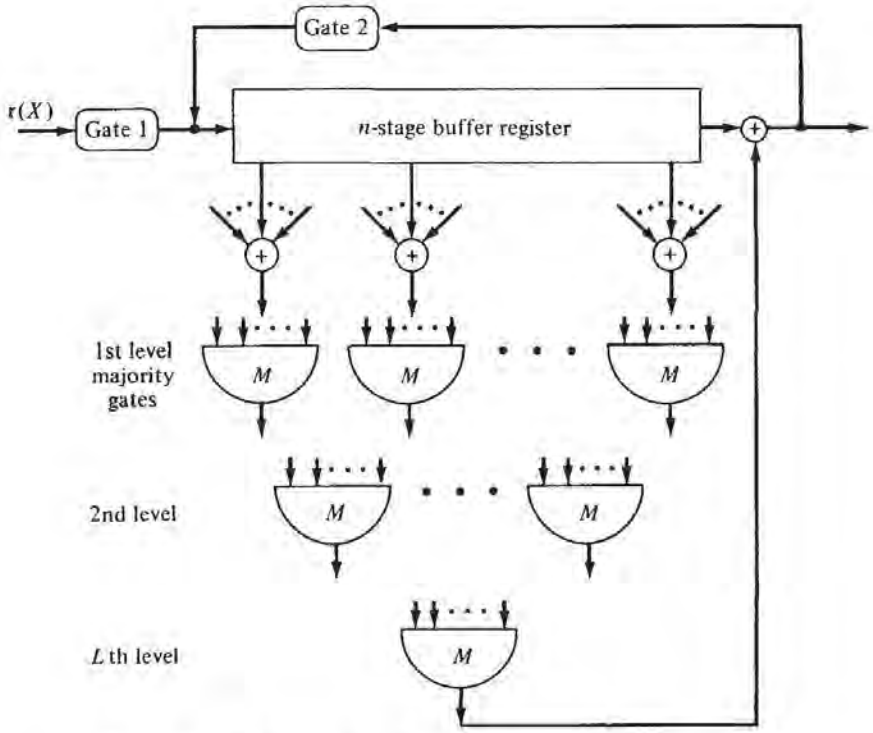


Figure 7.11 General type II L-step majority-logic decoder.

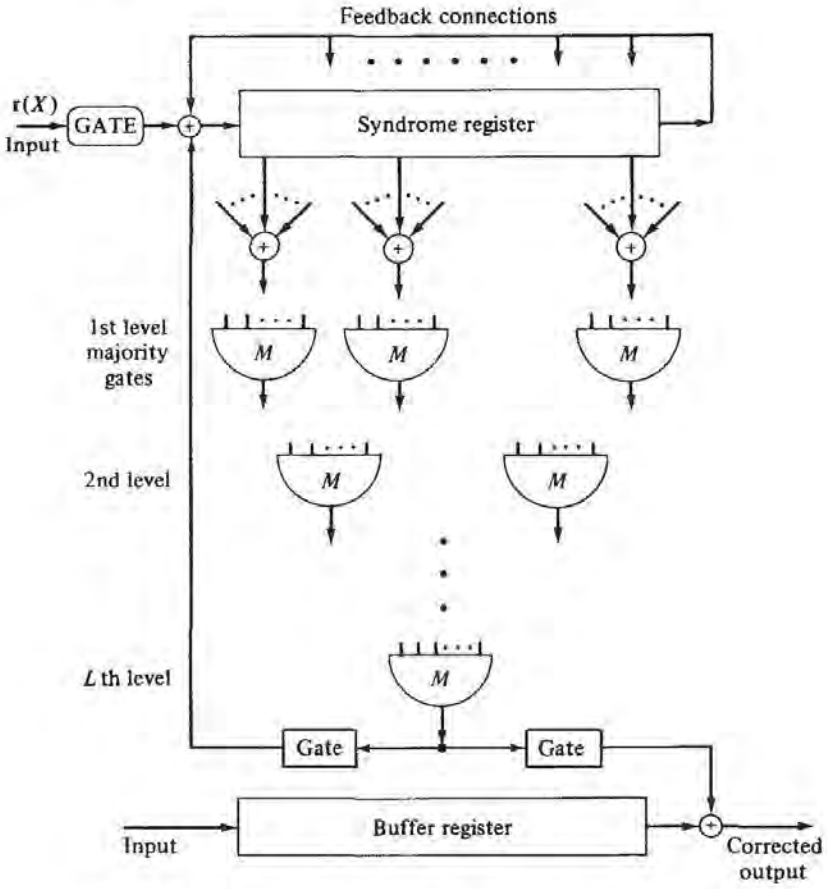


Figure 7.12 General type I L-step majority-logic decoder.

Ex.1020 / Page 237 of 305
Sandisk Technologies, Inc. et al

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 ith level, no more than $(J)^{L-i}$ gates are required. Thus, the total number of majority-logic gates needed is upper bounded by $1 + J + J^2 + \cdots + J^{L-1}$. In fact, Massey [3] 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 requires a total of $1 + J + J^2 + \cdots + J^{L-1}$ 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.

It has been shown [3] that the $(2^m - 1, 2^m - m - 1)$ Hamming code is completely orthogonalizable in $m - 1$ steps. Thus, the Hamming codes can be decoded by the majority-logic decoding described above. However, the error-trapping decoding for Hamming codes described in Section 4.6 can be much more simply implemented than the majority-logic decoding. The BCH codes which are known to be completely orthogonalizable are: (1) the subclass of $(2^{m-2} - 1)$-error correcting $(2^m - 1, m + 1)$ codes with $m \geq 3$, which is two-step orthogonalizable [3]; (2) the double-error-correcting (15, 7), code which is one-step orthogonalizable [3]; (3) the triple-error-correcting (31, 16) code, which can be completely orthogonalized in two steps [14–16]; and (4) the triple-error-correcting (63, 45) code and the seven-error-correcting (63, 24) code, both are two-step orthogonalizable [17]. Besides the codes above, 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 subjects of Chapter 8.

PROBLEMS

7.1. 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.

7.2. $P = \{0, 2, 3\}$ is a perfect simple difference set. Construct a difference-set code based on this set.

(a) What is the length n of this code?

(b) Determine its generator polynomial.

(c) Find all the polynomials orthogonal on the highest-order digit position X^{n-1}.

(d) Construct a type I majority-logic decoder for this code.

7.3. In Example 7.1 we showed 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.

Ex.1020 / Page 238 of 305
Sandisk Technologies, Inc. et al

7.4. Consider an (11, 6) linear code whose parity-check matrix is

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

(This code is not cyclic.)

(a) Show that the minimum distance of this code is exactly 4.

(b) 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.

(c) Construct all possible parity-check sums orthogonal on each message error digit e_i for $i = 5, 6, 7, 8, 9, 10$.

(d) Is this code completely orthogonalizable in one step?

7.5. Let $m = 6$. Express the integer 43 in radix 2 form. Find all the nonzero proper descendants of 43.

7.6. Let α be a primitive element of GF(2^4) given by Table 2.8. Apply the affine permutation $Z = \alpha^3 Y + \alpha^{11}$ to the following vector of 16 components:

	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}
$\mathbf{u} =$	(1	1	1	0	0	1	0	1	0	1	1	0	0	0	0	1)

What is the resultant vector?

7.7. 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}).

7.8. 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}.

7.9. Show that the all-one vector is not a code vector in a maximum-length code.

7.10. Let $\mathbf{v}(X) = v_0 + v_1 X + \cdots + v_{2^m-2} X^{2^m-2}$ 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 ith and jth cyclic shifts of $\mathbf{v}(X)$, respectively, with $0 \leq i < j < 2^m - 2$. Show that $\mathbf{v}^{(i)}(X) \neq \mathbf{v}^{(j)}(X)$.]

7.11. Arrange the 2^m code vectors of a maximum-length code as rows of a $2^m \times (2^m - 1)$ array.

(a) Show that each column of this array has 2^{m-1} ones and 2^{m-1} zeros.

(b) Show that the weight of each nonzero code vector is exactly 2^{m-1}.

7.12. In Example 7.12, we showed 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.

7.13. Show that the extended cyclic Hamming code is invariant under the affine permutations.

7.14. Show that the extended primitive BCH code is invariant under the affine permutations.

7.15. Let $P = \{l_0, l_1, l_2, \ldots, l_{2^s}\}$ be a perfect simple difference set of order 2^s such that

$$0 \leq l_0 < l_1 < l_2 < \cdots < l_{2^s} \leq 2^s(2^s + 1).$$

Ex.1020 / Page 239 of 305
Sandisk Technologies, Inc. et al

Construct a vector of $n = 2^{2s} + 2^s + 1$ components,

$$\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$$

whose nonzero components are $v_{l_0}, v_{l_1}, \ldots, v_{l_{2^s}}$, that is,

$$v_{l_0} = v_{l_1} = \cdots = v_{l_{2^s}} = 1.$$

Consider the following $n \times 2n$ matrix:

$$\mathbf{G} = [\mathbf{Q} \quad \mathbf{I}_n],$$

where (1) I_n is an $n \times n$ identity matrix, and (2) 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 code (not cyclic) whose parity-check matrix is

$$\mathbf{H} = [\mathbf{I}_n \quad \mathbf{Q}^T].$$

(a) Show that $J = 2^s + 1$ parity-check sums orthogonal on any message error digit can be formed.

(b) Show that the minimum distance of this code is $d = J + 1 = 2^s + 2$. (This code is referred to as a half-rate *quasi-cyclic code* [18].)

7.16. 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 $J + 1$.

7.17. Let $\mathbf{H}$ be the parity-check matrix of an (n, k) linear code C. Show that the matrix

$$\mathbf{H}_e = \left[\begin{array}{c:ccc} 0 & & & \\ 0 & & & \\ \vdots & & H & \\ 0 & & & \\ \hdashline 1 & 1 & 1 \cdots & 1 \end{array}\right]$$

is a parity-check matrix of the $(n + 1, k)$ linear code C_e obtained by adding an overall parity-check digit to each code vector in C.

7.18. Determine the weight enumerator for the extended Hamming code of length 2^m.

7.19. Determine the weight enumerator for the dual of the extended Hamming code.

7.20. Show that the probability of an undetected error $P_u(E)$ of the extended Hamming code of length 2^m satisfies the upper bound $2^{-(m+1)}$.

REFERENCES

1. I. S. Reed, "A Class of Multiple-Error-Correcting Codes and the Decoding Scheme," *IRE Trans.*, IT-4, pp. 38–49, September 1954.
2. D. E. Muller, "Applications of Boolean Algebra to Switching Circuit Design and to Error Detection," *IRE Trans.*, EC-3, pp. 6–12, September 1954.
3. J. L. Massey, *Threshold Decoding*, MIT Press, Cambridge, Mass. 1963.
4. T. Kasami, L. Lin, and W. W. Peterson, "Some Results on Cyclic Codes Which Are Invariant under the Affine Group and Their Applications," *Inf. Control*, 2(5 and 6), pp. 475–496, November 1968.
5. W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2nd ed. MIT Press, Cambridge, Mass., 1972.

Ex.1020 / Page 240 of 305
Sandisk Technologies, Inc. et al

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, Mass., 1958.
8. N. Zierler, "On a Variation of the First Order Reed-Muller Codes," Lincoln Laboratory Report, pp. 38–80, Lincoln Labs., MIT, Cambridge, Mass., 1958.
9. J. Singer, "A Theorem in Finite Projective Geometry and Some Applications to Number Theory," *AMS Trans.*, 43, pp. 377–385, 1938.
10. T. A. Evans and H. B. Mann, "On Simple Difference Sets," *Sankhya*, 11, pp. 464–481, 1955.
11. L. D. Rudolph, "Geometric Configuration and Majority Logic Decodable Codes," M.E.E. thesis, University of Oklahoma, Norman, Okla., 1964.
12. E. J. Weldon, Jr., "Difference-Set Cyclic Codes," *Bell Syst. Tech. J.*, 45, pp. 1045–1055, 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, pp. 1046–1070, September 1966.
14. L. D. Rudolph, "A Class of Majority Logic Decodable Codes," *IEEE Trans. Inf. Theory*, IT-13, pp. 305–307, April 1967.
15. E. J. Weldon, Jr., "New Generations of the Reed-Muller Codes-Part II: Non-Primitive Codes," *IEEE Trans. Inf. Theory*, IT-14, pp. 199–205, March 1968.
16. J. M. Coethals and P. Delsarte, "On a Class of Majority-Logic Decodable Codes," *IEEE Trans. Inf. Theory*, IT-14(2), pp. 182–189, March 1968.
17. S. Lin, "Multifold Euclidean Geometry Codes," *IEEE Trans. Inf. Theory*, IT-19(4), pp. 537–548, July 1973.
18. E. J. Weldon, Jr., "Quasi-Cyclic Codes," *IEEE Trans. Inf. Theory*, IT-12, pp. 183–195, April 1967.

Ex.1020 / Page 241 of 305
Sandisk Technologies, Inc. et al

8

Finite Geometry Codes

The construction and rules of orthogonalization for most of the multiple-step majority-logic decodable cyclic codes are based on the structure of finite geometries, namely *Euclidean* and *projective* geometries. In this chapter a brief discussion of these geometries is given. We introduce some basic concepts and state some useful properties without proofs. For details of finite geometries, the reader is referred to References 1 and 2.

Codes constructed based on the structure of finite geometries are called *finite geometry codes*. Finite geometry codes were first investigated by Rudolph [3]. Rudolph's work was later extended and generalized by many coding investigators. In this chapter several classes of finite geometry codes are presented.

8.1 EUCLIDEAN GEOMETRY

Consider all the m-tuples $(a_0, a_1, \ldots, a_{m-1})$, 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,

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

$$\beta(a_0, a_1, \ldots, a_{m-1}) = (\beta a_0, \beta a_1, \ldots, \beta a_{m-1}),$$

where addition $a_i + b_i$ and multiplication βa_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 EG(m, 2^s). Each m-tuple $\mathbf{a} =$

Ex.1020 / Page 242 of 305
Sandisk Technologies, Inc. et al

$(a_0, a_1, \ldots, a_{m-1})$ is called a *point* in EG$(m, 2^s)$. The all-zero m-tuple, $\mathbf{0} = (0, 0, \ldots, 0)$, 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, $\{\beta\mathbf{a}: \beta \in \text{GF}(2^s)\}$, constitute a *line* (or 1-*flat*) in EG$(m, 2^s)$. For convenience, we use the notation $\{\beta\mathbf{a}\}$ to represent this line. Since 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 $\beta \in \text{GF}(2^s)$, constitute 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, $\mathbf{a}_0 + (\beta'' - \beta')\mathbf{a} = \mathbf{0}$. 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 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. 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 EG$(m, 2^s)$, for every line passing through the origin, there are $2^{(m-1)s} - 1$ lines parallel to it.

Let $\mathbf{a}_1$ and $\mathbf{a}_2$ be two linearly independent points in 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 $\beta'\mathbf{a}_1 - \beta''\mathbf{a}_2 = \mathbf{0}$ 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 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 EG$(m, 2^s)$, there are

$$\frac{2^{ms} - 1}{2^s - 1} \tag{8.1}$$

lines in EG$(m, 2^s)$ that intersect at $\mathbf{a}_0$. (This is an important property that will be used to form orthogonal parity-check sums for the codes presented in the next section.) The total number of lines in EG$(m, 2^s)$ is

$$2^{(m-1)s}(2^{ms} - 1).$$

Example 8.1

Let $m = 3$ and $s = 1$. Consider the Euclidean geometry EG(3, 2) over GF(2). There are eight points and 28 lines. Each point $\mathbf{a}_i$ is 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 8.1.

Ex.1020 / Page 243 of 305
Sandisk Technologies, Inc. et al

TABLE 8.1

(a) Points in EG(3, 2)

$\mathbf{a}_0 = (0\ 0\ 0)$, $\mathbf{a}_1 = (0\ 0\ 1)$, $\mathbf{a}_2 = (0\ 1\ 0)$, $\mathbf{a}_3 = (0\ 1\ 1)$,
$\mathbf{a}_4 = (1\ 0\ 0)$, $\mathbf{a}_5 = (1\ 0\ 1)$, $\mathbf{a}_6 = (1\ 1\ 0)$, $\mathbf{a}_7 = (1\ 1\ 1)$.

(b) Lines in EG(3, 2)

$\{\mathbf{a}_0, \mathbf{a}_1\}$	$\{\mathbf{a}_1, \mathbf{a}_2\}$	$\{\mathbf{a}_2, \mathbf{a}_4\}$	$\{\mathbf{a}_3, \mathbf{a}_7\}$
$\{\mathbf{a}_0, \mathbf{a}_2\}$	$\{\mathbf{a}_1, \mathbf{a}_3\}$	$\{\mathbf{a}_2, \mathbf{a}_5\}$	$\{\mathbf{a}_4, \mathbf{a}_5\}$
$\{\mathbf{a}_0, \mathbf{a}_3\}$	$\{\mathbf{a}_1, \mathbf{a}_4\}$	$\{\mathbf{a}_2, \mathbf{a}_6\}$	$\{\mathbf{a}_4, \mathbf{a}_6\}$
$\{\mathbf{a}_0, \mathbf{a}_4\}$	$\{\mathbf{a}_1, \mathbf{a}_5\}$	$\{\mathbf{a}_2, \mathbf{a}_7\}$	$\{\mathbf{a}_4, \mathbf{a}_7\}$
$\{\mathbf{a}_0, \mathbf{a}_5\}$	$\{\mathbf{a}_1, \mathbf{a}_6\}$	$\{\mathbf{a}_3, \mathbf{a}_4\}$	$\{\mathbf{a}_5, \mathbf{a}_6\}$
$\{\mathbf{a}_0, \mathbf{a}_6\}$	$\{\mathbf{a}_1, \mathbf{a}_7\}$	$\{\mathbf{a}_3, \mathbf{a}_5\}$	$\{\mathbf{a}_5, \mathbf{a}_7\}$
$\{\mathbf{a}_0, \mathbf{a}_7\}$	$\{\mathbf{a}_2, \mathbf{a}_3\}$	$\{\mathbf{a}_3, \mathbf{a}_6\}$	$\{\mathbf{a}_6, \mathbf{a}_7\}$

The 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. 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 $\{\mathbf{a}_2, \mathbf{a}_7\}$.

Now, we extend the concepts of lines to planes in EG(m, 2^s). Let $\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_\mu$ be $\mu + 1$ linearly independent points in EG(m, 2^s), where $\mu < m$. The $2^{\mu s}$ points of the form

$$\mathbf{a}_0 + \beta_1\mathbf{a}_1 + \beta_2\mathbf{a}_2 + \cdots + \beta_\mu\mathbf{a}_\mu,$$

with $\beta_i \in \mathrm{GF}(2^s)$ for $1 \leq i \leq \mu$, constitute a *μ-flat* (or a μ-dimensional *hyperplane*) in EG(m, 2^s) which 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 $2^{\mu s}$ 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-flat in EG(m, 2^s) parallel to it.

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

$$\frac{2^{(m-\mu)s} - 1}{2^s - 1}. \tag{8.2}$$

Any point outside the μ-flat F is 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

$$2^{(m-\mu)s} \prod_{i=1}^{\mu} \frac{2^{(m-i+1)s} - 1}{2^{(\mu-i+1)s} - 1}.$$

Ex.1020 / Page 244 of 305
Sandisk Technologies, Inc. et al

Example 8.2

Consider the geometry EG(3, 2) over GF(2) given in Example 8.1. There are fourteen 2-flats, which are given in Table 8.2. The 2-flats that intersect on the line $\{\mathbf{a}_1, \mathbf{a}_3\}$ are $\{\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$, $\{\mathbf{a}_1, \mathbf{a}_3, \mathbf{a}_5, \mathbf{a}_7\}$, and $\{\mathbf{a}_1, \mathbf{a}_3, \mathbf{a}_4, \mathbf{a}_6\}$. The 2-flats, $\{\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$ and $\{\mathbf{a}_4, \mathbf{a}_5, \mathbf{a}_6, \mathbf{a}_7\}$ are parallel.

TABLE 8.2 2-FLATS IN EG(3, 2)

$\{\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$	$\{\mathbf{a}_4, \mathbf{a}_5, \mathbf{a}_6, \mathbf{a}_7\}$	$\{\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_4, \mathbf{a}_5\}$	$\{\mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_6, \mathbf{a}_7\}$
$\{\mathbf{a}_0, \mathbf{a}_2, \mathbf{a}_4, \mathbf{a}_6\}$	$\{\mathbf{a}_1, \mathbf{a}_3, \mathbf{a}_5, \mathbf{a}_7\}$	$\{\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_6, \mathbf{a}_7\}$	$\{\mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4, \mathbf{a}_5\}$
$\{\mathbf{a}_0, \mathbf{a}_2, \mathbf{a}_5, \mathbf{a}_7\}$	$\{\mathbf{a}_1, \mathbf{a}_3, \mathbf{a}_4, \mathbf{a}_6\}$	$\{\mathbf{a}_0, \mathbf{a}_4, \mathbf{a}_3, \mathbf{a}_7\}$	$\{\mathbf{a}_1, \mathbf{a}_5, \mathbf{a}_3, \mathbf{a}_6\}$
$\{\mathbf{a}_0, \mathbf{a}_3, \mathbf{a}_5, \mathbf{a}_6\}$	$\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_4, \mathbf{a}_7\}$		

Next, we show that the elements in the Galois field GE(2^{ms}) actually forms 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: $\alpha^\infty = 0$, $\alpha^0 = 1$, $\alpha^1, \alpha^2, \ldots, \alpha^{2^{ms}-2}$. 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

$$\alpha^i = a_{i0} + a_{i1}\alpha + a_{i2}\alpha^2 + \cdots + a_{i,m-1}\alpha^{m-1},$$

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 $(a_{i0}, a_{i1}, \ldots, a_{i,m-1})$ 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:

$$\alpha^{l_0} + \beta_1\alpha^{l_1} + \cdots + \beta_\mu\alpha^{l_\mu},$$

where $\alpha^{l_0}, \alpha^{l_1}, \ldots, \alpha^{l_\mu}$ are $\mu + 1$ linearly independent elements in GF(2^{ms}) and $\beta_i \in$ GF(2^s).

Example 8.3

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

$$0, 1, \beta, \beta^2$$

form a field of four elements, GF(2^2). Therefore, GF(2^2) is a subfield of GF(2^4). Table 8.3 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

$$\alpha^{14} + 0\cdot\alpha = \alpha^{14}, \qquad \alpha^{14} + 1\cdot\alpha = \alpha^7,$$
$$\alpha^{14} + \beta\cdot\alpha = \alpha^8, \qquad \alpha^{14} + \beta^2\cdot\alpha = \alpha^{10},$$

Ex.1020 / Page 245 of 305
Sandisk Technologies, Inc. et al

TABLE 8.3 ELEMENTS IN GF(2^4)*

	2-Tuples over GF(2^2)
$0 = 0$	$(0, 0)$
$1 = 1$	$(1, 0)$
$\alpha = \alpha$	$(0, 1)$
$\alpha^2 = \beta + \alpha$	$(\beta, 1)$
$\alpha^3 = \beta + \beta^2\alpha$	(β, β^2)
$\alpha^4 = 1 + \alpha$	$(1, 1)$
$\alpha^5 = \beta$	$(\beta, 0)$
$\alpha^6 = \beta\alpha$	$(0, \beta)$
$\alpha^7 = \beta^2 + \beta\alpha$	(β^2, β)
$\alpha^8 = \beta^2 + \alpha$	$(\beta^2, 1)$
$\alpha^9 = \beta + \beta\alpha$	(β, β)
$\alpha^{10} = \beta^2$	$(\beta^2, 0)$
$\alpha^{11} = \beta^2\alpha$	$(0, \beta^2)$
$\alpha^{12} = 1 + \beta^2\alpha$	$(1, \beta^2)$
$\alpha^{13} = 1 + \beta\alpha$	$(1, \beta)$
$\alpha^{14} = \beta^2 + \beta^2\alpha$	(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 line passing through the point α^{14}. The other four lines in EG(2, 2^2) passing through α^{14} are

$$\{\alpha^{14}, \alpha^{13}, \alpha, \alpha^5\}, \quad \{\alpha^{14}, \alpha^0, \alpha^6, \alpha^2\},$$
$$\{\alpha^{14}, \alpha^9, \alpha^4, 0\}, \quad \{\alpha^{14}, \alpha^{12}, \alpha^{11}, \alpha^3\}.$$

8.2 MAJORITY-LOGIC DECODABLE CYCLIC CODES BASED ON EUCLIDEAN GEOMETRY

Let

$$\mathbf{v} = (v_0, v_1, \ldots, v_{2^{ms}-2})$$

be a $(2^{ms} - 1)$-tuple over the binary field GF(2). Let α be a primitive element of the Galois field GF(2^{ms}). We may number the components of $\mathbf{v}$ with the nonzero elements of GF(2^{ms}) 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^{ms}) 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 flat F, we may form a vector over GF(2) as follows:

$$\mathbf{v}_F = (v_0, v_1, \ldots, v_{2^{ms}-2}),$$

whose ith 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.

Ex.1020 / Page 246 of 305
Sandisk Technologies, Inc. et al

Example 8.4

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 8.3, the four 1-flats passing through the point α^{14} but not the origin are

$$L_1 = \{\alpha^{14}, \alpha^7, \alpha^8, \alpha^{10}\}, \qquad L_2 = \{\alpha^{14}, \alpha^{13}, \alpha, \alpha^5\},$$
$$L_3 = \{\alpha^{14}, \alpha^0, \alpha^6, \alpha^2\}, \qquad L_4 = \{\alpha^{14}, \alpha^{12}, \alpha^{11}, \alpha^3\}.$$

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	1	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	0	1	1	0	1)

Definition 8.1. 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 EG(m, 2^s) that do not pass through the origin.

The generator polynomial of a (μ, s)th-order EG code will be 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 $W_{2^s}(h)$, is defined as the real sum of the coefficients in the radix 2^s expansion of h, that is,

$$W_{2^s}(h) = \sum_{i=0}^{m-1} \delta_i. \tag{8.3}$$

As an example, let $m = 3$ and $s = 2$. Then the integer $h = 45$ can be expanded 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 can be expressed 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 $h^{(0)} = h$.

Ex.1020 / Page 247 of 305
Sandisk Technologies, Inc. et al

Now we state a theorem (without proof) that characterizes the roots of the generator polynomial of a (μ, s)th-order EG code.

Theorem 8.1. 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

$$0 < \max_{0 \le l < s} W_{2^s}(h^{(l)}) \le (m - \mu - 1)(2^s - 1). \tag{8.4}$$

Example 8.5

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 8.1 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

$$0 < \max_{0 \le l < 2} W_{2^2}(h^{(l)}) \le 3.$$

The nonnegative integers less than 15 that satisfy the condition above are 1, 2, 3, 4, 6, 8, 9, and 12. Therefore, $\mathbf{g}(X)$ has α, α^2, α^3, α^4, α^6, α^8, α^9, and α^{12} as all its roots. The elements α, α^2, α^4, and α^8 have the same minimal polynomial $\phi_1(X) = 1 + X + X^4$, and the elements α^3, α^6, α^9, and α^{12} have the same minimal polynomial $\phi_3(X) = 1 + X + X^2 + X^3 + X^4$. 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 7.1. It is one-step majority-logic decodable.

Example 8.6

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. If follows from Theorem 8.1 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

$$0 < \max_{0 \le l < 2} W_{2^2}(h^{(l)}) \le 3.$$

The nonnegative integers less than 63 that satisfy the condition above 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. α, α^2, α^4, α^8, α^{16}, and α^{32} have $\phi_1(X) = 1 + X + X^6$ as their minimal polynomial.
2. α^3, α^6, α^{12}, α^{24}, α^{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 $\phi_9(X) = 1 + X^2 + X^3$.

Ex.1020 / Page 248 of 305
Sandisk Technologies, Inc. et al

Therefore, the generator polynomial of the (1, 2)th-order EG code of length 63 is

$$g(X) = (1 + X + X^6)(1 + X + X^2 + X^4 + X^6)(1 + X^2 + X^3)$$
$$= 1 + X^2 + X^4 + X^{11} + X^{13} + X^{14} + X^{15}.$$

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 EG$(m, 2^s)$. From Definition 8.1, we know that the null space of the code contains the incidence vectors of all the $(\mu + 1)$-flats of 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 (8.2) we know that there are

$$J = \frac{2^{(m-\mu)s} - 1}{2^s - 1} \tag{8.5}$$

$(\mu + 1)$-flats not passing through the origin which 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, for every μ-flat $F^{(\mu)}$ passing through the point $\alpha^{2^{ms}-2}$, the error sum $S(F^{(\mu)})$ can be determined. This forms the first step of orthogonalization.

The error sums $S(F^{(\mu)})$'s corresponding to all the μ-flats $F^{(\mu)}$ that pass through the point $\alpha^{2^{ms}-2}$ are then used for the second step of orthogonalization. Let $F^{(\mu-1)}$ be a $(\mu - 1)$-flat passing through the point $\alpha^{2^{ms}-2}$. From (8.2) 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 which 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 for every $(\mu - 1)$-flat $F^{(\mu-1)}$ passing through the point $\alpha^{2^{ms}-2}$ but not the origin, the error sum $S(F^{(\mu-1)})$ can be determined. 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 (8.2) 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

Ex.1020 / Page 249 of 305
Sandisk Technologies, Inc. et al

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. Since the code is cyclic, other error digits can be decoded in the same manner successively.

Since 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

$$t_{\mathrm{ML}} = \left\lfloor \frac{2^{(m-\mu)s}-1}{2(2^s-1)} - \frac{1}{2} \right\rfloor \tag{8.6}$$

or fewer errors. 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 8.7

Let $m = 2$, $s = 2$ and $\mu = 0$. Consider the (0, 2)th-order EG code of length 15. From Example 8.5 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 in 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 8.4. 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 7.1.

Example 8.8

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 8.1 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 condition above, we find that $\mathbf{g}(X)$ has the following roots: α, α^2, α^3, α^4, α^5, α^6, α^8, α^9, α^{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 7.12.

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

Ex.1020 / Page 250 of 305
Sandisk Technologies, Inc. et al

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

$\{\alpha^{13}, \alpha^{14}\}$, $\{\alpha^{12}, \alpha^{14}\}$, $\{\alpha^{11}, \alpha^{14}\}$, $\{\alpha^{10}, \alpha^{14}\}$, $\{\alpha^{9}, \alpha^{14}\}$, $\{\alpha^{8}, \alpha^{14}\}$,

$\{\alpha^{7}, \alpha^{14}\}$, $\{\alpha^{6}, \alpha^{14}\}$, $\{\alpha^{5}, \alpha^{14}\}$, $\{\alpha^{4}, \alpha^{14}\}$, $\{\alpha^{3}, \alpha^{14}\}$, $\{\alpha^{2}, \alpha^{14}\}$, $\{\alpha, \alpha^{14}\}$.

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

$\{\alpha^{4}, \alpha^{10}, \alpha^{13}, \alpha^{14}\}$, $\{\alpha^{7}, \alpha^{12}, \alpha^{13}, \alpha^{14}\}$, $\{\alpha^{9}, \alpha^{11}, \alpha^{13}, \alpha^{14}\}$,

$\{\alpha^{0}, \alpha^{8}, \alpha^{13}, \alpha^{14}\}$, $\{\alpha^{1}, \alpha^{5}, \alpha^{13}, \alpha^{14}\}$, $\{\alpha^{3}, \alpha^{6}, \alpha^{13}, \alpha^{14}\}$.

The incidence vectors for these six 2-flats are:

	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}
$\mathbf{w}_{11} = (0$	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	1	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).

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

$$A_{11} = \mathbf{w}_{11}\cdot\mathbf{r} = e_4 + e_{10} + e_{13} + e_{14}$$
$$A_{12} = \mathbf{w}_{12}\cdot\mathbf{r} = e_7 + e_{12} + e_{13} + e_{14}$$
$$A_{13} = \mathbf{w}_{13}\cdot\mathbf{r} = e_9 + e_{11} + e_{13} + e_{14}$$
$$A_{14} = \mathbf{w}_{14}\cdot\mathbf{r} = e_0 + e_8 + e_{13} + e_{14}$$
$$A_{15} = \mathbf{w}_{15}\cdot\mathbf{r} = e_1 + e_5 + e_{13} + e_{14}$$
$$A_{16} = \mathbf{w}_{16}\cdot\mathbf{r} = e_3 + e_6 + e_{13} + e_{14}.$$

We see that these six check sums orthogonal on $\{e_{13}, e_{14}\}$ are exactly the same check sums given in Example 7.12. Thus, the error sum $e_{13} + e_{14}$ corresponding to the 1-flat $\{\alpha^{13}, \alpha^{14}\}$ can be determined 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}. Since $J = 6$, we only need to determine six error sums corresponding to any six 1-flats passing through α^{14}. These error sums are then used 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. Complicate combinatorial expressions for the number of parity-check digits of EG codes can be found in References 4 and 5.

Ex.1020 / Page 251 of 305
Sandisk Technologies, Inc. et al

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

$$n - k = \binom{m+1}{m}^s - 1.$$

This result was obtained independently by Smith [6] and by MacWilliams and Mann [7].

For $s = 1$, we obtain another special subclass of EG codes, which is known as the class *Reed–Muller* (RM) codes. A $(\mu, 1)$th-order EG code is called a μth-order RM code. 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 8.1 that the generator polynomial $g(X)$ of a μth-order RM code of length $2^m - 1$ has α^h as a root if and only if

$$0 < W_2(h) \leq m - \mu - 1. \tag{8.7}$$

It has been proved that a μth-order RM code of length $n = 2^m - 1$ has the following parameters:

$$k = \sum_{i=0}^{\mu} \binom{m}{i},$$

$$d_{\min} = 2^{m-\mu} - 1,$$

$$J = 2^{m-\mu} - 2.$$

Since $J = d_{\min} - 1$, RM codes are completely orthogonalizable. Reed–Muller codes in noncyclic form were first constructed by Muller in 1954 [8]. In the same year, Reed [9] devised a majority-logic decoding algorithm for these codes. Cyclic structure of RM codes was proved independently by Kasami et al. [10, 11] and Kolesnik and Mironchikov [12].

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 more 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 8.4. See Reference 13 for a more extensive list.

EG codes were first studied by Rudolph [3]. Rudolph's work was later extended and generalized by other coding theorists [13–18]. Improvements for decoding EG codes were suggested by Weldon [19] and by Chen [20]. 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 [13, 16–18, 21] which all contain EG codes as subclasses. These generalizations will not be covered here. However, a simple generalization using parallel flats is presented next.

Ex.1020 / Page 252 of 305
Sandisk Technologies, Inc. et al

TABLE 8.4 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	30	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
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

Ex.1020 / Page 253 of 305
Sandisk Technologies, Inc. et al

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 $\{F, F_1\}$. Since 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 form a $(\mu, 2)$-frame. There are $2^s - 1$ such $(\mu, 2)$-frames which 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 one and only one of these* $2^s - 1$ $(\mu, 2)$-*frames*. Since there are

$$\frac{2^{(m-\mu)s} - 1}{2^s - 1}$$

$(\mu + 1)$-flats that intersect on F, there are

$$(2^s - 1) \cdot \frac{2^{(m-\mu)s} - 1}{2^s - 1} = 2^{(m-\mu)s} - 1 \tag{8.8}$$

$(\mu, 2)$-frames that intersect on F. Any point outside F is in 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

$$2^{(m-\mu)s} - 2 \tag{8.9}$$

$(\mu, 2)$-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

$$\mathbf{v} = (v_0, v_1, \ldots, v_{2^{ms}-2})$$

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:

$$\mathbf{v}_Q = (v_0, v_1, \ldots, v_{2^{ms}-2}),$$

where the ith component

$$v_i = \begin{cases} 1 & \text{if } \alpha^i \text{ is a point in } Q \\ 0 & \text{otherwise.} \end{cases}$$

Definition 8.2. 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) [22] which characterizes the roots of the generator polynomial of a (μ, s)th-order twofold EG code.

Theorem 8.2. 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

$$0 < \max_{0 \leq l < s} W_2(h^{(l)}) < (m - \mu)(2^s - 1). \tag{8.10}$$

Ex.1020 / Page 254 of 305
Sandisk Technologies, Inc. et al

Example 8.9

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 (8.10) 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

$$0 < \max_{0 \le l < 3} W_{2^3}(h^{(l)}) < 7.$$

The nonnegative integers less than 63 that satisfy the conditions above 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:

$$\alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{12},$$
$$\alpha^{16}, \alpha^{17}, \alpha^{20}, \alpha^{24}, \alpha^{32}, \alpha^{33}, \alpha^{34}, \alpha^{40}, \alpha^{48}.$$

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 $m = 3$ is a (63, 45) cyclic code. In fact, it is the (63, 45) BCH code with 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 EG($m, 2^s$) that do not pass 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 (8.9) we see that there are

$$J = 2^{(m-\mu)s} - 2 \tag{8.11}$$

$(\mu, 2)$-frames not passing through the origin which 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 this error sum, $S(F^{(\mu)})$, can be determined correctly from the J check sums orthogonal on it provided that there are no more than

$$\left\lfloor \frac{J}{2} \right\rfloor = 2^{(m-\mu)s-1} - 1 \tag{8.12}$$

Ex.1020 / Page 255 of 305
Sandisk Technologies, Inc. et al

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 μth-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

$$t_{\mathrm{ML}} = \left\lfloor \frac{J}{2} \right\rfloor = 2^{(m-\mu)s-1} - 1 \tag{8.13}$$

or fewer errors with majority-logic decoding. It has been proved [22] 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 coces is completely orthogonalizable.

Example 8.10

Consider the decoding of the (1, 3)th-order twofold EG of length 63 with $m = 2$ and $s = 3$. In Example 8.9 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 EG(2, 2^3) that do not pass through the origin. Regard GF(2^6) as the geometry EG(2, 2^3). Let α be a primitive element of GF(2^6) (use Table 6.2). From (8.1) we see that there are nine lines in EG(2, 2^3) that intersect at the point α^{62}. Eight of these lines do not pass the origin. From (8.9) 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 the location α^{62}. Since $J = 6$, we only need to find six lines in EG(2, 2^3) that intersect at the point α^{62} and do not pass the origin.

Let $\beta = \alpha^9$. Then $0, 1, \beta, \beta^2, \beta^3, \beta^4, \beta^5$, and β^6 ($\beta^7 = 1$) form a subfield, GF(2^3), of the field GF(2^6) (use Table 6.2). Then each line in 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 α^{62} are as follows:

$$L_1 = \{\alpha^{11}, \alpha^{16}, \alpha^{18}, \alpha^{24}, \alpha^{48}, \alpha^{58}, \alpha^{59}, \alpha^{62}\},$$
$$L_2 = \{\alpha^{1}, \alpha^{7}, \alpha^{31}, \alpha^{41}, \alpha^{42}, \alpha^{45}, \alpha^{57}, \alpha^{62}\},$$
$$L_3 = \{\alpha^{23}, \alpha^{33}, \alpha^{34}, \alpha^{37}, \alpha^{49}, \alpha^{54}, \alpha^{56}, \alpha^{62}\},$$
$$L_4 = \{\alpha^{2}, \alpha^{12}, \alpha^{19}, \alpha^{21}, \alpha^{27}, \alpha^{51}, \alpha^{61}, \alpha^{62}\},$$
$$L_5 = \{\alpha^{0}, \alpha^{3}, \alpha^{15}, \alpha^{20}, \alpha^{22}, \alpha^{28}, \alpha^{52}, \alpha^{62}\},$$
$$L_6 = \{\alpha^{9}, \alpha^{10}, \alpha^{13}, \alpha^{25}, \alpha^{30}, \alpha^{32}, \alpha^{38}, \alpha^{62}\}.$$

For each of the lines above, we want to 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 given above 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,

$$(\{\alpha^{62} + \eta\alpha\}, \{\alpha^{62} + \alpha^9 + \eta\alpha\})$$

Ex.1020 / Page 256 of 305
Sandisk Technologies, Inc. et al

form 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 8.5A to 8.5F.

To decode the code, the incidence vectors of the 36 (1, 2)-frames given in Tables 8.5A to 8.5F 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 on it. Thus, $S(L_i)$ can be determined correctly provided that there are three or fewer errors in the error

TABLE 8.5A POLYNOMIALS ORTHOGONAL ON $\{e_{11}, e_{16}, e_{18}, e_{24}, e_{48}, e_{58}, e_{59}, e_{62}\}$

$w_{11}(X)^* = (11, 16, 18, 21, 24, 31, 32, 35, 47, 48, 52, 54, 58, 59, 60, 62)$
$w_{12}(X) = (11, 12, 16, 18, 22, 23, 24, 26, 38, 43, 45, 48, 51, 58, 59, 62)$
$w_{13}(X) = (0, 6, 11, 16, 18, 24, 30, 40, 41, 44, 48, 56, 58, 59, 61, 62)$
$w_{14}(X) = (4, 5, 8, 11, 16, 18, 20, 24, 25, 27, 33, 48, 57, 58, 59, 62)$
$w_{15}(X) = (3, 11, 13, 14, 16, 17, 18, 24, 29, 34, 36, 42, 48, 58, 59, 62)$
$w_{16}(X) = (2, 7, 9, 11, 15, 16, 18, 24, 39, 48, 49, 50, 53, 58, 59, 62)$

TABLE 8.5B POLYNOMIALS ORTHOGONAL ON $\{e_1, e_7, e_{31}, e_{41}, e_{42}, e_{45}, e_{57}, e_{62}\}$

$w_{21}(X) = (1, 7, 13, 23, 24, 27, 31, 39, 41, 42, 44, 45, 46, 52, 57, 62)$
$w_{22}(X) = (1, 7, 22, 31, 32, 33, 36, 41, 42, 45, 48, 53, 55, 57, 61, 62)$
$w_{23}(X) = (0, 1, 7, 12, 17, 19, 25, 31, 41, 42, 45, 49, 57, 59, 60, 62)$
$w_{24}(X) = (1, 5, 6, 7, 9, 21, 26, 28, 31, 34, 41, 42, 45, 57, 58, 62)$
$w_{25}(X) = (1, 3, 7, 8, 10, 16, 31, 40, 41, 42, 45, 50, 51, 54, 57, 62)$
$w_{26}(X) = (1, 4, 7, 14, 15, 18, 30, 31, 35, 37, 41, 42, 43, 45, 57, 62)$

*In Table 8.5A to 8.5F, the integers inside the parentheses are powers of X.

TABLE 8.5C 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 8.5D POLYNOMIALS ORTHOGONAL ON $\{e_2, e_{14}, e_{19}, e_{21}, e_{27}, e_{51}, e_{61}, e_{62}\}$

$w_{41}(X) = (2, 7, 8, 11, 14, 19, 21, 23, 27, 28, 30, 36, 51, 60, 61, 62)$
$w_{42}(X) = (0, 2, 14, 19, 21, 24, 27, 34, 35, 38, 50, 51, 55, 57, 61, 62)$
$w_{43}(X) = (2, 14, 15, 19, 21, 25, 26, 27, 29, 41, 46, 48, 51, 54, 61, 62)$
$w_{44}(X) = (1, 2, 3, 9, 14, 19, 21, 27, 33, 43, 44, 47, 51, 59, 61, 62)$
$w_{45}(X) = (2, 6, 14, 16, 17, 19, 20, 21, 27, 32, 37, 39, 45, 51, 61, 62)$
$w_{46}(X) = (2, 5, 10, 12, 14, 18, 19, 21, 27, 42, 51, 52, 53, 56, 61, 62)$

Ex.1020 / Page 257 of 305
Sandisk Technologies, Inc. et al

TABLE 8.5E 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 8.5F POLYNOMIALS ORTHOGONAL ON $\{e_9, e_{10}, e_{13}, e_{25}, e_{30}, e_{32}, e_{38}, e_{62}\}$

$w_{61}(X) = (3, 5, 9, 10, 11, 13, 25, 30, 32, 35, 38, 45, 46, 49, 61, 62)$
$w_{62}(X) = (9, 10, 13, 17, 25, 27, 28, 30, 31, 32, 38, 43, 48, 50, 56, 62)$
$w_{63}(X) = (0, 1, 4, 9, 10, 13, 16, 21, 23, 25, 29, 30, 32, 38, 53, 62)$
$w_{64}(X) = (8, 9, 10, 13, 18, 19, 22, 25, 30, 32, 34, 38, 39, 41, 47, 62)$
$w_{65}(X) = (7, 9, 10, 12, 13, 14, 20, 25, 30, 32, 38, 44, 54, 55, 58, 62)$
$w_{66}(X) = (2, 9, 10, 13, 25, 26, 30, 32, 36, 37, 38, 40, 52, 57, 59, 62)$

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. Since 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 [22]

$$n - k = \binom{m+1}{m}^s - \binom{m}{m-1}^s. \tag{8.14}$$

A list of twofold EG codes is given in Table 8.6. 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

TABLE 8.6 LIST OF TWOFOLD EG CODES*

m	s	μ	n	k	J	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.

Ex.1020 / Page 258 of 305
Sandisk Technologies, Inc. et al

is a twostep majority-logic decodable (255, 191) twofold EG code; the corresponding RM code is a (255, 163) code which is five-step majority-logic decodable (using Chen's decoding algorithm [20], it may be decoded in two steps); the corresponding BCH code is a (255, 199) code.

The concept of twofold EG codes can be generalized to form a large class of multifold EG codes [18, 22]. The class of twofold EG codes is a proper subclass of several large classes of majority-logic decodable codes which are constructed based on Euclidean geometry [16–18, 22].

8.3 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 $GF(2^{(m+1)s})$ which contains $GF(2^s)$ as a subfield. Let α be a primitive element in $GF(2^{(m+1)s})$. Then the powers of α, $\alpha^0, \alpha^1, \ldots, \alpha^{2^{(m+1)s}-2}$, form all the nonzero elements of $GF(2^{(m+1)s})$. Let

$$n = \frac{2^{(m+1)s} - 1}{2^s - 1} = 2^{ms} + 2^{(m-1)s} + \cdots + 2^s + 1. \tag{8.15}$$

Then the order of $\beta = \alpha^n$ is $2^s - 1$. The 2^s elements $0, 1, \beta, \beta^2, \ldots, \beta^{2^s-2}$ form the Galois field $GF(2^s)$.

Consider the first n powers of α,

$$\Gamma = \{\alpha^0, \alpha^1, \alpha^2, \ldots, \alpha^{n-1}\}.$$

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 $\alpha^{i-j} = \eta$. Since $\eta^{2^s-1} = 1$, 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:

$$\{\alpha^0, \beta\alpha^0, \beta^2\alpha^0, \ldots, \alpha^{2^s-2}\alpha^0\},$$
$$\{\alpha^1, \beta\alpha^1, \beta^2\alpha^1, \ldots, \beta^{2^s-2}\alpha^1\},$$
$$\{\alpha^2, \beta\alpha^2, \beta^2\alpha^2, \ldots, \beta^{2^s-2}\alpha^2\},$$
$$\vdots$$
$$\{\alpha^{n-1}, \beta\alpha^{n-1}, \beta^2\alpha^{n-1}, \ldots, \beta^{2^s-2}\alpha^{n-1}\},$$

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:

$$(\alpha^i) = \{\alpha^i, \beta\alpha^i, \ldots, \beta^{2^s-2}\alpha^i\},$$

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). The n elements

$$(\alpha^0), (\alpha^1), (\alpha^2), \ldots, (\alpha^{n-1})$$

Ex.1020 / Page 259 of 305
Sandisk Technologies, Inc. et al

are said to form an *m-dimensional projective geometry over* GF(2^s), denoted by PG(m, 2^s). The elements $(\alpha^0), (\alpha^1), \ldots, (\alpha^{n-1})$ are called the *points* of PG(m, 2^s). Note that the $2^s - 1$ elements in $\{\alpha^i, \beta\alpha^i, \ldots, \beta^{2^s-2}\alpha^i\}$ are considered to be the *same* point. This is a *major difference* between a projective geometry and an Euclidean geometry.

Let (α^i) and (α^j) be any two *distinct* points in PG(m, 2^s). Then the *line* (1-flat) passing through (α^i) and (α^j) consists of points of the following form:

$$(\eta_1\alpha^i + \eta_2\alpha^j), \tag{8.16}$$

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:

$$\eta_1\alpha^i + \eta_2\alpha^j, \beta\eta_1\alpha^i + \beta\eta_2\alpha^j, \ldots, \beta^{2^s-2}\eta_1\alpha^i + \beta^{2^s-2}\eta_2\alpha^j$$

represent the same point in PG($2^{(m+1)s}$). 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 8.11

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, β, and β^2 form the field GF(2^2). The geometry PG(2, 2^2) consists of the following 21 points:

$$(\alpha^0),\ (\alpha^1),\ (\alpha^2),\ (\alpha^3),\ (\alpha^4),\ (\alpha^5),\ (\alpha^6),$$
$$(\alpha^7),\ (\alpha^8),\ (\alpha^9),\ (\alpha^{10}),\ (\alpha^{11}),\ (\alpha^{12}),\ (\alpha^{13}),$$
$$(\alpha^{14}),\ (\alpha^{15}),\ (\alpha^{16}),\ (\alpha^{17}),\ (\alpha^{18}),\ (\alpha^{19}),\ (\alpha^{20}),$$

Consider the line passing through the point (α) and (α^{20}) which 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, β, β^2}. The five distinct points are

$$(\alpha),$$
$$(\alpha^{20}),$$
$$(\alpha + \alpha^{20}) = (\alpha^{57}) = (\beta^2\alpha^{15}) = (\alpha^{15}),$$
$$(\alpha + \beta\alpha^{20}) = (\alpha + \alpha^{41}) = (\alpha^{56}) = (\beta^2\alpha^{14}) = (\alpha^{14}),$$
$$(\alpha + \beta^2\alpha^{20}) = (\alpha + \alpha^{62}) = (\alpha^{11}).$$

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^i) 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 $\{(\eta_1\alpha^i + \eta_2\alpha^j)\}$ have (α^j) as a common point (the only common point).

Ex.1020 / Page 260 of 305
Sandisk Technologies, Inc. et al

We say that they intersect at (α^j). The number of lines in PG$(m, 2^s)$ that intersect at a given point is

$$\frac{2^{ms}-1}{2^s-1} = 1 + 2^s + \cdots + 2^{(m-1)s}. \tag{8.17}$$

Let $(\alpha^{l_1}), (\alpha^{l_2}), \ldots, (\alpha^{l_{\mu+1}})$ 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

$$(\eta_1\alpha^{l_1} + \eta_2\alpha^{l_2} + \cdots + \eta_{\mu+1}\alpha^{l_{\mu+1}}), \tag{8.18}$$

where $\eta_i \in$ GF(2^s) and not all $\eta_1, \eta_2, \ldots, \eta_{\mu+1}$ are zero. There are $2^{(\mu+1)s} - 1$ choices for $\eta_1, \eta_2, \ldots, \eta_{\mu+1}$ ($\eta_1 = \eta_2 = \cdots = \eta_{\mu+1} = 0$ is not allowed). Since 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

$$\frac{2^{(\mu+1)s}-1}{2^s-1} = 1 + 2^s + \cdots + 2^{\mu s} \tag{8.19}$$

points in a μ-flat in PG$(m, 2^s)$. Let $\alpha^{l'_{\mu+1}}$ be a point not in the μ-flat

$$\{(\eta_1\alpha^{l_1} + \eta_1\alpha^{l_2} + \cdots + \eta_{\mu+1}\alpha^{l_{\mu+1}})\}.$$

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 $\{(\eta_1\alpha^{l_1} + \eta_2\alpha^{l_2} + \cdots + \eta_\mu\alpha^{l_\mu})\}$. The number of μ-flats in PG$(m, 2^s)$ that intersect on a given $(\mu - 1)$-flat in PG$(m, 2^s)$ is

$$\frac{2^{(m-\mu+1)s}-1}{2^s-1} = 1 + 2^s + \cdots + 2^{(m-\mu)s}. \tag{8.20}$$

Every point outside a $(\mu - 1)$-flat F is in one and only one of the μ-flats intersecting on F.

Let $\mathbf{v} = (v_0, v_1, \ldots, v_{n-1})$ be a 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 PG$(m, 2^s)$. The incidence vector for F is an n-tuple over GF(2),

$$\mathbf{v}_F = (v_0, v_1, \ldots, v_{n-1}),$$

whose ith component

$$v_i = \begin{cases} 1 & \text{if } (\alpha^i) \text{ is a point in } F \\ 0 & \text{otherwise.} \end{cases}$$

Definition 8.3. 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 PG$(m, 2^s)$.

Ex.1020 / Page 261 of 305
Sandisk Technologies, Inc. et al

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 (8.3). The following theorem characterizes the roots of the generator polynomial of a (μ, s)th-order PG code (the proof is omitted).

Theorem 8.3. 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

$$0 \leq \max_{0 \leq l < s} W_{2^s}(h^{(l)}) = j(2^s - 1) \tag{8.21}$$

with $0 \leq j \leq m - \mu$.

Example 8.12

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 form Theorem 8.3 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 condition above are 0, 3, 6, 9, 12, 18, 24, 33, 36, and 48. Thus, $\mathbf{g}(X)$ has $\alpha^0 = 1$, α^3, α^6, α^9, α^{12}, α^{18}, α^{24}, α^{33}, α^{36}, and α^{48} as roots. From Table 6.3 we find that (1) α^3, α^6, α^{12}, α^{24}, α^{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 7.9.

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 (8.20) 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 $F^{(\mu-1)}$.

Ex.1020 / Page 262 of 305
Sandisk Technologies, Inc. et al

Then this error sum, $S(F^{(\mu-1)})$, can be determined correctly 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}). These error sums are then used to determine the error sums, $S(F^{(\mu-2)})$'s, corresponding to all the $(\mu - 2)$-flats that contain (α^{n-1}). This process continues 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, the value of e_{n-1} can be determined. A total of μ steps of orthogonalization are required to decode e_{n-1}. Since the code is cyclic, other error digits can be decoded in the same manner. Thus, the code is μ-step decodable. At the rth 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, the error sums needed for the next step can always be determined correctly provided that there are no more than $\lfloor J/2 \rfloor$ errors in the received vector. Thus, the μth-order PG code of length $n = (2^{(m+1)s} - 1)/(2^s - 1)$ is capable of correcting

$$t_{\mathrm{ML}} = \left\lfloor \frac{J}{2} \right\rfloor = \left\lfloor \frac{2^{(m-\mu+1)s} - 1}{2(2^s - 1)} \right\rfloor \tag{8.22}$$

or fewer errors with majority-logic decoding.

Example 8.13

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 PG(2, 2^2). Let α be a primitive element in GF(2^6). The geometry PG(2, 2^2) consists of 21 points, (α^0) to (α^{20}), as given in Example 8.11. Let $\beta = \alpha^{21}$. Then, 0, 1, β, and β^2 form the field GF(2^2).

There are $2^2 + 1 = 5$ lines passing through the point (α^{20}), which are

$$\{(\eta_1\alpha^0 + \eta_2\alpha^{20})\} = \{(\alpha^0), (\alpha^5), (\alpha^7), (\alpha^{17}), (\alpha^{20})\},$$
$$\{(\eta_1\alpha^1 + \eta_2\alpha^{20})\} = \{(\alpha^1), (\alpha^{11}), (\alpha^{14}), (\alpha^{15}), (\alpha^{20})\},$$
$$\{(\eta_1\alpha^2 + \eta_2\alpha^{20})\} = \{(\alpha^2), (\alpha^3), (\alpha^8), (\alpha^{10}), (\alpha^{20})\},$$
$$\{(\eta_1\alpha^4 + \eta_2\alpha^{20})\} = \{(\alpha^4), (\alpha^6), (\alpha^{16}), (\alpha^{19}), (\alpha^{20})\},$$
$$\{(\eta_1\alpha^9 + \eta_2\alpha^{20})\} = \{(\alpha^9), (\alpha^{12}), (\alpha^{13}), (\alpha^{18}), (\alpha^{20})\}.$$

The incidence vectors of these lines (in polynomial form) are

$$\mathbf{w}_1(X) = 1 + X^5 + X^7 + X^{17} + X^{20},$$
$$\mathbf{w}_2(X) = X + X^{11} + X^{14} + X^{15} + X^{20},$$
$$\mathbf{w}_3(X) = X^2 + X^3 + X^8 + X^{10} + X^{20},$$
$$\mathbf{w}_4(X) = X^4 + X^6 + X^{16} + X^{19} + X^{20},$$
$$\mathbf{w}_5(X) = X^9 + X^{12} + X^{13} + X^{18} + X^{20}.$$

Ex.1020 / Page 263 of 305
Sandisk Technologies, Inc. et al

These vectors are orthogonal on digit position 20. They are exactly the orthogonal vectors for the (21, 11) difference-set code given in Example 7.10.

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 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 References 4 and 5. 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

$$n - k = 1 + \binom{m+1}{m}^s. \tag{8.23}$$

This expression was obtained independently by Goethals and Delsarte [23], Smith [6], and MacWilliams and Mann [7]. A list of PG codes is given in Table 8.7.

PG codes were first studied by Rudolph [3] and were later extended and generalized by many others [6, 13–16, 23, 24]. Extensive treatment for EG and PG codes can be found in References 25 to 28.

TABLE 8.7 LIST OF 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
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

8.4 MODIFICATIONS OF MAJORITY-LOGIC DECODING

For large J and μ, the conventional rule of orthogonalization for decoding long finite geometry codes described in Sections 8.2 and 8.3 requires a very large number of majority-logic gates, and thus the decoding complexity becomes prohibitive.

Ex.1020 / Page 264 of 305
Sandisk Technologies, Inc. et al

However, if we are willing to sacrifice the decoding speed, the decoding complexity may be reduced. This trade-off between the decoding complexity and the decoding speed is best illustrated by an example.

Consider the (1, 1)th-order (15, 5) EG code. This code is two-step majority-logic decodable and is capable of correcting three or fewer errors (see Examples 7.12 and 8.8). If the conventional rule of orthogonalization for decoding EG codes is used, the decoder would consist of seven six-input majority-logic gates; six gates are used at the first level to determine the check sums corresponding to six 1-flats in EG(4, 2) that pass through the point α^{14} and one gate is used at the second level to determine the value of the error digit e_{14}, as shown in Figure 7.9. We see that the outputs of the *rightmost* first-level majority-logic gate at the *successive times* are

$$S_1 = e_{13} + e_{14}, \quad S_2 = e_{12} + e_{13}, \quad S_3 = e_{11} + e_{12},$$
$$S_4 = e_{10} + e_{11}, \quad S_5 = e_9 + e_{10}, \quad S_6 = e_8 + e_9, \ldots.$$

If we add these sums successively as follows:

$$\begin{aligned} S_1 &= e_{13} + e_{14}, \\ S_1 + S_2 &= e_{12} + e_{14}, \\ S_1 + S_2 + S_3 &= e_{11} + e_{14}, \\ S_1 + S_2 + S_3 + S_4 &= e_{10} + e_{14}, \\ S_1 + S_2 + S_3 + S_4 + S_5 &= e_9 + e_{14}, \\ S_1 + S_2 + S_3 + S_4 + S_5 + S_6 &= e_8 + e_{14}, \\ &\vdots \end{aligned}$$

we obtain check sums orthogonal on e_{14}. Hence, e_{14} can be determined from these orthogonal check sums. Forming check sums orthogonal on e_{14} in this manner, only one majority-logic gate is needed at the first level of decoding. Since $S_1, S_2, \ldots, S_6, \ldots$ are formed at successive times, a buffer register is needed to store them. Since $J = 6$, to determine the value of e_{14} we only need to form the check sums $S_1, S_1 + S_2, \ldots, S_1 + S_2 + S_3 + S_4 + S_5 + S_6$. As a result, we obtain a decoding circuit for the (15, 5) EG codes as shown in Figure 8.1. Since five clock times are needed to form the check sums S_1 to S_6, the received vector must be delayed five units of time before it can be decoded. This delay is achieved by a buffer register of five stages.

The implementation described above for decoding the (1, 1)th-order (15, 5) EG code requires only two majority-logic gates, one at each level of orthogonalization. Therefore, a reduction of five majority-logic gates is obtained. This reduction is achieved at the cost of an increase of five units of time for decoding the received vector and 10 extra memory elements for buffering. The memory elements are much cheaper than the majority-logic gates. As a result, a reduction in decoding complexity is obtained.

Rudolph and Hartmann [29] have shown that finite geometry codes of length up to several thousand digits can be decoded in a manner similar to the example given above. For a L-step decodable geometry (EG or PG) code, only one J-input majority-

Ex.1020 / Page 265 of 305
Sandisk Technologies, Inc. et al

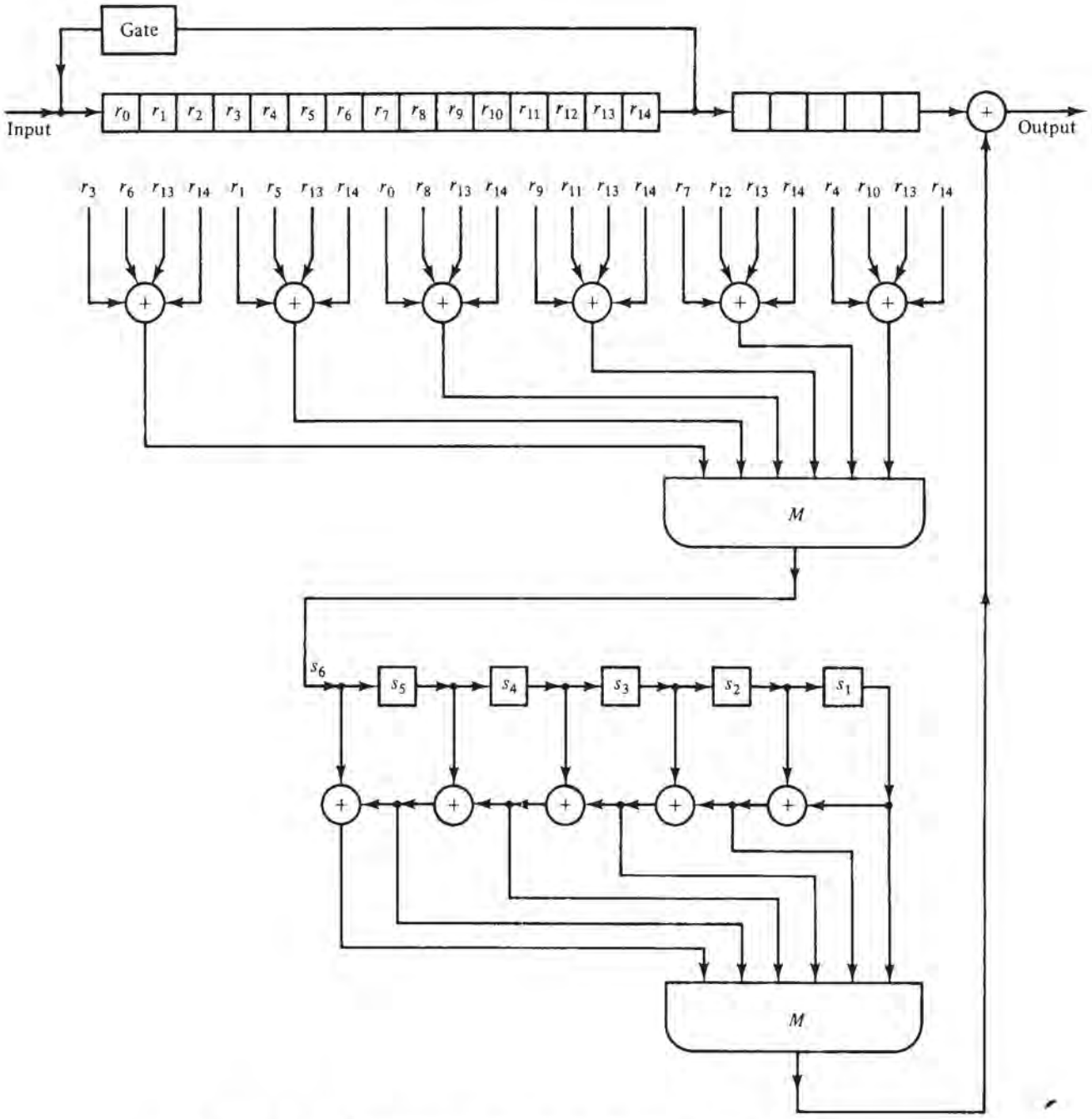


Figure 8.1 SCR majority-logic decoder for the (1, 1)th order (15, 5) EG code.

logic gate is needed at each level of orthogonalization. The successive check sums formed at the output of the ith-level majority-logic gate are first stored in a buffer register and are then *combined* to form J check sums that are orthogonal on a properly selected set of error digits. These J orthogonal check sums are inputs to the $(i + 1)$th-level majority-logic gate. The successive check sums formed at the output of this $(i + 1)$th-level gate are again stored and combined to form J check sums orthogonal on another properly selected set of error digits. These J new orthogonal check sums form the inputs to the $(i + 2)$th-level majority-logic gate. This process continues

Ex.1020 / Page 266 of 305
Sandisk Technologies, Inc. et al

until the last level of orthogonalization is reached; the output of the last-level majority-logic gate is the estimated value of a specific error digit. The majority-logic decoding algorithm described above is called *sequential-code-reduction* (SCR) decoding. Using this decoding technique, only L majority-logic gates are required for a L-step SCR majority-logic decodable code, one majority-logic gate at each level of orthogonalization. A buffer register of no more than n stages is needed between every two consecutive levels of orthogonalization.

Using SCR technique to decode finite geometry codes with large L and J greatly reduces the number of majority-logic gates, from $1 + J + J^2 + \cdots + J^{L-1}$ to L. Furthermore, the number of modulo-2 adders needed to form check sums are also reduced significantly. There is an increase of buffer storage, from one buffer register of n stages (for storing the received vector) to L buffer registers. The major disadvantage of SCR majority-logic decoding is the increase in decoding time. The conventional majority-logic decoding requires n clock times to decode the received vector; however, the SCR majority-logic decoding requires nL clock times to decode the received vector. In communication systems where decoding speed is critical, the SCR decoding algorithm may not be suitable.

Consider the decoding of the highest-order error digit e_{n-1} with SCR technique. Let

$$S_0 = e_{j_1} + e_{j_2} + \cdots + e_{j_m} + e_{n-1}$$

be the first check sum formed at the output of the $(i-1)$th-level majority-logic gate where $0 \leq j_1 < j_2 < \cdots < j_m < n-1$. Let

$$\mathbf{w}(X) = X^{j_1} + X^{j_2} + \cdots + X^{j_m} + X^{n-1}$$

be the orthogonal polynomial associated to S_0. At the successive clock times, the outputs of the $(i-1)$th-level majority-logic gate are the following check sums:

$$\begin{aligned}
S_1 &= e_{j_1-1} + e_{j_2-1} + \cdots + e_{j_m-1} + e_{n-2}, \\
S_2 &= e_{j_1-2} + e_{j_2-2} + \cdots + e_{j_m-2} + e_{n-3}, \\
&\vdots \\
S_l &= e_{j_1-l} + e_{j_2-l} + \cdots + e_{j_m-l} + e_{n-1-l} \\
&\vdots
\end{aligned}$$

with $1 \leq l < n$. Note that the indices of the error digits are cyclically shifted to the left one place every clock time (if $j_t - l$ is negative, it is replaced by $n + j_t - l$). The polynomial associated to the check sum S_l is

$$\mathbf{w}^{(-l)}(X) = X^{j_1-l} + X^{j_2-l} + \cdots + X^{j_m-l} + X^{n-1-l},$$

which is obtained by cyclically shifting $\mathbf{w}(X)$ to the left l times (or to the right $n-l$ times). In fact, $\mathbf{w}^{(-l)}(X)$ is the remainder resulting from dividing $X^{n-l}\mathbf{w}(X)$ by $X^n + 1$ (see Chapter 4). Therefore, we have the following relationship:

$$X^{n-l}\mathbf{w}(X) = q_l(X)(X^n + 1) + \mathbf{w}^{(-l)}(X). \tag{8.24}$$

Ex.1020 / Page 267 of 305
Sandisk Technologies, Inc. et al

Let

$$Q = e_{t_1} + e_{t_2} + \cdots + e_{t_p} + e_{n-1}$$

be the first check sum formed at the output of the ith-level majority-logic gate. Using SCR majority-logic decoding, the check sums orthogonal on Q are linear combinations of $S_0, S_1, S_2, \ldots$. Let ψ_l be a check sum orthogonal on Q. Then

$$\psi_l = b_0 S_0 + b_{n-1} S_1 + b_{n-2} S_2 + \cdots + b_1 S_{n-1}. \tag{8.25}$$

Let $\mathbf{f}_l(X)$ be the orthogonal polynomial associated with the error digits in ψ_l. Then, from (8.25), we have

$$\mathbf{f}_l(X) = b_0 \mathbf{w}(X) + b_{n-1} \mathbf{w}^{(-1)}(X) + b_{n-2} \mathbf{w}^{(-2)}(X) + \cdots + b_1 \mathbf{w}^{(-n+1)}(X). \tag{8.26}$$

Combining (8.24) and (8.26), we obtain

$$\mathbf{f}_l(X) = (b_0 + b_1 X + \cdots + b_{n-1} X^{n-1}) \mathbf{w}(X) + \mathbf{a}_l(X)(X^n + 1). \tag{8.27}$$

From (8.27) we see that each orthogonal polynomial $\mathbf{f}_l(X)$ at the ith level of orthogonalization is a multiple of $\mathbf{w}(X)$ plus a term $\mathbf{a}_l(X)(X^n + 1)$. The polynomial $\mathbf{w}(X)$ is called the *generating orthogonal polynomial.* At each level of SCR majority-logic decoding, if we know $\mathbf{w}(X)$ and $\mathbf{f}_l(X)$ and if we are able to express $\mathbf{f}_l(X)$ in the form of (8.27), we can form check sums orthogonal on Q from (8.25). Therefore, a cyclic code is SCR majority-logic decodable if and only if there exists a generating orthogonal polynomial at each level of orthogonalization.

Example 8.14

Let $m = 5$, $s = 1$, and $\mu = 2$. Consider the second-order RM code of length $n = 2^5 - 1 = 31$. Let α be a primitive element of GF(2^5) (see Table 8.8). Let h be a nonnegative integer less than 31. It follows from (8.7) that the generator polynomial $\mathbf{g}(X)$ of this RM code has α^h as a root if and only if

$$0 < W_2(h) \leq 2.$$

The nonnegative integers less than 31 that satisfy the condition above are 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 16, 17, 18, 20, and 24. Therefore, $\mathbf{g}(X)$ has α^1, α^2, α^3, α^4, α^5, α^6, α^8, α^9, α^{10}, α^{12}, α^{16}, α^{17}, α^{18}, α^{20}, and α^{24} as roots. The minimal polynomial of α, α^2, α^4, α^8, and α^{16} is $\phi_1(X) = 1 + X^2 + X^5$. The minimal polynomial of α^3, α^6, α^{12}, α^{17}, and α^{24} is $\phi_3(X) = 1 + X^2 + X^3 + X^4 + X^5$. The minimal polynomial of α^5, α^9, α^{10}, α^{18}, and α^{20} is $\phi_5(X) = 1 + X + X^2 + X^4 + X^5$. Thus,

$$\begin{aligned}\mathbf{g}(X) &= \phi_1(X)\phi_3(X)\phi_5(X) \\ &= 1 + X + X^2 + X^3 + X^5 + X^7 + X^8 + X^9 + X^{10} + X^{11} + X^{15}.\end{aligned}$$

Hence, the code is a (31, 16) cyclic code. Since

$$J = 2^{(5-2)} - 1 - 1 = 6,$$

it is capable of correcting three or fewer errors with majority-logic decoding.

The null space of this code contains incidence vectors of all the 3-flats in EG(5, 2) that do not pass the origin. These incidence vectors can be used to form parity-check sums for decoding. If the conventional rule of orthogonalization is used for decoding this code, three levels of orthogonalization and a total of 43 six-input majority-logic gates are needed. However, if we examine the geometry EG(5, 2) carefully, this code can be decoded in two steps using SCR majority-logic decoding.

Ex.1020 / Page 268 of 305
Sandisk Technologies, Inc. et al

TABLE 8.8 GALOIS FIELD GF(2^5) WITH $p(X) = 1 + \alpha^2 + \alpha^5 = 0$

0	(0 0 0 0 0)
1	(1 0 0 0 0)
α	(0 1 0 0 0)
α^2	(0 0 1 0 0)
α^3	(0 0 0 1 0)
α^4	(0 0 0 0 1)
$\alpha^5 = 1 + \alpha^2$	(1 0 1 0 0)
$\alpha^6 = \alpha + \alpha^3$	(0 1 0 1 0)
$\alpha^7 = \alpha^2 + \alpha^4$	(0 0 1 0 1)
$\alpha^8 = 1 + \alpha^2 + \alpha^3$	(1 0 1 1 0)
$\alpha^9 = \alpha + \alpha^3 + \alpha^4$	(0 1 0 1 1)
$\alpha^{10} = 1 + \alpha^4$	(1 0 0 0 1)
$\alpha^{11} = 1 + \alpha + \alpha^2$	(1 1 1 0 0)
$\alpha^{12} = \alpha + \alpha^2 + \alpha^3$	(0 1 1 1 0)
$\alpha^{13} = \alpha^2 + \alpha^3 + \alpha^4$	(0 0 1 1 1)
$\alpha^{14} = 1 + \alpha^2 + \alpha^3 + \alpha^4$	(1 0 1 1 1)
$\alpha^{15} = 1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4$	(1 1 1 1 1)
$\alpha^{16} = 1 + \alpha + \alpha^3 + \alpha^4$	(1 1 0 1 1)
$\alpha^{17} = 1 + \alpha + \alpha^4$	(1 1 0 0 1)
$\alpha^{18} = 1 + \alpha$	(1 1 0 0 0)
$\alpha^{19} = \alpha + \alpha^2$	(0 1 1 0 0)
$\alpha^{20} = \alpha^2 + \alpha^3$	(0 0 1 1 0)
$\alpha^{21} = \alpha^3 + \alpha^4$	(0 0 0 1 1)
$\alpha^{22} = 1 + \alpha^2 + \alpha^4$	(1 0 1 0 1)
$\alpha^{23} = 1 + \alpha + \alpha^2 + \alpha^3$	(1 1 1 1 0)
$\alpha^{24} = \alpha + \alpha^2 + \alpha^3 + \alpha^4$	(0 1 1 1 1)
$\alpha^{25} = 1 + \alpha^3 + \alpha^4$	(1 0 0 1 1)
$\alpha^{26} = 1 + \alpha + \alpha^2 + \alpha^4$	(1 1 1 0 1)
$\alpha^{27} = 1 + \alpha + \alpha^3$	(1 1 0 1 0)
$\alpha^{28} = \alpha + \alpha^2 + \alpha^4$	(0 1 1 0 1)
$\alpha^{29} = 1 + \alpha^3$	(1 0 0 1 0)
$\alpha^{30} = \alpha + \alpha^4$	(0 1 0 0 1)

Consider the following six 2-flats

$$\{\alpha^3, \alpha^{11}, \alpha^{14}, \alpha^{30}\}, \quad \{\alpha^7, \alpha^{23}, \alpha^{29}, \alpha^{30}\}, \quad \{\alpha^{15}, \alpha^{16}, \alpha^{28}, \alpha^{30}\},$$
$$\{\alpha^6, \alpha^{17}, \alpha^{27}, \alpha^{30}\}, \quad \{\alpha^{19}, \alpha^{20}, \alpha^{21}, \alpha^{30}\}, \quad \{\alpha^8, \alpha^{10}, \alpha^{12}, \alpha^{30}\}.$$

For each of these 2-flats, there exist six 3-flats intersecting (orthogonal) on it. For example, the six 3-flats intersecting on $\{\alpha^3, \alpha^{11}, \alpha^{14}, \alpha^{30}\}$ are

$$\{\alpha^0, \alpha^3, \alpha^7, \alpha^{11}, \alpha^{12}, \alpha^{14}, \alpha^{16}, \alpha^{30}\},$$
$$\{\alpha^1, \alpha^3, \alpha^8, \alpha^{11}, \alpha^{14}, \alpha^{21}, \alpha^{26}, \alpha^{30}\},$$
$$\{\alpha^2, \alpha^3, \alpha^{10}, \alpha^{11}, \alpha^{14}, \alpha^{24}, \alpha^{27}, \alpha^{30}\},$$
$$\{\alpha^3, \alpha^{11}, \alpha^{13}, \alpha^{14}, \alpha^{17}, \alpha^{19}, \alpha^{29}, \alpha^{30}\},$$
$$\{\alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^{11}, \alpha^{14}, \alpha^{15}, \alpha^{30}\},$$
$$\{\alpha^3, \alpha^{11}, \alpha^{14}, \alpha^{18}, \alpha^{20}, \alpha^{25}, \alpha^{28}, \alpha^{30}\}.$$

The error sums corresponding to the 2-flats above are

$$\psi_0 = e_3 + e_{11} + e_{14} + e_{30}, \qquad \psi_1 = e_7 + e_{23} + e_{29} + e_{30},$$

Ex.1020 / Page 269 of 305
Sandisk Technologies, Inc. et al

$$\psi_2 = e_{15} + e_{16} + e_{28} + e_{30}, \qquad \psi_3 = e_6 + e_{17} + e_{27} + e_{30},$$
$$\psi_4 = e_{19} + e_{20} + e_{21} + e_{30}, \qquad \psi_5 = e_8 + e_{10} + e_{12} + e_{30}.$$

Each of these error sums can be estimated from six parity-check sums orthogonal on it which are formed based on the incidence vectors of six 3-flats (in the null space of the code). We see that the error sums, ψ_0 to ψ_5, are orthogonal on the error digit e_{30}. Therefore, once ψ_0 to ψ_5 are formed, the error digit e_{30} can be estimated. Consequently, one level of orthogonalization (estimating error sums corresponding to 1-flats from error sums corresponding to 2-flats) is removed, and the code can be decoded in two steps. If we form ψ_0 to ψ_5 in parallel, the decoding circuit requires seven six-input majority-logic gates.

Next we show that the (31, 16) RM code is SCR majority-logic decodable. The incidence polynomials corresponding to the six 2-flats that intersect on α^{30} are

$$\mathbf{f}_0(X) = X^3 + X^{11} + X^{14} + X^{30}, \qquad \mathbf{f}_1(X) = X^7 + X^{23} + X^{29} + X^{30},$$
$$\mathbf{f}_2(X) = X^{15} + X^{16} + X^{28} + X^{30}, \qquad \mathbf{f}_3(X) = X^6 + X^{17} + X^{27} + X^{30},$$
$$\mathbf{f}_4(X) = X^{19} + X^{20} + X^{21} + X^{30}, \qquad \mathbf{f}_5(X) = X^8 + X^{10} + X^{12} + X^{30}.$$

Let $\mathbf{w}(X) = \mathbf{f}_0(X) = X^3 + X^{11} + X^{14} + X^{30}$ be the polynomial associated to the first parity-check sum S_0 formed at the output of the first-level majority-logic gate, (i.e., $S_0 = e_3 + e_{11} + e_{14} + e_{30}$). Then each of the polynomials $\mathbf{f}_l(X)$'s above can be expressed as a multiple of $\mathbf{w}(X)$ plus $\mathbf{a}_l(X)(X^{31} + 1)$ as follows:

$$\begin{aligned}
\mathbf{f}_0(X) &= \mathbf{w}(X),\\
\mathbf{f}_1(X) &= (1 + X^4 + X^{12} + X^{15})\mathbf{w}(X) + \mathbf{a}_1(X)(X^{31} + 1),\\
\mathbf{f}_2(X) &= (1 + X + X^2 + X^4 + X^6 + X^{10} + X^{12} + X^{13} + X^{15} + X^{17} + X^{18}\\
&\quad + X^{24})\mathbf{w}(X) + \mathbf{a}_2(X)(X^{31} + 1),\\
\mathbf{f}_3(X) &= (1 + X^3 + X^4 + X^8 + X^{16} + X^{19})\mathbf{w}(X) + \mathbf{a}_3(X)(X^{31} + 1),\\
\mathbf{f}_4(X) &= (1 + X + X^2 + X^3 + X^4 + X^5 + X^6 + X^9 + X^{10} + X^{12} + X^{16} + X^{19}\\
&\quad + X^{20} + X^{21} + X^{23} + X^{24})\mathbf{w}(X) + \mathbf{a}_4(X)(X^{31} + 1),\\
\mathbf{f}_5(X) &= (1 + X + X^2 + X^3 + X^5 + X^6 + X^7 + X^8 + X^{10} + X^{16} + X^{19} + X^{20}\\
&\quad + X^{21} + X^{24})\mathbf{w}(X) + \mathbf{a}_5(X)(X^{31} + 1).
\end{aligned} \tag{8.28}$$

Therefore, $\mathbf{w}(X) = \mathbf{f}_0(X) = X^3 + X^{11} + X^{14} + X^{30}$ is the generating orthogonal polynomial for the first-level orthogonalization. At the first level, only one majority-logic gate is needed. The first check sum to be formed at the output of this gate is $S_0 = \psi_0 = e_3 + e_{11} + e_{14} + e_{30}$ [corresponding to the generating orthogonal polynomial $\mathbf{w}(X)$]. Then, at the successive clock times, the outputs of this gate will be

$$\begin{aligned}
S_1 &= e_2 + e_{10} + e_{13} + e_{29},\\
S_2 &= e_1 + e_9 + e_{12} + e_{28},\\
S_3 &= e_0 + e_8 + e_{11} + e_{27},\\
S_4 &= e_7 + e_{10} + e_{26} + e_{30},\\
&\vdots\\
S_{30} &= e_0 + e_4 + e_{12} + e_{15}.
\end{aligned}$$

From these successive sums and (8.26) [using (8.25) and (8.27)], we obtain the following check sums orthogonal on e_{30}:

Ex.1020 / Page 270 of 305
Sandisk Technologies, Inc. et al

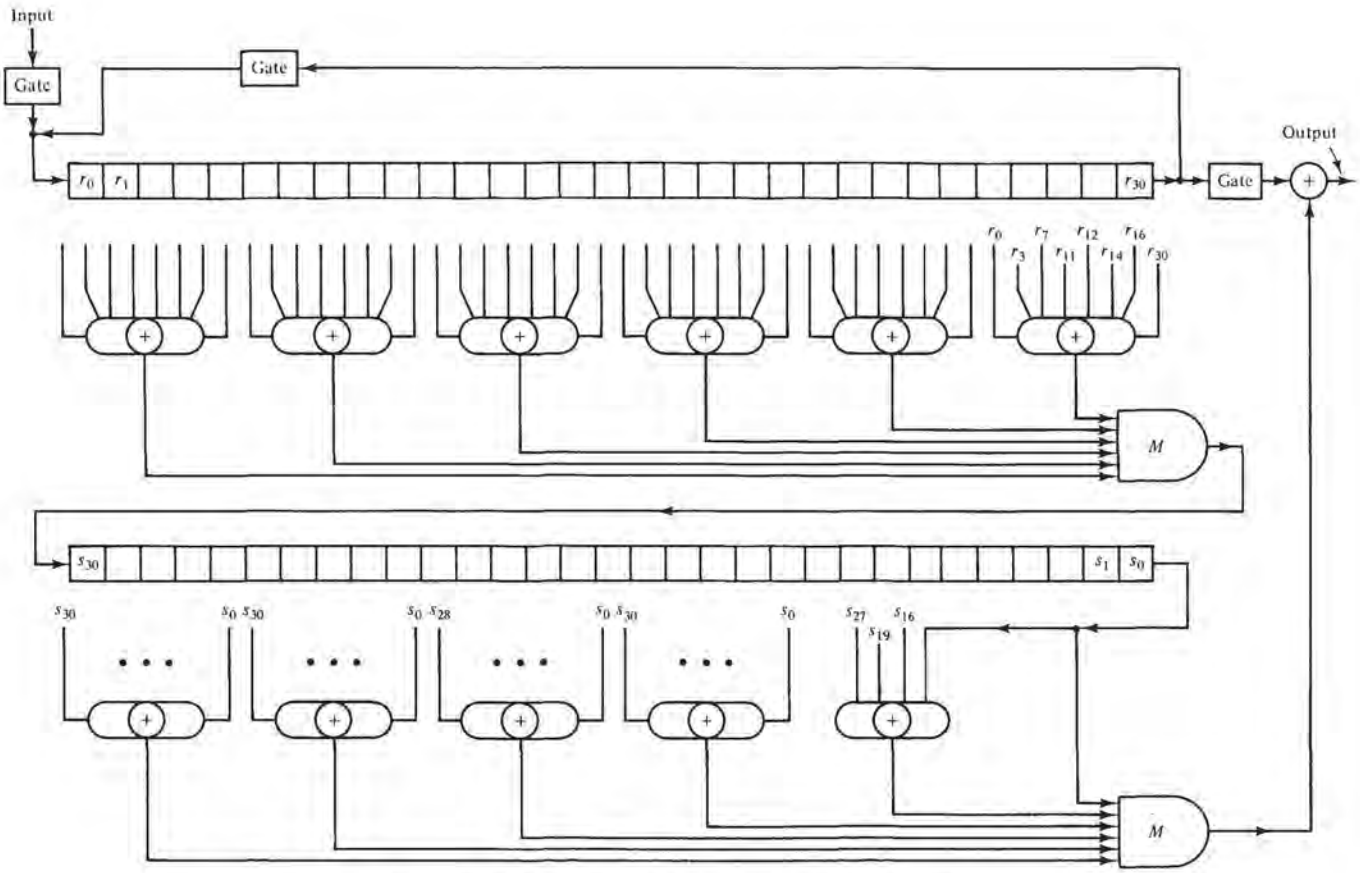


Figure 8.2 SCR majority-logic decoder for the (31, 16) RM code.

$$\psi_0 = S_0,$$

$$\psi_1 = S_0 + S_{16} + S_{19} + S_{27},$$

$$\psi_2 = S_0 + S_7 + S_{13} + S_{14} + S_{16} + S_{18} + S_{19} + S_{21} + S_{25} + S_{27} + S_{29} + S_{30},$$

$$\psi_3 = S_0 + S_{12} + S_{15} + S_{23} + S_{27} + S_{28},$$

$$\psi_4 = S_0 + S_7 + S_8 + S_{10} + S_{11} + S_{12} + S_{15} + S_{19} + S_{21} + S_{22} + S_{25} + S_{26} + S_{27} + S_{28} + S_{29} + S_{30},$$

$$\psi_5 = S_0 + S_7 + S_{10} + S_{11} + S_{12} + S_{15} + S_{21} + S_{23} + S_{24} + S_{25} + S_{26} + S_{28} + S_{29} + S_{30}.$$

Forming ψ_0 to ψ_5 in this manner, we obtain the SCR majority-logic decoding circuit as shown in Figure 8.2, where only two majority-logic gates are used.

The difficulty with SCR majority-logic decoding is that it is hard to find the generating orthogonal polynomial at each level of orthogonalization. For more on SCR majority-logic decoding, readers are referred to Reference 29. Besides EG and PG codes, there are many other generalized finite geometry codes which are SCR majority-logic decodable [13].

Majority-logic decoding using orthogonal parity-check sums can be generalized to decoding with nonorthogonal parity-check sums. Suppose that it is possible to form N parity-check sums such that:

1. Each sum contains all the error digits in $E = \{e_{j_1}, e_{j_2}, \ldots, e_{j_m}\}$.
2. Any error digit not in E is checked by at most λ sums.

Then we can readily see that if there are no more than $[N/2\lambda]$ errors in the received vector, the error sum

$$S(E) = e_{j_1} + e_{j_2} + \cdots + e_{j_m}$$

can be correctly determined from the parity-check sums above using majority-logic decision algorithm. These sums are called nonorthogonal parity-check sums. Using nonorthogonal parity-check sums, many more cyclic codes can be decoded with majority-logic decoding, such as the duals of primitive polynomial codes [21], generalized finite geometry codes [13, 16], and multifold EG codes [18, 22].

PROBLEMS

8.1. 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.

8.2. 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).

Ex.1020 / Page 272 of 305
Sandisk Technologies, Inc. et al

(a) Find all the 1-flats that pass through the point α^{63}.
(b) Find all the 2-flats that intersect on the 1-flat, $\{\alpha^{63} + \eta\alpha\}$, where $\eta \in GF(2^2)$.

8.3. 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}.

8.4. Consider the two-dimensional projective $PG(2, 2^2)$ constructed from $GF(2^{3\times 2})$. Construct all the 1-flats that pass through the point (α^7).

8.5. Let $m = 2$ and $s = 3$.
(a) Determine the 2^3-weight of 47.
(b) Determine $\max_{0\le l<3} W_{2^3}(47^{(l)})$.
(c) Determine all the positive integers h less than 63 such that

$$0 < \max_{0\le l<3} W_{2^3}(h^{(l)}) \le 2^3 - 1.$$

8.6. Find the generator polynomial of the first-order RM code of length $2^5 - 1$. Describe how to decode this code.

8.7. Find the generator polynomial of the third-order RM code of length $2^6 - 1$. Describe how to decode this code.

8.8. 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^{63} where α is a primitive element in $GF(2^{2\times 3})$. Design a type I majority-logic decoder for this code.

8.9. 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.

8.10. Let $m = 3$ and $s = 2$. Find the generator polynomial of the (1, 2)th-order PG code of length $(2^{4\times 2} - 1)/(2^2 - 1) = 85$. This code is two-step majority-logic decodable with $J = 4$. Find all the orthogonal polynomials at each step of orthogonalization.

8.11. The (7, 4) Hamming code is two-step majority-logic decodable (see Example 7.11). Show that this code is SCR decodable. Construct such a decoder.

8.12. Prove that the $(m - 2)$th-order RM code of length $2^m - 1$ is a Hamming code. (*Hint:* Show that its generator polynomial is a primitive polynomial of degree m.)

8.13. The first-order RM code of length $2^5 - 1$ is two-step majority-logic decodable.
(a) Find its generator polynomial.
(b) Show that it is SCR decodable.

8.14. Prove that the even-weight code vectors of the first-order RM code of length $2^m - 1$ form the maximum-length code of length $2^m - 1$.

8.15. Let $0 < \mu < m - 1$. Prove that the even-weight code vectors of the $(m - \mu - 1)$th-order RM code of length $2^m - 1$ form the dual of the μth-order RM code of length $2^m - 1$. [*Hint:* Let $\mathbf{g}(X)$ be the generator polynomial of the $(m - \mu - 1)$th-order RM code C. Show that the even-weight code vector of C is a cyclic code generated by $(X + 1)\mathbf{g}(X)$. Show that the dual of the μth-order RM code is also generated by $(X + 1)\mathbf{g}(X)$.]

8.16. The μth-order RM code of length $2^m - 1$ has minimum distance $d_{\min} = 2^{m-\mu} - 1$. 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 $\mathbf{g}(X)_{RM}$.]

8.17. Show that extended RM codes are invariant under the affine permutations.

Ex.1020 / Page 273 of 305
Sandisk Technologies, Inc. et al

REFERENCES

1. R. D. Carmichael, *Introduction to the Theory of Groups of Finite Order*, Dover, New York, 1956.
2. H. B. Mann, *Analysis and Design of Experiments*, Dover, New York, 1949.
3. L. D. Rudolph, "A Class of Majority Logic Decodable Codes," *IEEE Trans. Inf. Theory*, IT-13, pp. 305–307, April 1967.
4. 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, pp. 153–226, 1973.
5. S. Lin, "On the Number of Information Symbols in Polynomial Codes," *IEEE Trans. Inf. Theory*, IT-18, pp. 785–794, November 1972.
6. K. J. C. Smith, "Majority Decodable Codes Derived from Finite Geometries," Institute of Statistics Mimeo Series No. 561, University of North Carolina, Chapel Hill, N.C., 1967.
7. F. J. MacWilliams and H. B. Mann, "On the p-rank of the Design Matrix of a Different Set," *Inf. Control*, 12, pp. 474–488, 1968.
8. D. E. Muller, "Applications of Boolean Algebra to Switching Circuit Design and to Error Detection," *IRE Trans.*, EC-2, pp. 6–12, September 1954.
9. I. S. Reed, "A Class of Multiple-Error-Correcting Codes and the Decoding Scheme," *IRE Trans.*, IT-4, pp. 38–49, September 1954.
10. 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), pp. 100–106, September 1967.
11. T. Kasami, S. Lin, and W. W. Peterson, "New Generalizations of the Reed-Muller Codes, Part I: Primitive Codes," *IEEE Trans. Inf. Theory*, IT-14, pp. 189–199, March 1968.
12. V. D. Kolesnik and E. T. Mironchikov, "Cyclic Reed-Muller Codes and Their Decoding," *Probl. Inf. Transm.*, No. 4, pp. 15–19, 1968.
13. C. R. P. Hartmann, J. B. Ducey, and L. D. Rudolph, "On the Structure of Generalized Finite Geometry Codes," *IEEE Trans. Inf. Theory*, IT-20(2), pp. 240–252, March 1974.
14. D. K. Chow, "A Geometric Approach to Coding Theory with Applications to Information Retrieval," CSL Report No. R-368, University of Illinois, Urbana, Ill., 1967.
15. E. J. Weldon, Jr. "Euclidean Geometry Cyclic Codes," Proc. Symp. Combinatorial Math., University of North Carolina, Chapel Hill, N.C., April 1967.
16. P. Delsarte, "A Geometric Approach to a Class of Cyclic Codes," *J. Combinatorial Theory*, 6, pp. 340–358, 1969.
17. P. Delsarte, J. M. Goethals, and J. MacWilliams, "On GRM and Related Codes," *Inf. Control*, 16, pp. 403–442, July 1970.
18. S. Lin and K. P. Yiu, "An Improvement to Multifold Euclidean Geometry Codes," *Inf. Control*, 28(3), July 1975.
19. E. J. Weldon, Jr., "Some Results on Majority-Logic Decoding," Chap. 8, in *Error-Correcting Codes*, H. Mann, ed., Wiley, New York, 1968.
20. C. L. Chen, "On Majority-Logic Decoding of Finite Geometry Codes," *IEEE Trans. Inf. Theory*, IT-17(3), pp. 332–336, May 1971.
21. T. Kasami and S. Lin, "On Majority-Logic Decoding for Duals of Primitive Polynomial Codes," *IEEE Trans. Inf. Theory*, IT-17(3), pp. 322–331, May 1971.

Ex.1020 / Page 274 of 305
Sandisk Technologies, Inc. et al

22. S. Lin, "Multifold Euclidean Geometry Codes," *IEEE Trans. Inf. Theory*, IT-19(4), pp. 537–548, July 1973.
23. J. M. Goethals and P. Delsarte, "On a Class of Majority-Logic Decodable Codes," *IEEE Trans. Inf. Theory*, IT-14, pp. 182–189, March 1968.
24. E. J. Weldon, Jr., "New Generations of the Reed-Muller Codes, Part II: Non-primitive Codes," *IEEE Trans. Inf. Theory*, IT-14, pp. 199–205, March 1968.
25. E. R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.
26. W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2nd ed. MIT Press, Cambridge, Mass., 1972.
27. I. F. Blake and R. C. Mullin, *The Mathematical Theory of Coding*, Academic Press, New York, 1975.
28. F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, 1977.
29. L. D. Rudolph and C. R. P. Hartmann, "Decoding by Sequential Code Reduction," *IEEE Trans. Inf. Theory*, IT-19(4), pp. 549–555, July 1973.

Ex.1020 / Page 275 of 305
Sandisk Technologies, Inc. et al

9

Burst-Error-Correcting Codes

So far, we have been concerned primarily with coding techniques for channels on which transmission errors occur independently in digit positions (i.e., each transmitted digit is affected independently by noise). However, there are communication channels which are affected by disturbances that cause transmission errors to cluster into bursts. For example, on telephone lines, a stroke of lightening or a human-made electrical disturbance frequently affects many adjacent transmitted digits. On magnetic storage systems, magnetic tape defects may last up to several mils and cause clusters of errors. In general, codes for correcting random errors are not efficient for correcting burst errors. Therefore, it is desirable to design codes specifically for correcting burst errors. Codes of this kind are called burst-error-correcting codes.

Cyclic codes are effective not only for burst-error detection as discussed in Chapter 4; they are also very effective for burst-error correction. Many effective cyclic codes for correcting burst errors have been discovered for the past 20 years. Cyclic codes for single-burst-error correction were first studied by Abramson [1,2]. In an effort to generalize Abramson's results, Fire discovered a large class of burst-error-correcting cyclic codes [3]. Fire codes can be decoded with very simple circuitry. Besides the Fire codes, many other effective burst-error-correcting cyclic codes have been constructed both analytically and with the aid of a computer [4–22].

9.1 INTRODUCTION

A burst of length l is defined as a vector whose nonzero components are confined to l consecutive digit positions, the first and last of which are nonzero. For example, the error vector $\mathbf{e} = (0\ 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0)$ is a burst of length 6. A

Ex.1020 / Page 276 of 305
Sandisk Technologies, Inc. et al

linear code that is capable of correcting all error bursts of length l or less but not all error bursts of length $l + 1$ is called an *l-burst-error-correcting code*, or the code is said to have *burst-error-correcting capability l*.

It is clear that for given code length n and burst-error-correcting capability l, we desire to construct an (n, k) code with as small a redundancy $n - k$ as possible. We establish next certain restrictions on $n - k$ for given l, or restrictions on l for given $n - k$.

Theorem 9.1. A necessary condition for an (n, k) linear code to be able to correct all burst errors of length l or less is that no burst of length $2l$ or less can be a code vector.

Proof. Suppose that there exists a burst $\mathbf{v}$ of length $2l$ or less as a code vector. This code vector $\mathbf{v}$ can be expressed as a vector sum of two bursts $\mathbf{u}$ and $\mathbf{w}$ of length l or less (except the degenerate case, in which $\mathbf{v}$ is a burst of length 1). Then $\mathbf{u}$ and $\mathbf{w}$ must be in the same coset of a standard array for this code. If one of these two vectors is used as a coset leader (correctable error pattern), the other will be an uncorrectable error burst. As a result, this code would not be able to correct all burst errors of length l or less. Therefore, in order to correct all burst errors of length l or less, no burst of length $2l$ or less can be a code vector. Q.E.D.

Theorem 9.2. The number of parity-check digits of an (n, k) linear code that has no burst of length b or less as a code vector is at least b (i.e., $n - k \geq b$).

Proof. Consider the vectors whose nonzero components are confined to the first b digit positions. There are a toal of 2^b of them. No two such vectors can be in the same coset of a standard array for this code; otherwise, their vector sum, which is a burst of length b or less, would be a code vector. Therefore, these 2^b vectors must be in 2^b distinct cosets. There are a total of 2^{n-k} cosets for an (n, k) code. Thus, $n - k$ must be at least equal to b (i.e., $n - k \geq b$). Q.E.D.

It follows from Theorems 9.1 and 9.2 that we obtain a restriction on the number of parity-check digits of an l-burst-error-correcting code.

Theorem 9.3. The number of parity-check digits of an l-burst-error-correcting code must be at least $2l$, that is,

$$n - k \geq 2l. \tag{9.1}$$

For a given n and k, Theorem 9.3 implies that the burst-error-correcting capability of an (n, k) code is at most $\lfloor (n - k)/2 \rfloor$, that is,

$$l \leq \left\lfloor \frac{n - k}{2} \right\rfloor. \tag{9.2}$$

This is an upper bound on the burst-error-correcting capability of an (n, k) code and is called the Reiger bound [5]. Codes that meet the Reiger bound are said to be *optimal*. The ratio

$$z = \frac{2l}{n - k} \tag{9.3}$$

Ex.1020 / Page 277 of 305
Sandisk Technologies, Inc. et al

is used as a measure of the *burst-correcting efficiency* of a code. An optimal code has burst-correcting efficiency equal to 1.

It is possible to show that if an (n, k) code is designed to correct all burst errors of length l or less and simultaneously detect all burst errors of length $d \geq l$ or less, the number of parity-check digits of the code must be at least $l + d$ (see Problem 9.1).

9.2 DECODING OF SINGLE-BURST-ERROR-CORRECTING CYCLIC CODES

An l-burst-error-correcting cyclic code can be most easily decoded by the error-trapping technique, with a slight variation. Suppose that a code word $\mathbf{v}(X)$ from an l-burst-error-correcting (n, k) cyclic code is transmitted. Let $\mathbf{r}(X)$ and $\mathbf{e}(X)$ be the received and error vectors, respectively. Let

$$\mathbf{s}(X) = s_0 + s_1 X + \cdots + s_{n-k-1} X^{n-k-1}$$

be the syndrome of $\mathbf{r}(X)$. If the errors in $\mathbf{e}(X)$ are confined to the l high-order parity-check digit positions, $X^{n-k-l}, \ldots X^{n-k-2}, X^{n-k-1}$, of $\mathbf{r}(X)$, then the l high-order syndrome digits, $s_{n-k-l}, \ldots, s_{n-k-2}, s_{n-k-1}$, match the errors of $\mathbf{e}(X)$ and the $n-k-l$ low-order syndrome digits, $s_0, s_1, \ldots, s_{n-k-l-1}$, are zeros. Suppose that the errors in $\mathbf{e}(X)$ are not confined to the positions $X^{n-k-l}, \ldots, X^{n-k-2}, X^{n-k-1}$, of $\mathbf{r}(X)$ but are confined to l consecutive positions of $\mathbf{r}(X)$ (including the end-around case). Then, after a certain number of cyclic shifts of $\mathbf{r}(X)$, say i cyclic shifts, the errors will be shifted to the positions $X^{n-k-l}, \ldots, X^{n-k-2}, X^{n-k-1}$ of $\mathbf{r}^{(i)}(X)$, the ith shift of $\mathbf{r}(X)$. Let $\mathbf{s}^{(i)}(X)$ be the syndrome of $\mathbf{r}^{(i)}(X)$. Then the first l high-order digits of $\mathbf{s}^{(i)}(X)$ match the errors at the positions $X^{n-k-l}, \ldots, X^{n-k-2}, X^{n-k-1}$ of $\mathbf{r}^{(i)}(X)$, and the $n-k-l$ low-order digits of $\mathbf{s}^{(i)}(X)$ are zeros. Using these facts, we may trap the errors in the syndrome register by cyclic shifting $\mathbf{r}(X)$.

An error-trapping decoder for an l-burst-correcting cyclic code is shown in Figure 9.1, where the received vector is shifted into the syndrome register from the left end. The decoding procedure is described in the following steps:

Step 1. The received vector $\mathbf{r}(X)$ is shifted into the syndrome and buffer registers simultaneously. (If we do not want to decode the received parity-check digits, the buffer register needs only k stages.) As soon as $\mathbf{r}(X)$ has been shifted into the syndrome register, the syndrome $\mathbf{s}(X)$ is formed.

Step 2. The syndrome register starts to shift with gate 2 on. As soon as its $n-k-l$ leftmost stages contain only zeros, its l rightmost stages contain the burst-error pattern. The error correction begins. There are three cases to be considered.

Step 3. If the $n-k-l$ leftmost stages of the syndrome register contain all zeros after the ith shift for $0 \leq i \leq n-k-l$, the errors of the burst $\mathbf{e}(X)$ are confined to the parity-check positions of $\mathbf{r}(X)$. In this event, the k received information digits in the buffer register are error-free. Gate 4 is then activated and the k error-free information digits in the buffer are shifted out to the data sink. If the $n-k-l$ leftmost stages of the syndrome register never contain

Ex.1020 / Page 278 of 305
Sandisk Technologies, Inc. et al

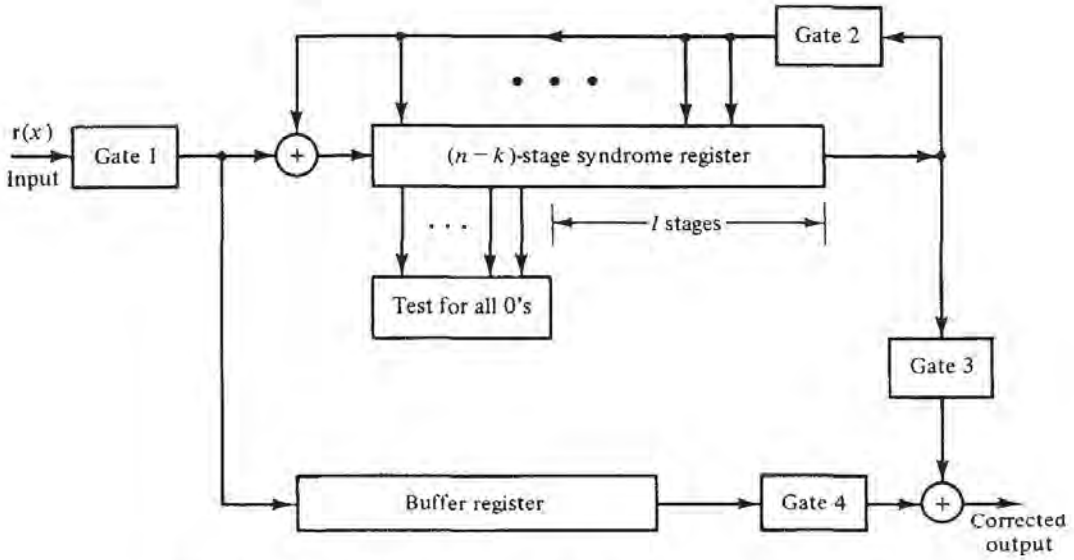


Figure 9.1 Error-trapping decoder for burst-error-correcting codes.

all zeros during the first $n - k - l$ shifts of the syndrome register, the error burst is not confined to the $n - k$ parity-check positions of $\mathbf{r}(X)$.

Step 4. If the $n - k - l$ leftmost stages of the syndrome register contain all zeros after the $(n - k - l + i)$th shift of the syndrome register for $1 \leq i \leq l$, the error burst is confined to positions $X^{n-i}, \ldots, X^{n-1}, X^0, \ldots, X^{l-i-1}$ of $\mathbf{r}(X)$. (This is an end-around burst.) In this event, the $l - i$ digits contained in the $l - i$ rightmost stages of the syndrome register match the errors at the parity-check positions, $X^0, X^1, \ldots, X^{l-i-1}$ of $\mathbf{r}(X)$, and the i digits contained in the next i stages of the syndrome register match the errors at the positions $X^{n-i}, \ldots, X^{n-2}, X^{n-1}$ of $\mathbf{r}(X)$. At this instant, a clock starts to count from $(n - k - l + i + 1)$. The syndrome register is then shifted (in step with the clock) with gate 2 turned off. As soon as the clock has counted up to $n - k$, the i rightmost digits in the syndrome register match the errors at the positions $X^{n-i}, \ldots, X^{n-2}, X^{n-1}$ of $\mathbf{r}(X)$. Gates 3 and 4 are then activated. The received information digits are read out of the buffer register and corrected by the error digits shifted out from the syndrome register.

Step 5. If the $n - k - l$ leftmost stages of the syndrome register never contains all zeros by the time that the syndrome register has been shifted $n - k$ times, the received information digits are read out of the buffer register one at a time with gate 4 activated. At the same time, the syndrome register is shifted with gate 2 activated. As soon as the $n - k - l$ leftmost stages of the syndrome register contain all zeros, the digits in the l rightmost stages of the syndrome register match the errors in the next l received information digits to come out of the buffer register. Gate 3 is then activated and the erroneous information digits are corrected by the digits coming out from the syndrome register with gate 2 disabled.

Ex.1020 / Page 279 of 305
Sandisk Technologies, Inc. et al

If the $n - k - l$ stages of the syndrome register never contain all zeros by the time the k information digits have been read out of the buffer, an uncorrectable burst of errors has been detected. With the decoder described above, the decoding process takes $2n$ clock cycles; the first n clock cycles are required for syndrome computation and the next n clock cycles are needed for error trapping and error correction. The n clock cycles for syndrome computation are concurrent with the reception of the received vector from the channel; no time delay occurs in this operation. The second n clock cycles for error trapping and correction represent decoding delay.

In this decoder, the received vector is shifted into the syndrome register from the left end. If the received vector is shifted into the syndrome register from the right end, the decoding operation would be slightly different (see Problem 9.2).

This decoder corrects only burst errors of length l or less. The number of these burst error patterns is $n2^{l-1}$, which, for large n, is only a small fraction of 2^{n-k} correctable error patterns (coset leaders). It is possible to modify the decoder in such a way that it corrects all the correctable burst errors of length $n - k$ or less. That is, besides correcting all the bursts of length l or less, the decoder also corrects those bursts of length $l + 1$ to $n - k$ which are used as coset leaders. This modified decoder operates as follows. The entire received vector is first shifted into the syndrome register. Before performing the error correction, the syndrome register is cyclically shifted n times (with feedback connections operative). During this cycling, the length b of the shortest burst that appears in the b rightmost stages of the syndrome register is recorded by a counter. This burst is assumed to be the error burst added by the channel. Having completed these precorrection shifts, the decoder begins its correction process. The syndrome register starts to shift again. As soon as the shortest burst reappears in the b rightmost stages of the syndrome register, the decoder starts to make corrections as described earlier. This decoding is an optimum decoding for burst-error-correcting codes which was proposed by Gallager [23].

9.3 SINGLE-BURST-ERROR-CORRECTING CODES

Fire Codes

Fire codes are the first class of cyclic codes constructed systematically for correcting burst errors. Let $\mathbf{p}(X)$ be an irreducible polynomial of degree m over GF(2). Let ρ be the smallest integer such that $\mathbf{p}(X)$ divides $X^{\rho} + 1$. The integer ρ is called the *period* of $\mathbf{p}(X)$. Let l be a positive integer such that $l \leq m$ and $2l - 1$ is not divisible by ρ. An l-burst-error-correcting Fire code is generated by the following polynomial:

$$\mathbf{g}(X) = (X^{2l-1} + 1)\mathbf{p}(X). \tag{9.4}$$

The length n of this code is the least common multiple of $2l - 1$ and the period ρ of $\mathbf{p}(X)$, that is,

$$n = \text{LCM}\,(2l - 1, \rho). \tag{9.5}$$

The number of parity-check digits of this code is $m + 2l - 1$. Note that the two factors $X^{2l-1} + 1$ and $\mathbf{p}(X)$ of $\mathbf{g}(X)$ are relatively prime.

Ex.1020 / Page 280 of 305
Sandisk Technologies, Inc. et al

Example 9.1

Consider the irreducible polynomial $\mathbf{p}(X) = 1 + X^2 + X^5$. Since $\mathbf{p}(X)$ is a primitive polynomial, its period is $\rho = 2^5 - 1 = 31$. Let $l = 5$. Clearly, 31 does not divide $2l - 1 = 9$. The Fire code generated by

$$\begin{aligned}\mathbf{g}(X) &= (X^9 + 1)(1 + X^2 + X^5) \\ &= 1 + X^2 + X^5 + X^9 + X^{11} + X^{14}\end{aligned}$$

has length $n = \text{LCM}(9, 31) = 279$. Therefore, it is a (279, 265) cyclic code that is capable of correcting any burst error of length 5 or less.

Next we prove that the Fire code generated by the polynomial of (9.4) is indeed capable of correcting any burst error of length l or less. To prove this it is sufficient to show that all the bursts of length l or less are in different cosets of the code; so they can be used as coset leaders and form correctable error patterns. Let $X^iA(X)$ and $X^jB(X)$ be the polynomial representations of two bursts of length l_1 and l_2, respectively, with $l_1 \leq l$ and $l_2 \leq l$, where

$$A(X) = 1 + a_1X + a_2X^2 + \cdots + a_{l_1-2}X^{l_1-2} + X^{l_1-1}$$

and

$$B(X) = 1 + b_1X + b_2X^2 + \cdots + b_{l_2-2}X^{l_2-2} + X^{l_2-1}.$$

Suppose that $X^iA(X)$ and $X^jB(X)$ are in the same coset of the code. Then the polynomial

$$\mathbf{v}(X) = X^iA(X) + X^jB(X) \tag{9.6}$$

must be a code polynomial in the code. Without loss of generality, we assume that $i \leq j$. Dividing $j - i$ by $2l - 1$, we obtain

$$j - i = q(2l - 1) + b \tag{9.7}$$

where $0 \leq b < 2l - 1$. Substituting (9.7) into (9.6), the polynomial $\mathbf{v}(X)$ can be expressed in the form

$$\mathbf{v}(X) = X^i[A(X) + X^bB(X)] + X^{i+b}B(X)[X^{q(2l-1)} + 1]. \tag{9.8}$$

Since $\mathbf{v}(X)$ is a code polynomial based on the assumption that $X^iA(X)$ and $X^jB(X)$ are in the same coset, $\mathbf{v}(X)$ must be divisible by $X^{2l-1} + 1$ (a factor of the generator polynomial). Since $X^{q(2l-1)} + 1$ is divisible by $X^{2l-1} + 1$, it follows from (9.8) that $A(X) + X^bB(X)$ is either divisible by $X^{2l-1} + 1$ or equal to zero. Suppose that

$$A(X) + X^bB(X) = D(X)(X^{2l-1} + 1). \tag{9.9}$$

Let d be the degree of $D(X)$. Then the degree of $D(X)(X^{2l-1} + 1)$ is $2l - 1 + d$. Since the degree of $A(X)$ is $l_1 - 1$, which is less than $2l - 1$, the degree of $A(X) + X^bB(X)$ must be the degree of $X^bB(X)$, which is $b + l_2 - 1$. From (9.9) we obtain

$$b + l_2 - 1 = 2l - 1 + d. \tag{9.10}$$

Since $l_1 \leq l$ and $l_2 \leq l$, it follows from (9.10) that

$$b \geq l_1 + d. \tag{9.11}$$

Since $l_1 - 1 \geq 0$, it follows from the equality above that

$$\begin{aligned} b &> l_1 - 1, \\ b &> d. \end{aligned} \tag{9.12}$$

Ex.1020 / Page 281 of 305
Sandisk Technologies, Inc. et al

From the equalities of (9.12), we see that $A(X) + X^b B(X)$ has the term X^b. Since $d < b < 2l - 1$, $D(X)(X^{2l-1} + 1)$ does not have the term X^b. This contradicts the hypothesis that $A(X) + X^b B(X) = D(X)(X^{2l-1} + 1)$. Therefore, we must have $D(X) = 0$ and $A(X) + X^b B(X) = 0$. This requires that $b = 0$ and

$$A(X) = B(X). \tag{9.13}$$

Since $b = 0$, it follows from (9.7) that

$$j - i = q(2l - 1). \tag{9.14}$$

Substituting (9.13) and (9.14) into (9.8), we obtain

$$\mathbf{v}(X) = X^i B(X)(X^{j-i} + 1). \tag{9.15}$$

Note that the degree of $B(X)$ is $l_2 - 1$, which is less than l. Therefore, the degree of $B(X)$ is less than the degree m of $\mathbf{p}(X)$, and $B(X)$ and $\mathbf{p}(X)$ are relatively prime. Since $\mathbf{v}(X)$ is assumed to be a code polynomial, $X^{j-i} + 1$ must be divisible by $\mathbf{p}(X)$. As a result, $j - i$ must be a multiple of ρ, the period of $\mathbf{p}(X)$. From (9.14) we see that $j - i$ is also a multiple of $2l - 1$. Therefore, $j - i$ must be a multiple of $n = \text{LCM}(2l - 1, \rho)$. This is impossible since both j and i are less than n and $j - i$ cannot be a multiple of n. Therefore, our hypothesis that two bursts, $X^i A(X)$ and $X^j B(X)$, of length l or less are in the same coset is invalid. As a result, all the bursts of length l or less are in different cosets of the Fire code generated by $\mathbf{g}(X)$ of (9.4) and they are correctable error patterns. Since the code is cyclic, it also corrects the end-around bursts of length l or less.

Fire codes can be decoded with the error-trapping circuit shown in Figure 9.1. The error-trapping decoder for the (279, 265) Fire code considered in Example 9.1 is shown in Figure 9.2.

In a data transmission (or storage) system, if the receiver has some computation capability, a fast decoder for Fire codes may be implemented. Consider a Fire code with generator polynomial $\mathbf{g}(X) = (X^{2l-1} + 1)\mathbf{p}(X)$, where $2l - 1$ and the period ρ of $\mathbf{p}(X)$ are relatively prime. Let $\mathbf{r}(X)$ be the received polynomial. Let $\mathbf{s}_1(X)$ and $\mathbf{s}_2(X)$ be the remainders resulting from dividing $\mathbf{r}(X)$ by $X^{2l-1} + 1$ and $\mathbf{p}(X)$, respectively. Then we may take

$$[\mathbf{s}_1(X), \mathbf{s}_2(X)]$$

as a syndrome of $\mathbf{r}(X)$. We can readily see that $\mathbf{s}_1(X) = \mathbf{s}_2(X) = 0$ if and only if $\mathbf{r}(X)$ is a code polynomial. If $\mathbf{r}(X)$ contains a nonzero error burst of length l or less, we must have $\mathbf{s}_1(X) \neq 0$ and $\mathbf{s}_2(X) \neq 0$. If $\mathbf{s}_1(X) = 0$ and $\mathbf{s}_2(X) \neq 0$ [or $\mathbf{s}_1(X) \neq 0$ and $\mathbf{s}_2(X) = 0$], then $\mathbf{r}(X)$ must contain a detectable but uncorrectable burst of length greater than l.

Now, consider an error-trapping decoder as shown in Figure 9.3. This decoder consists of two syndrome registers; one is called the *error-pattern register* and the other is called the *error-location register*. The feedback connections of the error-pattern register are based on the factor $X^{2l-1} + 1$, and the feedback connections of the error-location register are based on the factor $\mathbf{p}(X)$. The received polynomial $\mathbf{r}(X)$ is first read into the two syndrome registers and the buffer register. As soon as the entire $\mathbf{r}(X)$ has been shifted into the two syndrome registers, $\mathbf{s}_1(X)$ and $\mathbf{s}_2(X)$ are formed. The decoder tests $\mathbf{s}_1(X)$ and $\mathbf{s}_2(X)$. If $\mathbf{s}_1(X) = \mathbf{s}_2(X) = 0$, the received poly-

Ex.1020 / Page 282 of 305
Sandisk Technologies, Inc. et al

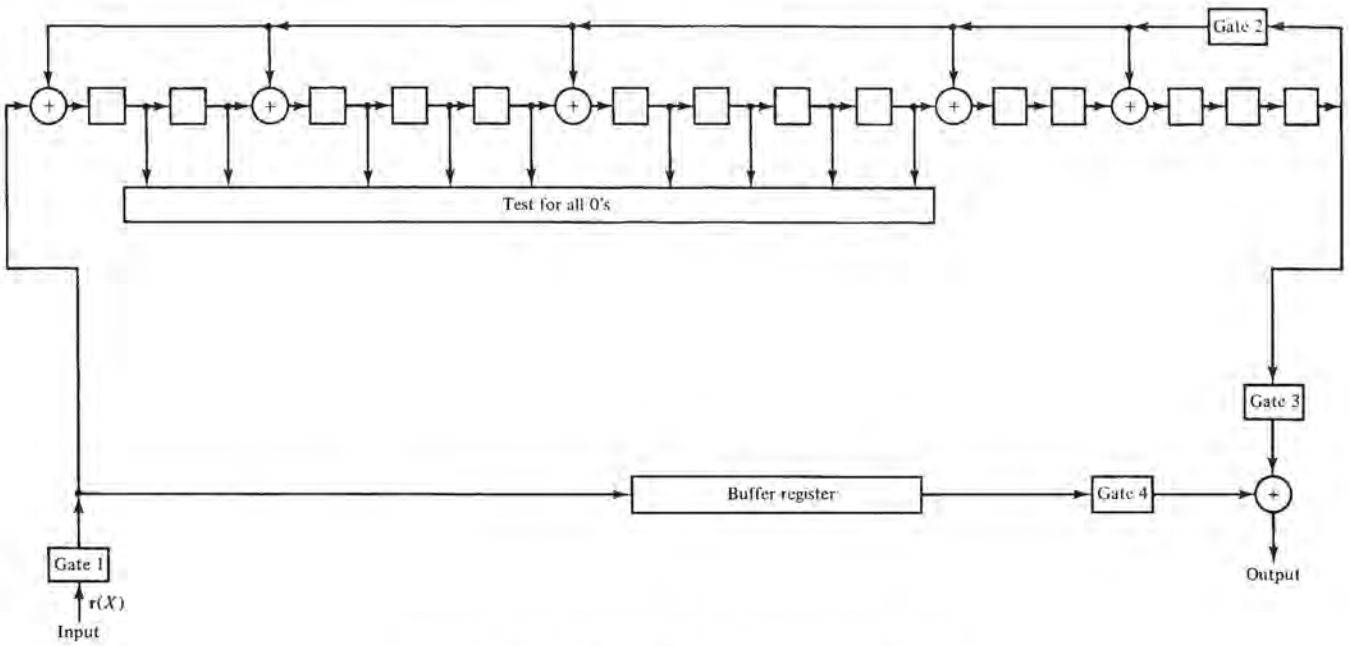


Figure 9.2 Error-trapping decoder for the (279, 265) Fire code.

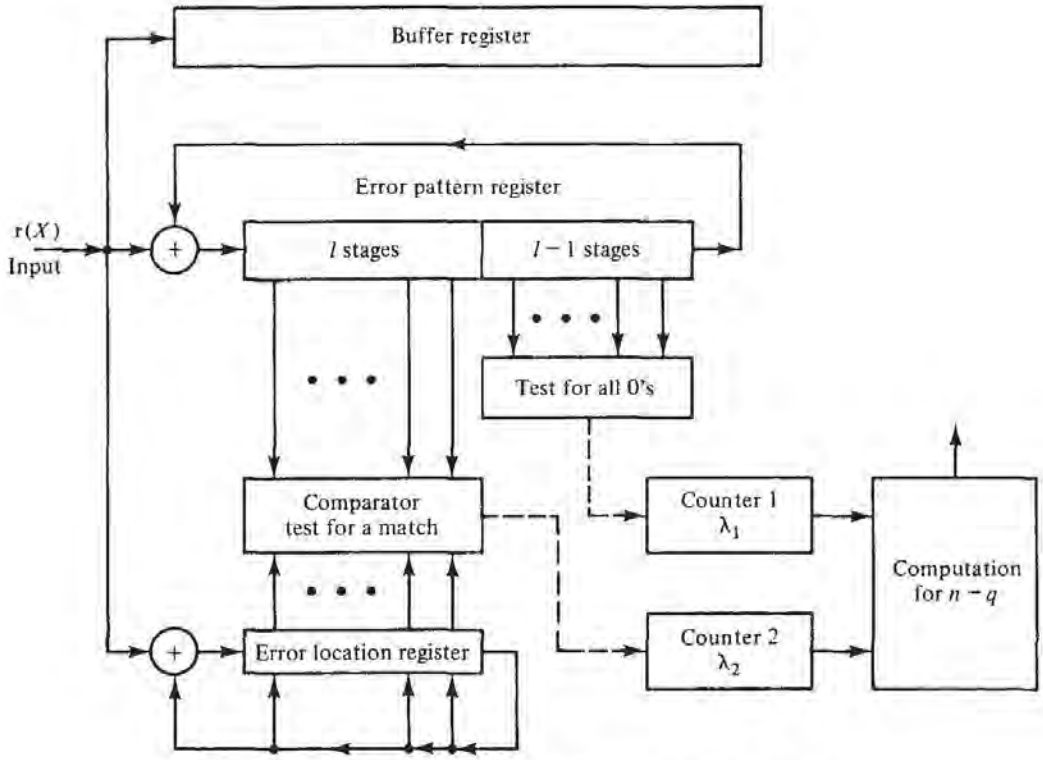


Figure 9.3 High-speed error-trapping decoder for Fire codes.

nomial $\mathbf{r}(X)$ is assumed to be error-free and is then delivered to the user. If $\mathbf{s}_1(X) = 0$ and $\mathbf{s}_2(X) \neq 0$ [or $\mathbf{s}_1(X) \neq 0$ and $\mathbf{s}_2(X) = 0$], then $\mathbf{r}(X)$ contains a detectable but uncorrectable error burst and is therefore discarded. If $\mathbf{s}_1(X) \neq 0$ and $\mathbf{s}_2(X) \neq 0$, then $\mathbf{r}(X)$ is assumed to contain a correctable error burst and the decoder starts the error correction process. The error correction process is discribed as follows:

Step 1. Shift the error-pattern register and test for zeros at the $l - 1$ high-order stages. Stop shifting as soon as the $l - 1$ high-order stages contain all zeros. The error burst is then trapped in the l low-order stages of the error-pattern register. Let λ_1 be the number of shifts performed (in counter 1). Note that no more than $2l - 2$ shifts are needed to trap the error-burst.

Step 2. Shift the error-location register until the contents in its l low-order stages match the burst pattern in the l low-order stages of the error-pattern register. Let the number of shifts be λ_2 (in counter 2). In this step, no more than $\rho - 1$ shifts are required.

Step 3. Since $2l - 1$ and ρ are relatively prime, there exists a unique non-negative integer q less than n (code length) such that the remainders resulting from dividing q by $2l - 1$ and ρ are λ_1 and λ_2, respectively. Determine the integer q by computation. Then the error burst begins at position X^{n-q} and ends at position $X^{n-q+l-1}$ of $\mathbf{r}(X)$. In the case that $q = 0$, the burst begins at position X^0 and ends at position X^{l-1} of $\mathbf{r}(X)$.

Ex.1020 / Page 284 of 305
Sandisk Technologies, Inc. et al

Step 4. Let $B(X)$ be the burst pattern trapped in the error-pattern register. Add $X^{n-q}B(X)$ to $\mathbf{r}(X)$ in the buffer register. This completes the error correction process.

If, in step 1, the $l-1$ high-order stages of the error-pattern register never contain all zeros by the time the register has been shifted $2l-2$ times, an uncorrectable error burst has been detected. In this event, the decoder stops the error-correction process.

The error-location number $n-q$ can be computed easily. Since $2l-1$ and ρ are relatively prime, there exist two integers, A_1 and A_2, such that

$$A_1(2l-1) + A_2\rho = 1.$$

The q is simply the remainder resulting from dividing

$$A_1(2l-1)\lambda_2 + A_2\rho\lambda_1$$

by n. Once A_1 and A_2 are determined, the numbers $A_1(2l-1)$ and $A_2\rho$ can be stored in the receiver permanently for use in each decoding. Therefore, computing $n-q$ needs two multiplications, one addition, one division, and one subtraction.

We note that the error-trapping decoder for Fire codes described above requires at most $2l+\rho-3$ shifts of the two syndrome registers and five arithmetic operations to carry out the error correction process. However, the error-trapping decoder described in Section 9.2 takes n shifts (cycle times) to complete the error-correction process. Since $n = \text{LCM}(2l-1, \rho)$, it is much greater than $2l+\rho-3$. Therefore, decoding speed is improved. This improvement in decoding speed is possible only when the receiver has some computation capability or computation facility is available at the receiver. Furthermore, the fast error-trapping decoder requires more logic.

Example 9.2

Consider the (279, 265) Fire code considered in Example 9.1. This code is capable of correcting any burst of length $l = 5$ or less. The fast error-trapping decoder for this code is shown in Figure 9.4. Suppose that the error burst

$$\mathbf{e}(X) = X^2 + X^3 + X^4 + X^5 + X^6$$

has occurred. It is a solid burst of length 5 starting at position $n-q = 2$. The syndrome $\mathbf{s}_1(X)$ and $\mathbf{s}_2(X)$ are remainders resulting from dividing $\mathbf{e}(X)$ by $X^{11}+1$ and $\mathbf{p}(X) = 1 + X^2 + X^5$, respectively. They are

$$\mathbf{s}_1(X) = X^2 + X^3 + X^4 + X^5 + X^6,$$
$$\mathbf{s}_2(X) = 1 + X + X^4.$$

As soon as the entire received polynomial $\mathbf{r}(X)$ has been shifted into the error-pattern and error-location registers, the contents in the two registers are $\mathbf{s}_1(X)$ and $\mathbf{s}_2(X)$. Since the four high-order stages of the error-pattern register do not contain all zeros, the error burst is not trapped in the five low-order stages. The error-pattern register starts to shift. Table 9.1 shows the contents in the error-pattern register after each shift. We see that the error burst is trapped in the five low-order stages after $\lambda_1 = 7$ shifts. Now, the error-location register begins to shift. Table 9.2 displays the contents in the error-location register after each shift. At the twenty-ninth shift, the contents in the error-location register match the contents in the five low-order stages of the error-

Ex.1020 / Page 285 of 305
Sandisk Technologies, Inc. et al

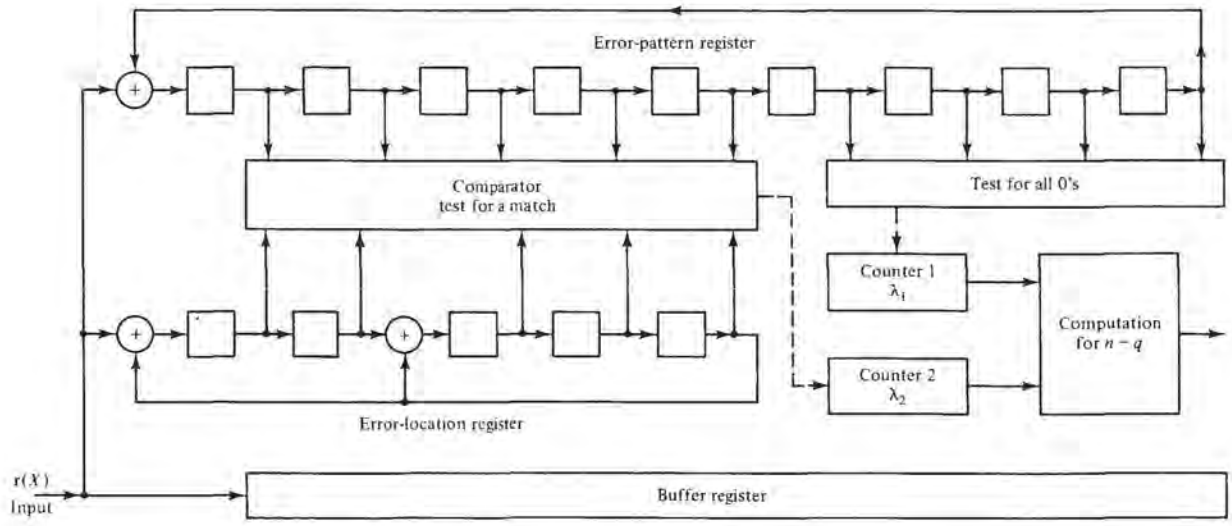


Figure 9.4 High-speed error-trapping decoder for the (279, 265) Fire Code.

Ex.1020 / Page 286 of 305
Sandisk Technologies, Inc. et al

TABLE 9.1 CONTENTS IN THE ERROR-PATTERN REGISTER OF THE DECODER SHOWN IN FIGURE 9.4 AFTER EACH SHIFT

Shift	Content
0	0 0 1 1 1 1 1 0 0
1	0 0 0 1 1 1 1 1 0
2	0 0 0 0 1 1 1 1 1
3	1 0 0 0 0 1 1 1 1
4	1 1 0 0 0 0 1 1 1
5	1 1 1 0 0 0 0 1 1
6	1 1 1 1 0 0 0 0 1
7*	1 1 1 1 1 0 0 0 0

*At the seventh shift, the contents in the four high-order stages are all zeros.

TABLE 9.2 CONTENTS IN THE ERROR-LOCATION REGISTER OF THE DECODER SHOWN IN FIGURE 9.4 AFTER EACH SHIFT

Shift	Contents	Shift	Contents
0	1 1 0 0 1	15	0 1 0 0 0
1	1 1 0 0 0	16	0 0 1 0 0
2	0 1 1 0 0	17	0 0 0 1 0
3	0 0 1 1 0	18	0 0 0 0 1
4	0 0 0 0 1	19	1 0 1 0 0
5	1 0 1 0 1	20	0 1 0 1 0
6	1 1 1 1 0	21	0 0 1 0 1
7	0 1 1 1 1	22	1 0 1 1 0
8	1 0 0 1 1	23	0 1 0 1 1
9	1 1 1 0 1	24	1 0 0 0 1
10	1 1 0 1 0	25	1 1 1 0 0
11	0 1 1 0 1	26	0 1 1 1 0
12	1 0 0 1 0	27	0 0 1 1 1
13	0 1 0 0 1	28	1 0 1 1 1
14	1 0 0 0 0	29*	1 1 1 1 1

*At the twenty-ninth shift, the contents match the burst pattern in the error-pattern register.

pattern register. Therefore, $\lambda_2 = 29$. Next we need to compute the error-location number $n - q$. First we find that

$$7 \times 9 + (-2) \times 31 = 1$$

($A_1 = 7$ and $A_2 = -2$). Then we compute

$$7 \times 9 \times 29 + (-2) \times 31 \times 7 = 1393.$$

Dividing 1393 by $n = 279$, we obtain $q = 277$. Consequently, $n - q = 2$, which is exactly the error-location number. Error correction is achieved by adding the error burst $X^2 + X^3 + X^4 + X^5 + X^6$ to the received polynomial $\mathbf{r}(X)$ in the buffer

Ex.1020 / Page 287 of 305
Sandisk Technologies, Inc. et al

register. Error-correction process takes at most $8 + 30 = 38$ cycle times. Using the decoder shown in Figure 9.2, the error-correction process takes $n = 279$ cycle times.

The fast error-trapping decoder for Fire codes was first devised by Peterson [24] and then refined by Chien [20].

The burst-correcting efficiency of a Fire code is $z = 2l/(m + 2l - 1)$. If l is chosen to be equal to m, then $z = 2m/(3m - 1)$. For large m, z is approximately 2/3. Thus, Fire codes are not very efficient with respect to the Reiger bound. However, they can be simply implemented.

A Fire code that is capable of correcting any burst of length l or less and simultaneously detecting any burst of length $d \geq l$ is generated by

$$\mathbf{g}(X) = (X^c + 1)\mathbf{p}(X),$$

where $c \geq l + d - 1$ and c is not divisible by the period ρ of $\mathbf{p}(X)$. The length of this code is the least common multiple of c and ρ.

Other Codes

Besides Fire codes, some very efficient cyclic codes and shortened cyclic codes for correcting short single bursts have been found either analytically or with the aid of a computer [7,11,14,15]. A list of these codes with their generator polynomials is given in Table 9.3. These codes and the codes derived from them by interleaving, described in the following section, are the most efficient single-burst-error-correction codes known.

TABLE 9.3 SOME BURST-CORRECTING CYCLIC AND SHORTENED CYCLIC CODES

$n - k - 2l$	Code (n, k)	Burst-correcting capability l	Generator polynomial $\mathbf{g}(X)$*
0	(7, 3)	2	35
	(15, 9)	3	171
	(15, 7)	4	721
	(15, 5)	5	2467
	(19, 11)	4	1151
	(21, 9)	6	14515
	(21, 7)	7	47343
	(21, 5)	8	214537
	(21, 3)	9	1647235
	(27, 17)	5	2671

*Generator polynomials are given in an octal representation. Each digit represents three binary digits according to the following code:

$0 \leftrightarrow 0\ 0\ 0$	$2 \leftrightarrow 0\ 1\ 0$	$4 \leftrightarrow 1\ 0\ 0$	$6 \leftrightarrow 1\ 1\ 0$
$1 \leftrightarrow 0\ 0\ 1$	$3 \leftrightarrow 0\ 1\ 1$	$5 \leftrightarrow 1\ 0\ 1$	$7 \leftrightarrow 1\ 1\ 1$

The binary digits are then the coefficients of the polynomial, with the high-order coefficients at the left. For example, the binary representation of 171 is 0 0 1 1 1 1 0 0 1, and the corresponding polynomial is $\mathbf{g}(X) = X^6 + X^5 + X^4 + X^3 + 1$.

Ex.1020 / Page 288 of 305
Sandisk Technologies, Inc. et al

TABLE 9.3 CONTINUED

$n-k-2l$	Code (n, k)	Burst-correcting capability l	Generator polynomial $g(X)$
	(34, 22)	6	15173
	(38, 24)	7	114361
	(50, 34)	8	224531
	(56, 38)	9	1505773
	(59, 39)	10	4003351
1	(15, 10)	2	65
	(21, 14)	3	171
	(21, 12)	4	1663
	(21, 10)	5	7707
	(23, 12)	5	5343
	(27, 20)	3	311
	(31, 20)	5	4673
	(38, 29)	4	1151
	(48, 37)	5	4501
	(63, 50)	6	22377
	(63, 48)	7	105437
	(63, 46)	8	730535
	(63, 44)	9	2002353
	(67, 54)	6	36365
	(96, 79)	7	114361
	(103, 88)	8	501001
2	(17, 9)	3	471
	(21, 15)	2	123
	(31, 25)	2	161
	(31, 21)	4	3551
	(35, 23)	5	13627
	(39, 27)	5	13617
	(41, 21)	9	6647133
	(51, 41)	4	3501
	(51, 35)	7	304251
	(55, 35)	9	7164555
	(57, 39)	8	1341035
	(63, 55)	3	711
	(63, 53)	4	2263
	(63, 51)	5	16447
	(63, 49)	6	61303
	(73, 63)	4	2343
	(85, 75)	4	2651
	(85, 73)	5	10131
	(105, 91)	6	70521
	(131, 119)	5	15163
	(169, 155)	6	55725
3	(51, 42)	3	1455
	(63, 56)	2	305
	(85, 76)	3	1501
	(89, 78)	4	4303
	(93, 82)	4	6137
	(121, 112)	3	1411
	(151, 136)	6	114371

Ex.1020 / Page 289 of 305
Sandisk Technologies, Inc. et al

TABLE 9.3 CONTINUED

$n-k-2l$	Code (n, k)	Burst-correcting capability l	Generator polynomial $g(X)$
	(164, 153)	4	6255
	(195, 182)	5	22475
	(217, 202)	6	120247
	(290, 277)	5	24711
4	(43, 29)	5	52225
	(91, 79)	4	10571
	(93, 83)	3	2065
	(117, 105)	4	13413
	(133, 115)	7	1254355
	(255, 245)	3	3523
	(255, 243)	4	17667
	(255, 241)	5	76305
	(255, 239)	6	301565
	(273, 261)	4	10743
	(511, 499)	4	10451
	(595, 581)	5	64655
5	(465, 454)	3	7275
	(1023, 1010)	4	22365

9.4 INTERLEAVED CODES

Given an (n, k) cyclic code, it is possible to construct a $(\lambda n, \lambda k)$ cyclic code (i.e., a code λ times as long with λ times as many information digits) by *interleaving*. This is done simply by arranging λ code vectors in the original code into λ rows of a rectangular array and then transmitting them column by column as shown in Figure 9.5. The resulting code is called an *interleaved code*. The parameter λ is referred to as the *interleaving degree*.

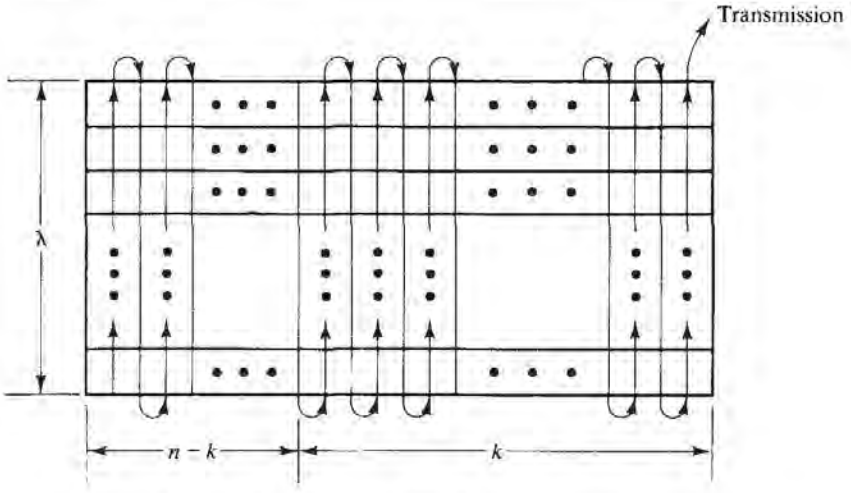


Figure 9.5 Transmission of an interleaved code.

Ex.1020 / Page 290 of 305
Sandisk Technologies, Inc. et al

Obviously, a pattern of errors can be corrected for the whole array if and only if the pattern of errors in each row is a correctable pattern for the original code. No matter where it starts, a burst of length λ will affect no more than one digit in each row. Thus, if the original code corrects single errors, the interleaved code corrects single bursts of length λ or less. If the original code corrects any single burst of length l or less, the interleaved code will correct any single burst of length λl or less. If an (n, k) code has maximum possible burst-error-correcting capability (i.e., $n - k - 2l = 0$), the interleaved $(\lambda n, \lambda k)$ code also has maximum possible burst-error-correcting capability. By interleaving short codes with maximum possible burst-error-correcting capability, it is possible to construct codes of practically any length with maximum burst-error-correcting ability. Therefore, the interleaving technique reduces the problem of searching long efficient burst-error-correcting codes to search good short codes.

The obvious way to implement an interleaved code is to set up the array and operate on rows in encoding and decoding. This is generally not the simplest implementation. The simplest implementation results from the observation that *if the original code is cyclic, the interleaved code is also cyclic.* If the generator polynomial of the original code is $\mathbf{g}(X)$, the generator polynomial for the interleaved code is $\mathbf{g}(X^\lambda)$ (see Problem 9.6). Thus, encoding and syndrome computation can be accomplished by using shift registers. It turns out that the decoder for the interleaved code can be derived from the decoder of the original code simply by replacing each register stage of the original decoder by λ stages without changing the other connections. This essentially allows the decoder circuitry to look at successive rows of the code array on successive decoder cycles. Therefore, if the decoder of the original code is simple (this is usually true for short codes), so is the decoder for the interleaved code.

The interleaving technique described above is effective not only for deriving long powerful single-burst-error-correcting codes from short optimal single-burst-error-correcting codes, but also for deriving long powerful burst-and-random-error-correcting codes from short codes.

9.5 PHASED-BURST-ERROR-CORRECTING CODES

Consider an (n, k) code whose length n is a multiple of m, say $n = \sigma m$. The σm digits of each code vector may be grouped into σ subblocks; each subblock consists of m consecutive code digits. For example, let

$$\mathbf{v} = (v_0, v_1, v_2, \ldots, v_{\sigma m-1})$$

be a code vector. Then the ith subblock consists of the following consecutive code digits:

$$v_{im}, v_{im+1}, \ldots, v_{(i+1)m-1},$$

with $0 \leq i < \sigma$. A burst of length λm or less is called a *phased burst* if and only if it is confined to λ consecutive subblocks, where λ is a positive integer less than σ. A linear code of length $n = \sigma m$ that is capable of correcting all phased error bursts confined to λ or fewer subblocks is called a *λm-phased-burst-error-correcting code.* Since a burst of length $(\lambda - 1)m + 1$, no matter where it starts, can affect at most λ subblocks, it is clear that a λm-phased-burst-error-correcting code is capable of cor-

Ex.1020 / Page 291 of 305
Sandisk Technologies, Inc. et al

recting any single burst of length $(\lambda - 1)m + 1$ or less. Thus, a λm-phased-burst-error-correcting code can be used as a $[(\lambda - 1)m + 1]$-single-burst-error-correcting code.

Burton Codes

In the following, a class of phased-burst-error-correcting cyclic codes is presented. This class of codes is similar to the class of Fire codes and was discovered by Burton [21]. Let $\mathbf{p}(X)$ be an irreducible polynomial of degree m and period ρ. Let n be the least common multiple of m and ρ. Then $n = \sigma m$. For any positive integer m, there exists an m-phased burst-error-correcting Burton code of length $n = \sigma m$ which is generated by

$$\mathbf{g}(X) = (X^m + 1)\mathbf{p}(X). \tag{9.16}$$

The number of parity-check digits of this code is $2m$. Thus, it is a $[\sigma m, (\sigma - 2)m]$ cyclic code. Each code vector consists of σ subblocks. In order to show that the Burton code generated by $\mathbf{g}(X) = (X^m + 1)\mathbf{p}(X)$ is capable of correcting all phased bursts confined to a single subblock of m digits, it is necessary and sufficient to prove that no two such bursts are in the same coset of a standard array for the code. The proof is similar to the proof given for the Fire codes and is left as an exercise (see Problem 9.7).

The decoding of a Burton code can be accomplished with the error trapping decoder as described in Section 9.2 except that the contents of the m leftmost stages of the syndrome register are tested for zero at every mth shift. If m and the period ρ of $\mathbf{p}(X)$ are relatively prime and if the receiver has some computation power, the Burton codes can be decoded with the fast error-trapping algorithm described in Section 9.3.

It is possible to interleave an m-phased-burst-error-correcting Burton code in such a way that the interleaved $(\lambda n, \lambda k)$ code is capable of correcting any phased burst that is confined to λ consecutive subblocks. To accomplish this, we arrange λ code vectors in the m-phased-burst-error-correcting code into λ rows of a rectangular array as usual. We regard a subblock of each row as a single element. Then the array consists of σ columns, each column consists of λ subblocks. The array is transmitted column by column, one subblock at a time from each row. Therefore, a code vector in the interleaved code consists of $\lambda\sigma$ subblocks. No matter where it starts, any phased-error burst confined to λ or fewer subblocks will affect no more than one subblock in each row. Thus, a phased burst of length λm will be corrected if the array is decoded on a row-by-row basis. If the interleaved code is used as a $[(\lambda - 1)m + 1]$-burst-error-correcting code, its burst-error-correcting efficiency is

$$z = \frac{2[(\lambda - 1)m + 1]}{2\lambda m} = 1 - \frac{1}{\lambda}\left(\frac{m - 1}{m}\right).$$

As the interleaving degree λ becomes large, the burst-error-correcting efficiency of a Burton code approached 1. Thus, by interleaving the Burton codes, we obtain a class of asymptotically optimal burst-error-correcting codes.

The obvious way to implement an interleaved Burton code is to set up the code array and operate on rows in encoding and decoding. Thus, the encoder of the inter-

Ex.1020 / Page 292 of 305
Sandisk Technologies, Inc. et al

leaved code consists of the encoder of the original code and a buffer for the storage of the row vectors of the code array; the decoder consists of the decoder of the original code and a buffer for the storage of the received code array. Of course, the interleaved code can be decoded with the error-trapping decoder of Figure 9.1, where the contents of the λm leftmost stages are tested for zeros at every mth shift.

9.6 BURST-AND-RANDOM-ERROR-CORRECTING CODES

On many communication channels, errors occur neither independently at random nor in well-defined single bursts, but occur in a mixed manner. To combat these mixed errors, the random-error-correcting codes or single-burst-error-correcting codes will be either inefficient or inadequate. Consequently, it is desirable to design codes that are capable of correcting random errors and/or single or multiple bursts. There are several methods of constructing codes for the correction of random errors as well as burst errors. The most effective method is the interleaving technique as described in Section 9.4. By interleaving a t-random-error-correcting (n, k) code to degree λ, we obtain a $(\lambda n, \lambda k)$ code which is capable of correcting any combination of t bursts of length λ or less. Several other methods of deriving new codes from known codes for simultaneously correcting random errors and burst errors are discussed in this section.

Product Codes

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 code word is a rectangular array of n_1 columns and n_2 rows in which every row is a code vector in C_1 and every column is a code vector in C_2, as shown in Figure 9.6. 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

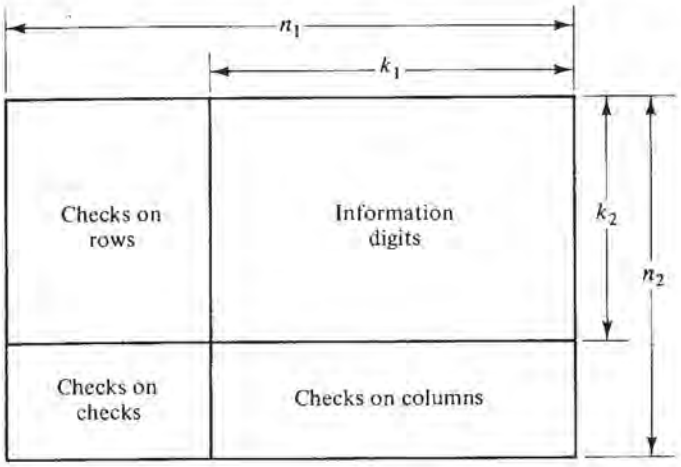


Figure 9.6 Code array for the product code $C_1 \times C_2$.

Ex.1020 / Page 293 of 305
Sandisk Technologies, Inc. et al

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 would yield the same $(n_1 - k_1) \times (n_2 - k_2)$ check digits (see Problem 9.17), and it is possible to have all row code vectors in C_1 and all column code vectors in C_2 simultaneously. Therefore, for encoding the product code $C_1 \times C_2$, we may first encode the k_2 rows of the information array based on the parity-check rules for C_1 and then encode the n_1 resulting columns based on the rules for C_2, or vice versa.

If the code C_1 has minimum weight d_1 and the code C_2 has minimum weight d_2, the minimum weight of the product code is exactly $d_1 d_2$. A minimum-weight code vector in the product code is formed by (1) choosing a minimum-weight code vector in C_1 and a minimum-weight code vector in C_2; and (2) forming an array in which all columns corresponding to zeros in the code vector from C_1 are zeros and all columns corresponding to ones in the code vector C_1 are the minimum-weight code vector chosen from C_2.

It is not easy to characterize the correctable error patterns for the product code; this depends upon how the correction is actually done. One method involves using the correction first 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 by decoding rows, columns, then columns and rows again. This, of course, increases the decoding delay.

The product code is capable of correcting any combination of $\lfloor (d_1 d_2 - 1)/2 \rfloor$ errors, but the method described above 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.

Let l_1 and l_2 be the burst-error-correcting capabilities of code C_1 and the code C_2, respectively. The burst-error-correcting capability of the product code of C_1 and C_2 can be analyzed as follows. Suppose that a code array is transmitted row by row and that, at the output of the channel, the received digits are rearranged back into an array row by row. No matter where it starts, any existing error burst of length $n_1 l_2$ or less will affect no more than $l_2 + 1$ consecutive rows; when the received digits are rearranged back into an array, each column is at most affected by a burst of length l_2. Now if the array is decoded on a column-by-column basis, the burst will be corrected. Therefore, the burst-error-correcting capability of the product code is at least $n_1 l_2$. Suppose that a code array is transmitted on a column-by-column basis and decoded on a row-by-row basis. By a similar argument, it is possible to show that any error burst of length $n_2 l_1$ or less can be corrected. Thus, the burst-correcting capability of the product code is at least $n_2 l_1$. Consequently, we may conclude that the burst-error-correcting capability l of the product code is at least $\max \{n_1 l_2, n_2 l_1\}$.

So far, we have considered a product code for either random-error correction or burst-error correction. However, a product code can be used for simultaneous random-error correction and burst-error correction. Let d_1 and d_2 be the minimum

Ex.1020 / Page 294 of 305
Sandisk Technologies, Inc. et al

distances of codes C_1 and C_2, respectively. Then it is possible to show that the product code of C_1 and C_2 is capable of correcting any combination of $t = \lfloor (d_1 d_2 - 1)/2 \rfloor$ or fewer random errors and simultaneously correcting any error burst of length $l = \max(n_1 t_2, n_2 t_1)$ or less, where $t_1 = \lfloor (d_1 - 1)/2 \rfloor$ and $t_2 = \lfloor (d_2 - 1)/2 \rfloor$ [26,27]. To prove this assertion, it is sufficient to show that an error burst of length l or less and a random-error pattern of t or fewer errors cannot be in the same coset of a standard array for the product code. Suppose that $n_1 t_2 \geq n_2 t_1$. Then $l = n_1 t_2$. Consider a burst of length $n_1 t_2$ or less. When this vector is arranged as an array of n_2 rows and n_1 columns, each column contains at most t_2 errors. Suppose that this burst and some random-error pattern of t or fewer errors in the same coset of the product code. Then the sum of these two error patterns (in array form) is a code array in the product code. As a result, each column of the sum array must either have no nonzero components or have at least d_2 nonzero components. Each nonzero column of the sum array must be composed of at least $d_2 - t_2$ errors from the random-error pattern and at most t_2 errors from the burst-error pattern. Since there are at most t random errors, these errors can be distributed among at most $\lfloor t/(d_2 - t_2) \rfloor$ columns. Thus, the sum array contains at most $\lfloor t/(d_2 - t_2) \rfloor t_2 + t$ nonzero components. However,

$$\left\lfloor \frac{t}{d_2 - t_2} \right\rfloor t_2 + t \leq t\left(\frac{t_2}{d_2 - t_2} - 1\right) < 2t.$$

Hence, the sum array contains fewer than $2t < d_1 d_2$ nonzero components and cannot be a code array in the product code. This contradiction implies that a burst of length $l = n_1 t_2$ or less and a random-error pattern of t or fewer errors cannot be in the same coset of a standard array for the product code. Therefore, they can both be used as coset leaders and are correctable error patterns. If $n_2 t_1 > n_1 t_2$, then $l = n_2 t_1$. The same argument can be applied to rows instead of columns of the sum array.

The obvious way to implement a product code is to set up the code array and operate on rows and then 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 are transmitted in a proper order [26]. Start with the upper right corner and move down and *to the left* on a 45° diagonal as shown in Figure 9.7. When we reach the end of a column, move to the top of the next column. When we reach the end of a row, move to the rightmost digit of the next row.

Since 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 [26,28] that the generator polynomial $\mathbf{g}(X)$ of the cyclic product code of C_1 and C_2 is the greatest common divisor of $X^{n_1 n_2} - 1$ and $\mathbf{g}_1(X^{bn_2})\mathbf{g}_2(X^{an_1})$, that is,

$$\mathbf{g}(X) = \text{GCD}\,[X^{n_1 n_2} - 1, \mathbf{g}_1(X^{bn_2})\mathbf{g}_2(X^{an_1})] \tag{9.17}$$

Ex.1020 / Page 295 of 305
Sandisk Technologies, Inc. et al

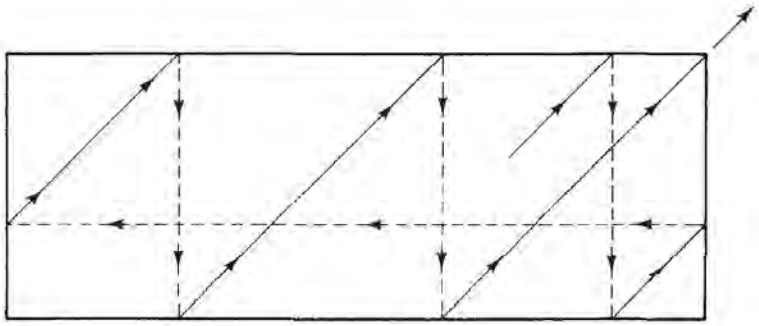


Figure 9.7 Transmission of a cyclic product code.

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,

$$\mathbf{h}(X) = \text{GCD}\,[\mathbf{h}_1(X^{bn_2}), \mathbf{h}_2(X^{an_1})]. \tag{9.18}$$

Suppose that the cyclic code C_1 has random-error-correcting capability t_1 and burst-error-correcting capability l_1 and the cyclic code C_2 has random-error-correcting capability t_2 and burst-error-correcting capability l_2. Then the burst-error-correcting capability l of the cyclic product code of C_1 and C_2 is at least equal to max $(n_1t_2 + l_1, n_2t_1 + l_2)$, [26], that is,

$$l \geq \max\,(n_1t_2 + l_1, n_2t_1 + l_2). \tag{9.19}$$

This can be shown as follows. Suppose that an error burst of length $n_2t_1 + l_2$ or less occurred during the transmission of a code array. When the received digits are rearranged back into an array, all except l_2 adjacent rows will contain t_1 or fewer errors. Each of these l_2 adjacent rows will contain $t_1 + 1$ or fewer errors. If the rows are decoded first, these l_2 adjacent rows may contain errors after the row decoding. Therefore, after row decoding, each column of the array contains an error burst of length at most l_2. Since the column code C_2 is capable of correcting any error burst of length l_2 or less, all the remaining errors in the array will be corrected by column decoding. By a similar argument, any error burst of length $n_1t_2 + l_1$ or less will be corrected if the column decoding is performed before the row decoding. Therefore, we obtain the result as stated by (9.19).

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 retangular array. Thus, the decoder can decode each of the row (or column) code vectors separately and then decode each of the column (or row) code vectors. Alternatively, in the transmitted code vector, the set of n_1 digits formed by selecting every (n_2)th digit are the n_1 digits of a code vector of C_1 permuted in a fixed way. They can be permuted back to their original form and corrected in a Meggitt-type decoder. The digits in the permuted form are a code vector in a related code and can be decoded directly in this form in a Meggitt-type decoder. Similarly, correction for the column code C_2 can be done by selecting every (n_1)th digit from the large code vector. Thus, the total equipment required is roughly that required to decode the two individual codes.

Ex.1020 / Page 296 of 305
Sandisk Technologies, Inc. et al

There is room for a good deal of engineering ingenuity in divising decoding procedures for cyclic product codes [28–35].

Code Derived from Reed–Solomon Codes

In Chapter 2 it was pointed out that any element β in the Galois field GF(2^m) can be expressed uniquely as a sum of $1, \alpha, \alpha^2, \ldots, \alpha^{m-1}$ in the following form:

$$\beta = a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{m-1}\alpha^{m-1},$$

where α is a primitive element in GF(2^m) and $a_i = 0$ or 1. Thus, the correspondence between β and $(a_0, a_1, \ldots, a_{m-1})$ is one-to-one. We shall call the m-tuple $(a_0, a_1, \ldots, a_{m-1})$ an *m-bit byte* representation of β.

Consider a t-error-correcting Reed–Solomon code with code symbols from GF(2^m). If each symbol is represented by its corresponding m-bit byte, we obtain a binary linear code with the following parameters:

$$n = m(2^m - 1),$$
$$n - k = 2mt.$$

This code is capable of correcting any error pattern that affects t or fewer m-bit bytes. It is immaterial whether a byte has one error or whether all the m bits are in error; it is counted as one byte error. This can be seen as follows. At the channel output, the binary received vector is divided into $2^m - 1$ bytes; each byte is transformed back into a symbol in GF(2^m). Thus, if an error pattern affects t or fewer bytes, it affects t or fewer symbols in a Reed–Solomon code. Obviously, the error pattern can be corrected by the decoding method described in Sections 6.2 and 6.5. We shall call this binary code a *t-byte-correcting code*. Actually, this code is a multiple-phased-burst-error-correcting code.

Binary codes derived from Reed–Solomon codes are more effective against clustered errors than random errors since clustered errors usually involve several errors per byte and thus relatively few byte errors. For example, since a burst of length $3m + 1$ cannot affect more than 4 bytes, a 4-byte-correcting code can correct any single burst of length $3m + 1$ or less. It can also simultaneously correct any combination of two bursts of length $m + 1$ or less because each such burst can affect no more than 2 bytes. At the same time, it can correct any combination of four or fewer random errors. In general, a t-byte-correcting binary Reed–Solomon code is capable of correcting any combination of

$$\lambda = \frac{t}{1 + \lfloor (l + m - 2)/m \rfloor}$$

or fewer bursts of length l [or correcting any single burst of length $(t - 1)m + 1$ or less]. Simultaneously, it corrects any combination of t or fewer random errors.

Concatenated Codes

Concatenation is a specific method of constructing long codes from shorter codes. This method was first proposed by Forney [36] as a means of constructing long block codes which can be decoded without the equipment complexity usually required in the use of long codes. A simple concatenated code is formed from two codes: an

Ex.1020 / Page 297 of 305
Sandisk Technologies, Inc. et al

(n_1, k_1) binary code C_1 and an (n_2, k_2) nonbinary code C_2 with symbols form GF(2^{k_1}). The symbols of C_2 are represented by their corresponding bytes of k_1 binary symbols. Usually, a Reed–Solomon code is used for C_2. Encoding consists of two steps, as shown in Figure 9.8. First, the k_1k_2 binary information digits are divided into k_2 bytes of k_1 information digits each. These k_2 bytes are encoded according to the rules for C_2 to form an n_2-byte code vector. Second, each k_1-digit byte is encoded into a code vector in C_1, resulting in a string of n_2 code vectors of C_1, a total of n_2n_1 digits. These digits are then transmitted, one C_1 code vector at a time, in succession. Thus, the resultant code is an (n_1n_2, k_1k_2) binary linear code. The component codes C_1 and C_2 are called the *inner* and the *outer* codes, respectively. If the minimum distances of the inner and outer codes are d_1 and d_2, respectively, the minimum distance of their concatenation is at least d_1d_2 (see Problem 9.18).

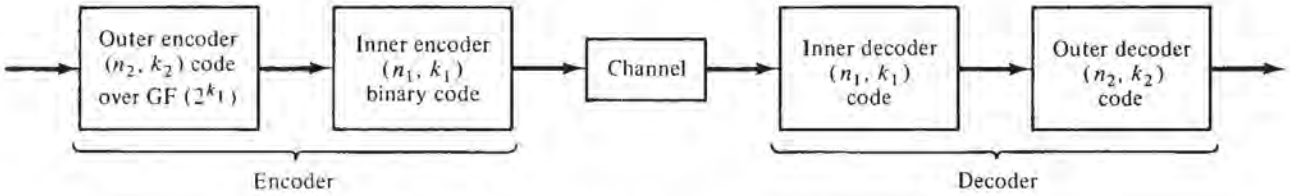


Figure 9.8 Communication system using a concatenated code.

Decoding of the concatenated code of C_1 and C_2 is also done in two steps, as shown in Figure 9.8. First, decoding is done for each C_1 code vector as it arrives, and the check digits are removed, leaving a sequence of n_2k_1-digit bytes. These bytes are then decoded according to the method for C_2, to leave the final corrected message. Decoding implementation is the straightforward combination of the implementations for codes C_1 and C_2, and the amount of hardware required is roughly that required by both codes.

Concatenated codes are effective against a mixture of random errors and bursts, and the pattern of bytes not correctable by the C_1 code must form a correctable error pattern for C_2 if the concatenated code is to correct the error pattern. Scattered random errors are corrected by C_1. Bursts may affect relatively few bytes, but probably so badly that C_1 cannot correct them. These few bytes can then be corrected by C_2.

Example 9.3

Consider the concatenation of the (15, 11) Reed–Solomon code with symbols from GF(2^4) and the (7, 4) binary Hamming code. Each code symbol of the Reed–Solomon code is represented by a byte of four binary digits, as in Table 2.8. Then each 4-bit byte is encoded into a code vector in the (7, 4) Hamming code. The resultant concatenated code is a (105, 44) binary code. Since the minimum distance of the (7, 4) Hamming code is 3 and the minimum distance of the (15, 11) Reed–Solomon code is 5, the concatenated code has a minimum distance at least 15. If the code is decoded in two steps, first the inner code and then the outer code, the decoder is capable of correcting any error pattern such that the number of inner code vectors with more than a single error is less than 3.

It is possible to construct a concatenated code from a single outer code and *multiple* inner codes. For example, one may use an (n_2, k_2) code C_2 with symbols

Ex.1020 / Page 298 of 305
Sandisk Technologies, Inc. et al

from GF(2^{k_1}) as the outer code and n_2 (n_1, k_1) binary codes, $C_1^{(1)}, C_1^{(2)}, \ldots, C_1^{(n_2)}$, as inner codes. Again, the encoding consists of two steps. First the k_1k_2 information digits are divided into k_2 bytes of k_1 digits each. These k_2 bytes are encoded according to the rules for C_2 to form an n_2-byte code word,

$$(a_0, a_1, \ldots, a_{n_2-1}),$$

where each byte a_i is regarded as an element in GF(2^{k_1}). Second, the ith byte a_i for $0 \leq i < n_2$ is encoded into a code vector in the ith inner code $C_1^{(i)}$. The overall encoding again results in an (n_1n_2, k_1k_2) concatenated code. By concatenating one outer code with multiple inner codes, Justesen [37] was able to construct a class of asymptotically good concatenated codes.

Another interesting generalization of concatenated codes is due to Hirasawa et al. [38]. The outer code of this generalization is formed by interleaving λ codes with symbols from GF(2^k). The jth component of the outer code is an (n, k_j) code with minimum distance d_j and $n \leq 2^k$. Only one inner code is used; it is an $(N, \lambda k)$ binary code. The resultant concatenated code is an (n_0, k_0) code with $n_0 = nN$ and $k_0 = k(k_1 + k_2 + \cdots + k_\lambda)$.

9.7 MODIFIED FIRE CODES FOR SIMULTANEOUS CORRECTION OF BURST AND RANDOM ERRORS

Let β be an element of order n in the Galois field GF(2^m). It follows from Theorem 2.5 that n is a factor of $2^m - 1$. Let $\phi(X)$ be the minimal polynomial of β. The period of $\phi(X)$ is n. The degree of $\phi(X)$, m_0, is either equal to m or a factor of m. Suppose that n has a proper factor b such that

$$\frac{b+1}{2} \leq m_0.$$

Let $n = a \cdot b$. Then

$$X^n + 1 = (X^b + 1)(1 + X^b + X^{2b} + \cdots + X^{(a-1)b}).$$

Since the order of β is n, and $b < n$, β cannot be a root of $X^b + 1$ and must be a root of $1 + X^b + X^{2b} + \cdots + X^{(a-1)b}$. Therefore, $\phi(X)$ divides $1 + X^b + X^{2b} + \cdots + X^{(a-1)b}$. The code generated by

$$\mathbf{g}_1(X) = (X^b + 1)\phi(X) \tag{9.20}$$

is a Fire code C_1 of length n which is capable of correcting any error-burst of length $(b + 1)/2$ or less.

Let $\mathbf{g}_2(X)$ be the generator polynomial of a cyclic code C_2 of length n which is capable of correcting t or fewer random errors. Clearly, $\mathbf{g}_2(X)$ is a factor of $X^n + 1$. Let $\mathbf{g}(X)$ be the least common multiple of $\mathbf{g}_1(X)$ and $\mathbf{g}_2(X)$:

$$\mathbf{g}(X) = \text{LCM}\,\{\mathbf{g}_1(X), \mathbf{g}_2(X)\}. \tag{9.21}$$

Clearly $\mathbf{g}(X)$ divides $X^n + 1$ and can be expressed in the form

$$\mathbf{g}(X) = (X^b + 1)\mathbf{g}_0(X), \tag{9.22}$$

Ex.1020 / Page 299 of 305
Sandisk Technologies, Inc. et al

where $\mathbf{g}_0(X)$ is a factor of $1 + X^b - X^{2b} + \cdots + X^{(a-1)b}$. Now, we consider the cyclic code C of length n generated by $\mathbf{g}(X)$. This code C is a *subcode* of both the Fire code C_1 generated by $\mathbf{g}_1(X) = (X^b + 1)\phi(X)$ and the t-error-correcting code C_2 generated by $\mathbf{g}_2(X)$. Since C is a subcode of the Fire code C_1, C is capable of correcting any single error burst of length $(b + 1)/2$ or less. Since C is a subcode of the t-error-correcting code C_2, it is capable of correcting any combination of t or fewer random errors. Since $\mathbf{g}(X)$ has $(X + 1)$ as a factor, the minimum distance of C is even and is at least $2t + 2$. It is possible to show that C is capable of correcting any single error burst of length $(b + 1)/2$ or less *as well as* any combination of t or fewer random errors. To show this, it is necessary and sufficient to prove that a burst of length $(b + 1)/2$ or less and an error pattern of weight t or less cannot be in the same coset of C except that they are identical (see Problem 9.19).

Example 9.4

Let α be a primitive element of the Galois field GF(2^6). The order of α is $2^6 - 1 = 63$ and the minimal polynomial of α is

$$\phi(X) = 1 + X + X^6.$$

The integer 63 can be factored as follows: $63 = 7 \cdot 9$. Thus, we have

$$X^{63} + 1 = (X^9 + 1)(1 + X^9 + X^{18} + X^{27} + X^{36} + X^{45} + X^{54}).$$

The code generated by the polynomial

$$\mathbf{g}_1(X) = (X^9 + 1)(1 + X + X^6)$$

is a Fire code of length 63 which is capable of correcting any single error burst of length 5 or less. Let $\mathbf{g}_2(X)$ be the generator polynomial of the double-error-correcting BCH code of length 63. From Table 6.4 we find that

$$\mathbf{g}_2(X) = (1 + X + X^6)(1 + X + X^2 + X^4 + X^6).$$

Note that both factors of $\mathbf{g}_2(X)$ are factors of $1 + X^9 + X^{18} + X^{27} + X^{36} + X^{45} + X^{54}$. The least common multiple of $\mathbf{g}_1(X)$ and $\mathbf{g}_2(X)$ is

$$\mathbf{g}(X) = (X^9 + 1)(1 + X + X^6)(1 + X + X^2 + X^4 + X^6).$$

Hence, $\mathbf{g}(X)$ generates a (63, 42) cyclic code which is a subcode of both the Fire code generated by $\mathbf{g}_1(X) = (X^9 + 1)(1 + X + X^6)$ and the double-error-correcting BCH code generated by $\mathbf{g}_2(X) = (1 + X + X^6)(1 + X + X^2 + X^4 + X^6)$. Therefore, it is capable of correcting any single error burst of length 5 or less as well as any combination of two or fewer random errors.

Decoding of the codes defined by (9.21) can be implemented in a straightforward manner as shown in Figure 9.9, where the two decoders operate in parallel. Decoder I is an error-trapping decoder which is implemented based on the Fire code generated by $\mathbf{g}_1(X) = (X^b + 1)\phi(X)$; and decoder II is implemented based on the random-error-correcting code generated by $\mathbf{g}_2(X)$. The received polynomial $\mathbf{r}(X)$ is shifted into both decoders simultaneously. Both decoders attempt to decode $\mathbf{r}(X)$. The error-trapping decoder gives a decoded message only if the error pattern is either a burst of length $(b + 1)/2$ or less or an undetectable burst. The random-error decoder gives a decoded message only if the error pattern either contains t or fewer errors or is an undetectable error pattern. The only time when both decoders will provide decoded

Ex.1020 / Page 300 of 305
Sandisk Technologies, Inc. et al

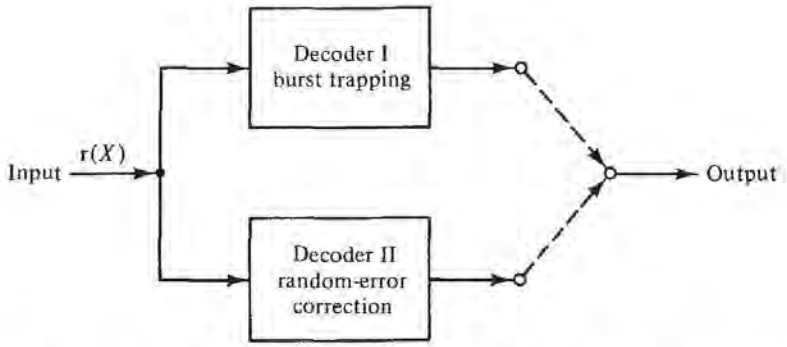


Figure 9.9 Parallel decoding for simultaneous correction of burst and random errors.

messages simultaneously is when the error pattern is in a coset with a coset leader that is a burst of length $\leq (b + 1)/2$ and of weight $\leq t$. In this case, the decoded messages from the two decoders are identical. If both decoders fail to decode $\mathbf{r}(X)$, errors are detected.

Example 9.5

In Example 9.4, let us choose $\mathbf{g}_2(X)$ as the generator polynomial of the (1, 3)th-order twofold (63, 45) EG code. From Example 8.9 we find that

$$\mathbf{g}_2(X) = (1 + X + X^6)(1 + X + X^2 + X^4 + X^6)(1 + X + X^2 + X^5 + X^6).$$

Then the least common multiple of $\mathbf{g}_1(X) = (X^9 + 1)(1 + X + X^6)$ and $\mathbf{g}_2(X)$ is

$$\mathbf{g}(X) = (X^9 + 1)(1 + X + X^6)(1 + X + X^2 + X^4 + X^6)(1 + X + X^2 + X^5 + X^6).$$

Hence, $\mathbf{g}(X)$ generates a (63, 36) cyclic code which is capable of correcting any single error burst of length 5 or less as well as any three or fewer random errors with majority-logic decoding.

There is a (63, 36) BCH code which is capable of correcting any combination of five or fewer errors. Clearly, this BCH code is more powerful than the (63, 36) code above. However, decoding for the (63, 36) BCH code is more complex than the decoding of the (63, 36) code above.

By combining Fire codes and BCH codes and with the aid of a computer, Hsu et al. have constructed several classes of shortened cyclic codes which are capable of correcting burst errors as well as random errors [39]. Other works on constructing burst-and-random error-correcting block codes can be found in References 11, 22, and 39 to 42.

PROBLEMS

9.1. Show that if an (n, k) cyclic code is designed to correct all burst errors of length l or less and simultaneously to detect all burst errors of length $d \geq l$ or less, the number of parity-check digits of the code must be at least $l + d$.

9.2. Devise an error-trapping decoder for an l-burst-error-correcting cyclic code. The received polynomial is shifted into the syndrome register from the right end. Describe the decoding operation of your decoder.

Ex.1020 / Page 301 of 305
Sandisk Technologies, Inc. et al

9.3. The polynomial $\mathbf{p}(X) = 1 + X + X^4$ is a primitive polynomial over GF(2). Find the generator polynomial of a Fire code that is capable of correcting any single error burst of length 4 or less. What is the length of this code? Devise a simple error-trapping decoder for this code.

9.4. Devise a high-speed error-trapping decoder for the Fire code constructed in Problem 9.3. Describe the decoding operation.

9.5. Use a code from Table 9.3 to derive a new code with burst-error-correcting capability $l = 51$, length $n = 255$, and burst-error-correcting efficiency $z = 1$. Construct a decoder for this new code.

9.6. Let $\mathbf{g}(X)$ be the generator polynomial of an (n, k) cyclic code. Interleave this code to a degree λ. The resultant code is a $(\lambda n, \lambda k)$ linear code. Show that this interleaved code is cyclic and its generator polynomial is $\mathbf{g}(X^\lambda)$.

9.7. Show that the Burton code generated by $\mathbf{g}(X) = (X^m + 1)\mathbf{p}(X)$, where $\mathbf{p}(X)$ is an irreducible polynomial of degree m, is capable of correcting all phased bursts confined to a single subblock of m digits.

9.8. Let $m = 5$. Construct a Burton code that is capable of correcting any phased burst confined to a single subblock of five digits. Suppose that this code is interleaved to a degree $\lambda = 6$. What are the length, the number of parity-check digits, and the burst-error-correcting capability of this interleaved code?

9.9. Interleave the (164, 153) code in Table 9.3 to a degree $\lambda = 6$. Compare this interleaved code with the interleaved Burton code of Problem 9.8. Which code is more efficient?

9.10. Interleave the (15, 7) BCH code to a degree 7. Discuss the error-correcting capability of this interleaved code. Devise a decoder for this code and describe the decoding operation.

9.11. Let C_1 be the (3, 1) cyclic code generated by $\mathbf{g}_1(X) = 1 + X + X^2$ and let C_2 be the (7, 3) maximum-length code generated by $\mathbf{g}_2(X) = 1 + X + X^2 + X^4$. Find the generator and parity-check polynomials of the cyclic product code of C_1 and C_2. What is the minimum distance of this product code? Discuss its error-correcting capability.

9.12. In Problem 9.11, both codes C_1 and C_2 are completely orthogonalizable in one step. Show that the product of these two codes is also completely orthogonalizable in one step.

9.13. 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 orthogonalized 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 [28].)

9.14. Consider the (31, 15) Reed–Solomon code with symbols from GF(2^5). Use Table 8.8 to find its generator polynomial. Convert this RS code to a binary code. Discuss the error-correcting capability of the binary RS code.

9.15. Suppose that the Fire code constructed in Problem 9.3 is shortened by deleting the 15 high-order message digits. Devise a decoder for the shortened code such that the 15 extra shifts of the syndrome register after the received vector has entered can be avoided.

9.16. Find a modified Fire code of length 63 that is capable of correcting any single burst of

Ex.1020 / Page 302 of 305
Sandisk Technologies, Inc. et al

length 4 or less as well as any combination of two or fewer random errors. Determine its generator polynomial.

9.17. Show that the digits for checking the parity-check digits of a product code array shown in Figure 9.6 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.

9.18. Prove that the concatenation of an (n_1, k_1) inner code with minimum distance d_1 and an (n_2, k_2) outer code with minimum distance d_2 has minimum distance at least $d_1 d_2$.

9.19. Consider the modified Fire code C generated by $g(X)$ of (9.21). Show that a burst of length $(b + 1)/2$ or less and error pattern of weight t or less cannot be in the same coset.

REFERENCES

1. N. Abramson, "A Class of Systematic Codes for Non-independent Errors," *IRE Trans. Inf. Theory*, IT-4(4), pp. 150–157, December 1959.
2. N. Abramson and B. Elspas, "Double-Error-Correcting Coders and Decoders for Non-independent Binary Errors," presented at the UNESCO Inf. Process. Conf., Paris, 1959.
3. P. Fire, "A Class of Multiple-Error-Correcting Binary Codes for Non-independent Errors," Sylvania Report No. RSL-E-2, Sylvania Electronic Defense Laboratory, Reconnaissance Systems Division, Mountain View, Calif., March 1959.
4. C. M. Melas, "A New Group of Codes for Correction of Dependent Errors in Data Transmission," *IBM J. Res. Dev.*, 4, pp. 58–64, January 1960.
5. S. H. Reiger, "Codes for the Correction of 'Clustered' Errors," *IRE Trans. Inf. Theory*, IT-6, pp. 16–21, March 1960.
6. B. Elspas, "A Note on Binary Adjacent-Error-Correcting Codes," *IRE Trans. Inf. Theory*, IT-6, pp. 13–15, March 1960.
7. B. Elspas and R. A. Short, "A Note on Optimum Burst-Error-Correction Codes," *IRE Trans. Inf. Theory*, IT-8, pp. 39–42, January 1962.
8. J. E. Meggitt, "Error Correcting Codes for Correcting Bursts of Errors," *IBM J. Res. Dev.*, 4, pp. 329–334, July 1960.
9. J. E. Meggitt, "Error Correcting Codes for Correcting Bursts of Errors," *Trans. AIEE*, 80, pp. 708–711, January 1961.
10. C. R. Foulk, "Some Properties of Maximally-Efficient Cyclic Burst-Correcting Codes and Results of a Computer Search for Such Codes," File No. 375, Digital Computer Lab., University of Illinois, Urbana, Ill., June 12, 1961.
11. J. J. Stone, "Multiple Burst Error Correction," *Inf. Control*, 4, pp. 324–331, December 1961.
12. L. H. Zetterburg, "Cyclic Codes from Irreducible Polynomials for Correction of Multiple Errors," *IEEE Trans. Inf. Theory*, IT-8, pp. 13–21, January 1962.
13. T. Kasami, "Cyclic Codes for Burst-Error-Correction," *J. Inst. Elec. Comm. Eng. Jap.*, 45, pp. 9–16, January 1962.
14. T. Kasami, "Optimum Shortened Cyclic Codes for Burst-Error-Correction," *IEEE Trans. Inf. Theory*, IT-9, pp. 105–109, April 1963.

Ex.1020 / Page 303 of 305
Sandisk Technologies, Inc. et al

15. T. Kasami and S. Matoba, "Some Efficient Shortened Cyclic Codes for Burst-Error-Correction," *IEEE Trans. Inf. Theory*, IT-10, pp. 252–253, July 1964.
16. A. J. Gross, "Binary Group Codes Which Correct in Bursts of Three or Less for Odd Redundancy," *IEEE Trans. Inf. Theory*, IT-8, pp. 356–359, October 1962.
17. A. J. Gross, "A Note on Some Binary Group Codes Which Correct Errors in Bursts of Four or Less," *IRE Trans. Inf. Theory*, IT-8, p. 384, October 1962.
18. A. J. Gross, "Augmented Bose-Chaudhuri Codes Which Correct Single Bursts of Errors," *IEEE Trans. Inf. Theory*, IT-9, p. 121, April 1963.
19. E. Gorog, "Some New Classes of Cyclic Codes Used for Burst Error Correction," *IBM J. Res. Dev.*, 7, pp. 102–111, 1963.
20. R. T. Chien, "Burst-Correction Codes with High-Speed Decoding," *IEEE Trans. Inf. Theory*, IT-1, pp. 109–113, January 1969.
21. H. O. Burton, "A Class of Asymptotically Optimal Burst Correcting Block Codes," presented at the ICCC, Boulder, Colo., June 1969.
22. S. E. Tavares and S. G. S. Shiva, "Detecting and Correction Multiple Bursts for Binary Cyclic Codes," *IEEE Trans. Inf. Theory*, IT-16, pp. 643–644, 1970.
23. R. G. Gallager, *Information Theory and Reliable Communication*, Wiley, New York, 1968.
24. W. W. Peterson, *Error-Correcting Codes*, MIT Press, Cambridge, Mass., 1961.
25. P. Elias, "Error-Free Coding," *IRE Trans. Inf. Theory*, PGIT-4, pp. 29–37, September 1954.
26. H. O. Burton and E. J. Weldon, Jr., "Cyclic Product Codes," *IEEE Trans. Inf. Theory*, IT-11, pp. 433–440, July 1965.
27. W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2nd ed., MIT Press, Cambridge, Mass., 1970.
28. S. Lin and E. J. Weldon, Jr., "Further Results on Cyclic Product Codes," *IEEE Trans. Inf. Theory*, IT-6(4), pp. 452–459, July 1970.
29. N. M. Abramson, "Cascade Decoding of Cyclic Product Codes," *IEEE Trans. Commun. Technol.*, COM-16, pp. 398–402, 1968.
30. W. C. Gore, "Further Results on Product Codes," *IEEE Trans. Inf. Theory*, IT-16(4), pp. 446–451, July 1970.
31. S. M. Reddy, "On Decoding Iterated Codes," *IEEE Trans. Inf. Theory*, IT-16, pp. 624–627, September 1970.
32. E. J. Weldon, Jr., "Decoding Binary Block Codes on Q-ary Output Channels," *IEEE Trans. Inf. Theory*, IT-17, pp. 713–718, November 1971.
33. S. M. Reddy and J. P. Robinson, "Random Error and Burst Correction by Iterated Codes," *IEEE Trans. Inf. Theory*, IT-18, pp. 182–185, January 1972.
34. N. Q. Duc and L. V. Skattebol, "Further Results on Majority-Logic Decoding of Product Codes," *IEEE Trans. Inf. Theory*, IT-18(2), pp. 308–310, March 1972.
35. S. Wainberg, "Error-Erasure Decoding of Product Codes," *IEEE Trans. Inf. Theory*, IT-8(6), pp. 821–823, November 1972.
36. G. D. Forney, Jr., *Concatenated Codes*, MIT Press, Cambridge, Mass., 1966.
37. J. Justesen, "A Class of Constructive Asymptotically Good Algebraic Codes," *IEEE Trans. Inf. Theory*, IT-18(5), pp. 652–656, September 1972.

Ex.1020 / Page 304 of 305
Sandisk Technologies, Inc. et al

38. S. Hirasawa, M. Kasahara, Y. Sugiyama, and T. Namekawa, "Certain Generalization of Concatenated Codes—Exponential Error Bounds and Decoding Complexity," *IEEE Trans. Inf. Theory*, IT-26(5), pp. 527–534, September 1980.

39. H. T. Hsu, T. Kasami, and R. T. Chien, "Error-Correction Codes for a Compound Channel," *IEEE Trans. Inf. Theory*, It-14, pp. 135–139, January 1968.

40. J. K. Wolf, "On Codes Derivable from the Tensor Product of Check Matrices," *IEEE Trans. Inf. Theory*, IT-11, pp. 281–284, April 1965.

41. W. Posner, "Simultaneous Error-Correction and Burst-Error Detection Binary Linear Cyclic Codes," *J. Soc. Ind. Appl. Math.*, 13, pp. 1087–1095, December 1965.

42. L. Bahl and R. T. Chien, "A Class of Multiple-Burst-Error-Correcting Codes," presented at the IEEE Int. Symp. Inf. Theory, Ellenville, N.Y., 1969.

Ex.1020 / Page 305 of 305
Sandisk Technologies, Inc. et al