WILEY

CODE DESIGN FOR DEPENDABLE SYSTEMS

Theory and Practical Applications

EIJI FUJIWARA

Code Design for Dependable Systems

Theory and Practical Applications

Eiji Fujiwara
Tokyo Institute of Technology

WILEY-INTERSCIENCE
A JOHN WILEY & SONS, INC., PUBLICATION

Code Design for Dependable Systems

Code Design for Dependable Systems

Theory and Practical Applications

Eiji Fujiwara
Tokyo Institute of Technology

WILEY-INTERSCIENCE
A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey
Published simultaneously in Canada

Library of Congress Cataloging-in-Publication Data is available.

ISBN-13 978-0-471-75618-7
ISBN-10 0-471-75618-0

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Contents

Preface

Error control coding theory has been studied for over half a century, and it is still going stronger than ever. The most recent examples are the turbo codes and the low-density parity check codes (LDPCs). Also, during these years, error control codes have been extensively applied to various digital systems, such as computer and communication systems, as an essential technique to improve system reliability. As an integral part of modern day high-speed dependable systems and semiconductor memories, high-speed parallel decoding is essential. Error control codes suitable for high-speed parallel decoding are regularly expressed and studied in parity-check matrices. For highly reliable communication systems and disk memory systems, on the other hand, serial decoding based on linear feedback shift registers (LFSRs) is used. Error control codes for serial decoding are typically expressed and studied using generator polynomials. In this book, the former codes are called *matrix codes* and the latter *polynomial codes*. So far, traditional coding theory has been studied mainly using code generator polynomials. We emphasize that the linear codes expressed in polynomials can always be expressed using parity-check matrices, but the converse is not always possible. This book focuses specifically on the design theory for matrix codes and their practical applications, which has been seriously lacking in the traditional scope of coding theory investigations.

In dependable computer systems, many types of error control codes have been applied to memory subsystems and processors in order to achieve efficient and reliable data processing and storage. Some systems could never have been realized without the application of cost-effective error control codes, mainly very large capacity, high-speed semiconductor memories, very high-density magnetic disk memories, and recent optical disk memories such as compact disc (CD) and digital versatile disc (DVD). More recently mobile digital systems have gained wide popularity, and these systems are sometimes operated under unfavorable environments where electromagnetic noise, α-particles and cosmic rays abound. Modern high-speed, high-density VLSI processors and semiconductor memories are operated at low supply voltage levels and thus low logic signal swing; they therefore are vulnerable to external disturbances that can induce transient errors. Transient errors are a dominant concern in today's digital systems. Error control

coding is the most efficient and effective way to tolerate these errors, and is expected to become ever more important in future VLSI systems.

The challenge is to choose among many different applications of error control codes. Often a new application calls for a new type of code that can be developed most efficiently to fit a new requirement. Matrix codes are far more flexible compared with polynomial codes. Parity-check matrices can be manipulated easily. Some known examples are column vector exchange in a matrix, the odd-weight-column matrix, the low-density matrix, and the rotational matrix form. These manipulations of matrices have yielded many useful codes for important applications. Polynomial codes, on the other hand, are impossible to be manipulated in a similar way for code design fine-tuning. The main reason is that the matrix code is capable of expressing various types of code functions and thus allows for very high design flexibility. In practice, such flexibility has led to excellent code designs, satisfying the various reliability requirements of the dependable systems.

This book builds on the author's previous book, *Error Control Coding for Computer Systems* (Prentice-Hall, 1989), and it likewise aims at introducing the latest developments and advances in the field. However, as was mentioned earlier, additionally the book is unique in its concentration on the treatment of matrix codes. Unlike any existing coding theory books, this book will not burden the reader with unnecessary background on polynomial algebra. The book includes only the mathematical background essential for the understanding of matrix code construction and design. Such an arrangement frees up space for the description of some fine artistry of matrix code design strategies and techniques. Matrix code designs are presented with respect to practical applications, such as high-speed semiconductor memories, mass memories of disks and tapes, logic circuits and systems, data entry systems, and distributed storage systems. Also new classes of matrix codes, such as error locating codes, spotty byte error control codes, and unequal error control codes, are presented in their practical settings. The new parallel decoding algorithm of the burst error control codes is demonstrated and further extended to the generalized parallel decoding of the codes.

Chapter 1 provides background and a preview of material covered in the subsequent chapters. First, it defines faults, errors, and failures and explains the many types of faults and errors. This is the core knowledge needed to understand what constitutes a good code. To design an efficient and effective code for a given application, it is important first to know what types of errors matter, how much the system's reliability can be improved by coding techniques, and what are the constraints on check-bit length, decoding speed, and so forth. The matrix code designing procedure is laid out in this chapter from this standpoint. The chapter concludes with a brief introduction to the competitors of the coding technique in dependable systems, namely conventional error recovery techniques and / or error masking techniques.

Chapter 2 provides the fundamental mathematical background and coding theory necessary to understand the later chapters. The chapter covers the matrix representations of well-known error control codes, such as simple parity-check codes, cyclic codes, Hamming codes, BCH codes, Reed-Solomon codes, and Fire codes. These codes are manipulated in the later chapters in examples of how matrix codes satisfy the system requirements for given applications.

Chapter 3 discusses the matrix code design techniques related to high-speed decoding, area efficient encoding / decoding hardware, modularized organization of encoding / decoding circuits, and so forth.

Chapters 4, 5, 6, and 7 cover topics on matrix code design for high-speed semiconductor memories. Depending on the application, the matrix code can be designed to handle bit or byte errors and in some cases a mixture of both bit and byte errors. The latter are typical errors found in large capacity semiconductor memory systems using high-density RAM chips. Chapter 4 discusses bit error control codes, such as the modified Hamming single-bit error correcting and double-bit error detecting (SEC-DED) codes, the modified double-bit error correcting BCH codes, and the memory on-chip codes. For the memory systems using byte-organized RAM chips, single-byte error correcting (S*b*EC) codes, and single-byte error correcting and double-byte error detecting (S*b*EC-D*b*ED) codes, are presented in Chapter 5. The codes for the mixed type of bit errors and byte errors are presented in Chapter 6. Among them, a byte error detecting SEC-DED code, developed by the author and his colleague in the 1980s, has found practical application in recent workstations. Chapter 7 presents a relatively new class of byte error control codes: spotty byte error control codes. This class of codes has been specifically designed to fit the large capacity memory systems that use high-density RAM chips with wide input/output data of 8, 16, and 32 bits. Also a general class of these codes with minimum Hamming distance-d and with maximum distance separable (MDS) characteristics is presented in this chapter. The well-known Reed-Solomon codes are included in these generalized codes, which makes them practically and theoretically important. They will be quite useful for future applications.

Chapter 8 presents the generalized parallel decoding algorithm for error control codes. Initially developed for burst error control codes, this new decoding algorithm includes the conventional parallel decoding algorithm of the existing bit/byte error correcting codes. The generalized algorithm can also be used for multiple burst or byte error correcting codes. The chapter takes this new algorithm and demonstrates how the parallel decoding method can be implemented in combinational circuits. In addition the chapter addresses the important problem of glitches in parallel decoding circuits. Parallel decoding circuits depend heavily on large exclusive-OR tree circuits, which are well known to readily produce glitches. The glitches are the unwanted logic signal transitions that can generate, propagate, and accumulate in the logic circuits and then induce noise and instability on the power supply lines. The chapter explains why the glitches are generated, how they are propagated and accumulated in the circuits, and how to reduce these undesirable effects.

Chapter 9 presents a new class of codes, namely error locating codes. Error location is an error control function lying midway between error correction and error detection. An error locating code will indicate where the errors lie but not the precise erroneous digit positions. This type of codes is useful for identifying the faulty block, faulty package, or faulty chip, and thus enables fault isolation and reconfiguration. The chapter includes practical codes for memory systems to use in locating faulty packages/cards. It also provides a practical code for locating faulty chips. Both codes have the capability to correct single-bit errors, even though the codes are mainly designed for identifying the faulty areas. In addition, burst error locating codes are introduced here. The chapter concludes with a precise analysis of error locating codes with an emphasis on the code conditions and their relation between error locating codes and error correcting/detecting codes.

Chapter 10 shows yet another new class of unequal error control (UEC) codes. In many applications certain positions in a word have higher error rates or require more protection. The UEC codes can indicate the area in a word having a higher error rate with stronger error control code functions, and the area having a lower error rate with weaker error

control functions. In other words, this type of code has different code functions within a code word, depending on the area and the associated error rate. The chapter provides optimal codes with some UEC code functions. Similar codes exist in unequal error protection (UEP) codes. This type of code protects the valuable information part of a word against errors. For example, control information or address information in communication messages or computer words, or similarly pointer information in the database words, must be more protected from errors than their other parts. The chapter provides some UEP codes that protect against burst errors and also against single-bit errors. The chapter includes examples of UEC and UEP codes used in holographic memories and lossless compressed data.

Chapters 11, 12, 13, and 14 present the codes for some specific systems, namely mass memories such as magnetic tapes and disks, logic circuits and systems, data entry systems, and distributed storage systems. Chapter 11 covers the codes designed specifically for mass memories such as tape memories, magnetic disk memories, and recent optical disk memories. The various modified types of Reed-Solomon codes and adaptive parity codes are presented to the tape memories and to the disk memories. Codes for recent CDs and DVDs are also introduced. Chapter 12 mentions error checking for logic systems using efficient error detecting codes. An important concept of self-checking is first introduced. The chapter then clarifies how the errors in the logic circuits and systems are detected, how the error detecting checker circuits are implemented, how the errors in the checker itself are detected, and how the self-testing checkers are implemented. Especially self-checking ALU is presented by using parity-based codes, and also self-checking design for processor systems is demonstrated. Chapter 13 presents the codes for data entry systems. In these systems, in general, nonbinary symbols are routinely used in character recognition systems, and recent two-dimensional symbols. The chapter characterizes the errors that occur in these nonbinary symbols as asymmetric errors and presents some asymmetric error control codes. These codes are basically nonlinear, and are designed by using elements in newly defined rings. Also nonsystematic nonbinary asymmetric error correcting codes are designed based on a multilevel coding method and a set-partitioning algorithm, and QR codes and two-dimensional unidirectional clustered error correcting codes are presented for two-dimensional matrix symbols. Chapter 14 provides the codes for distributed storage systems connected via networks. Codes for recent RAID systems that tolerate two disk failures are introduced, and then an efficient error recovery scheme from multiple disk failures in the distributed storge system is discussed and is implemented by using block design in combinatorial theory.

The introductory portion of the book, Chapters 1 and 2, and the parts of Chapters 3, 4, 5, 6, 8, 9, and 10, can be used as the text for a course at an advanced undergraduate level or for an introductory one-semester course at the graduate level. For graduate classes and advanced students who have the background in mathematics, logic circuits, and rudimentary knowledge of codes, the book can be used as a whole with selected topics from each of the chapters. Practicing engineers / designers will find useful discussions in Chapters 6 to 14, which demonstrate, in detail, the procedure of designing sophisticated codes in practical form. For the practicing engineer, Chapter 2 presents mathematics and coding theory, not in strict form but in introductory form, which is necessary in understanding the later chapters. Many examples, figures, exercises, and references are provided in each chapter of the book. Many attractive codes with practical code parameters and their evaluation data on decoding hardware and error detection capabilities

are fully demonstrated. These can be used by practicing engineers as a practical guide and handy reference.

My sincere appreciation goes to many people. Professors Jack K. Wolf of the University of California San Diego, Hideki Imai of the University of Tokyo, T. R. N. Rao of the University of Louisiana Lafayette, and Bella Bose of Oregon State University encouraged me to continue my research on code design theory and to write this book. Emeritus professor Yoshihiro Tohma of Tokyo Institute of Technology, Professors Takashi Nanya of the University of Tokyo, Hideo Ito of Chiba University, and Jien-Chung Lo of the University of Rhode Island gave important suggestions and valuable discussions on research for dependable systems. Recently Professor Lo also provided valuable comments on the final book and an important discussion on glitches, (i.e., logical noise) that are generated, propagated, and accumulated in large exclusive-OR tree circuits in the parallel decoder of the codes. The author's NTT colleagues, Dr. Shigeo Kaneda, now professor at Doshisha University, and Dr. Kazumitsu Matsuzawa, now professor at Kanagawa University, collaborated to develop practical codes for computer memories. Dr. Masato Kitakami, now associate professor at Chiba University, Dr. Mitsuru Hamada, now associate professor at Tamagawa University, Dr. Shuxin Jiang, Dr. Saowapa Kiattichai, Dr. Hongyuang Chen, Dr. Kazuteru Namba, Dr. Ganesan Umanesan, Dr. Haruhiko Kaneko, Dr. Kazuyoshi Suzuki, Mr. Tsuyoshi Tanaka, Mr. Toshihiko Kashiyama, and Mr. Hiroyuki Ohde devoted themselves to designing the excellent codes in their master's and/or doctorate course programs at the Tokyo Institute of Technology. Much of the motivation for making the codes practical was due to discussions with many researchers and engineers in Japanese industry.

Thanks also go to art designer, Mr. Ippei Inoh, a friend of mine, who proposed and directed the marvelous idea of the front cover design. Ms. Tiki Ishizuka, a computer graphic designer, arranged the wonderful fine art of this cover. You can see ''Hoh-Oh,'' a legendary happy bird, in the center of the front cover whose original pattern was introduced from China more than one thousand years ago to Japan, and since then appeared as an art design in Japanese art and craft products. I sincerely hope the book will bring happiness and pleasure to the reader.

At this point in a preface, I usually thank my wife, Sachiko, and my daughter's family, Sayaka, Makoto, and Asuka, for encouraging me in continuing this difficult project.

EIJI FUJIWARA

(*Autumn in 2005 on the foot of Mt. Fuji*)

CONTENTS

1

Introduction

Before designing a dependable system, we need to have enough knowledge of the system's faults, errors, and failures of the dependable techniques including coding techniques, and of the design process for practical codes. This chapter provides the background on code design for dependable systems.

1.1 FAULTS AND FAILURES

First, we need to make clear the difference between three frequently encountered technical terms in designing dependable systems—namely faults, errors, and failures. These terms are fully defined in [LAPR92, AVIZ04]. Faults are primarily identified as the generic sources of abnormalities that alter the operation of circuits, devices, modules, systems, and / or software products. Failure can arise from any type of possible faults. Faults are often called *defects* when they occur in hardware and *bugs* when in software.

1.1.1 Faults

As causes of failure, faults are sometimes predictable but difficult to identify. Faults can occur during any stage in a system's or product's life cycle: during specification, design, production, or operation. Faults are characterized by their origin and their nature [LAPR92, GEFF02].

Origin of Faults Timing is a factor because faults can provoke failure in the operation phase at any one of a system's previous life phases: specification, design, production, and operation.

During the specification phase, for example, an incomplete definition of services may lead to different interpretations by the client, the designer, and the user. Eventually, in the

Code Design for Dependable Systems: Theory and Practical Applications, by Eiji Fujiwara

operation phase, the failure becomes evident when the services provided differ from the user's expectations.

During the design and the production phases, for example, a designer's lack of sufficient knowledge of architectural levels, structural levels, and the like, may result in a type of physical defect that induces, for example, short or open circuits.

During the operation phase, for example, an elevation of ambient temperature can cause electronic devices and products to malfunction.

Nature of Faults During the specification and the design phases, faults that occur are called *human-made faults*. During the production and the operation phases, these may occur *physical faults*, *hardware faults*, *or solid faults.* Each type is due to some physical abnormality in the component arising from aging or defective materials. Faults are of two types in their duration:

1. *Permanent.* These faults arise, for example, from a power supply breakdown, defective open or short circuits, *bridging* or *open lines*, *electro-migration*, and so forth. The defects in the input / output of the logical circuits or lines are called *stuck-at '1' faults* or *stuck-at '0' faults*.
2. *Temporal*. These faults can be *transient* or *intermittent*. Transient faults occur randomly and externally because of external noise, namely environmental problems of external *electromagnetic waves* but also external particles such as α-particles and neutrons. Intermittent faults occur randomly but internally because of unstable or marginally stable hardware, varying hardware or software state as a function of load or activity, or *signal coupling* (i.e., *crosstalk*) between adjacent signal lines. Some intermittent faults may be due to *glitches* [LO05], which are unpredictable spike noise pulses occurring and propagated especially in large exclusive-OR (XOR) tree networks (see Chapter 8). Parallel decoding circuits of error control codes with large code lengths require large exclusive-OR tree networks, so glitches can become serious problems. This topic will be covered in more detail in Section 8.3.

Transient faults and Intermittent faults are the major source of errors in modern-day digital systems. Some reports show that more than 60% of all failures in computer systems are caused by transient or intermittent faults. For example, in DRAM (Dynamic Random Access Memory) chips, transient errors result mainly from α*-particles* emitted by the decay of radioactive particles in the semiconductor materials [MAY79, NOOR80, SAIH82]. One identified source of α-particles is the lead solder balls used to attach the chip to the substrate. As they pass through the chip, α-particles create sufficient electron-hole pairs to add charge to the DRAM capacitor cells. These particles have low energy level, and thus have very low probability of causing more than one memory cell to flip when the memory cells are not packed in extreme density. In today's ultra–high-density RAMs, not only DRAMs but also SRAMs (Static Random Access Memories), it has been recognized that multiple cosmic-ray-induced transient errors are a serious problem [OSAD03, 04].

Temporal errors have also been observed in microprocessor chips. The trend toward smaller geometries by ever-shrinking semiconductor designs results in lower operating signal voltages and higher speed operation, and therefore brings additional transient or intermittent errors into play [KARN04]. In today's ubiquitous digital device or system environment, PDAs and personal computers equipped with these high-speed microprocessor chips and high-density RAM chips are further prone to be damaged by even worse circumstances when operated in airplanes at high altitude or near the high-voltage electric power lines.

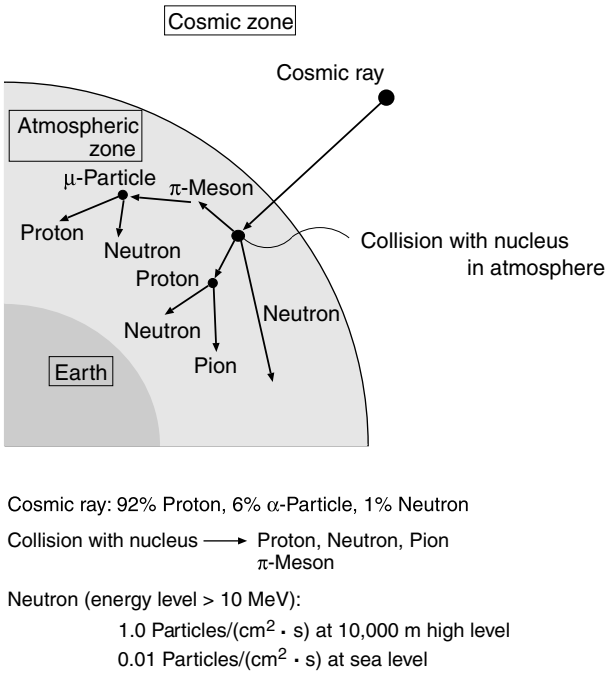


Figure 1.1 *Cosmic rays.*

The important point is that the faults due to temporary environmental problems do not need repair because the hardware is physically undamaged.

Cosmic rays, however, can give rise to significant transient errors, called soft errors [KARN04, MAKI00, HAZU00, ZIEG98, MASS96, CALV94]. Figure 1.1 shows the cosmic ray and its influence at the earth surface level. In the cosmic environment heavy particles with very high energy from solar winds can penetrate the semiconductor chips in satellite digital systems and cause more than double-bit errors [MUEL99]. Sometimes they can cause physical faults such as *latchup* in CMOS circuits.

A detailed report of field testing for soft errors due to cosmic rays was presented in 1996 [ZIEG96a, 96b, 96c, OGOR96, SRIN96]. In the report cosmic rays are defined as particles in solar wind originating in the sun or as galactic particles that enter the solar system striking atmospheric atoms and creating a shower of secondary particles. Most such particles produced by the shower either decay spontaneously or lose energy gradually, and eventually lose all energy in the cascade. Some of these particles may strike the earth. Therefore the cosmic rays at sea level consist mostly of neutrons, protons, pions, muons, electrons, and photons. About 95% of these particles are neutrons with no charge but with the high energy (more than 10 MeV) that causes significant soft errors or *latchups* in electronic circuits. So cosmic rays can create multiple errors. Altitude causes the neutron flux to increase exponentially, and hence the fail rate of electronic circuits at airplane altitude is about one hundred times worse than at terrestrial level. Concrete shielding with several feet of thickness can significantly attenuate the flux of these high-energy particles.

Figure 1.2 shows how neutrons and other particles, including α-particles, generated by the collision of nuclei in the atmosphere, can strike silicon chips and produce sufficient electron-hole pairs in the chips to impair their functioning.

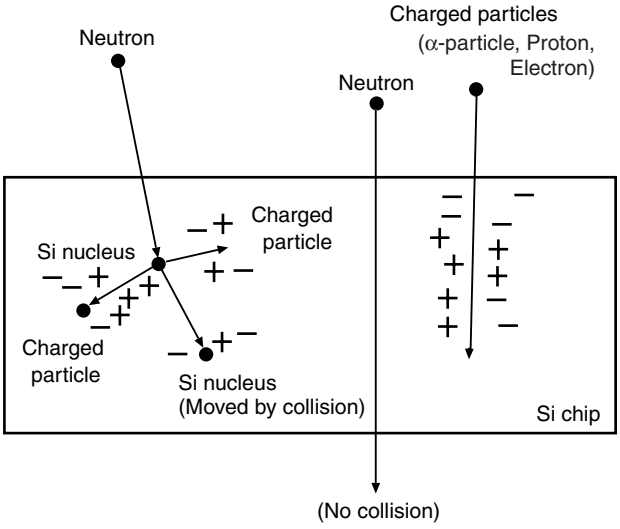


Figure 1.2 *Electron holes in a silicon chip caused by particles.*

1.1.2 Failures

A failure is defined as *nonperformance that occurs when a delivered service no longer complies with its specifications* [LAPR92], and a failure is also defined as *nonperformance when the system or component is unable to perform its intended function for a specified time under specified environmental conditions* [LEVE95].

Some types of failure are defined with respect to specific conditions. For example, a *value failure* means that the value of the delivered service does not comply with the specification and a *timing failure* represents a response in incorrect timing, either faster or slower than the specified time. A *temporary failure* means an erroneous behavior at a certain moment lasting only a short time. A *crash failure*, or *catastrophic failure*, is the one that stops the mission because the system is completely blocked.

1.2 ERROR MODELS

An error is a manifestation of an unexpected fault within a system that is liable to lead to system failure. The transformation of a fault to an error is called *fault activation*. The mechanism that creates errors in the system and finally provokes a failure is called *error propagation*. Before provoking a failure, errors can be masked or corrected by some error control mechanisms such as error correcting codes, retries, or triple modular redundancy (TMR) and thus recovered without inducing a system failure.

A fault remains in passive mode until an error first appears at some structure of the system. This occurrence is called an *initial activation* and the error is called a *primitive error*. In this case *latency* is defined as the mean time between the fault's occurrence and its initial activation as an error. Figure 1.3 presents the causal relationship between fault, error, and failure. Various types of errors can occur, and these different types are covered below.

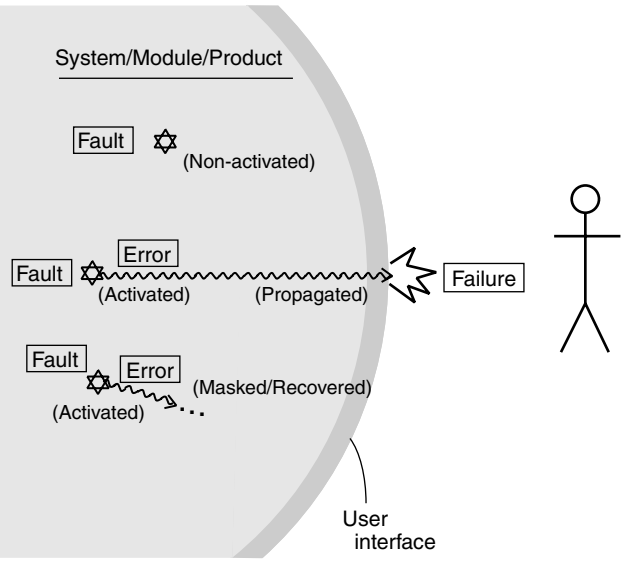


Figure 1.3 *Fault, error, and failure.*

1.2.1 Hard Errors and Soft Errors

Hard errors are caused by permanent faults; they therefore affect the system functions for a long period of time. This type of error is typically provoked by faults that appear as open or short anywhere on the chips, modules, cards, or boards. Hard errors are also called *permanent errors*.

Soft errors, on the other hand, are caused by temporal faults, especially those resulting from external causes. Soft errors have a limited duration, meaning they interrupt system functions for a very short time period. The most likely sources of soft errors are radioactive particles and external noise. Alpha particles and cosmic particles [ZIEG96a, ZIEG96b, ZIEG96c, OGOR96, SRIN96] are the major contributors mentioned previously. Therefore soft errors are also called *transient errors*. The *intermittent errors* are provoked by intermittent faults.

1.2.2 Random Errors, Clustered Errors, and Their Mixed-Type Errors

Multiple errors that occur randomly in time and / or space are called *random errors*. Error can occur in every bit position of a word with almost equal probability. The random type of error is unpredictable and is typically caused by white noise or external particles.

Errors may cluster non-uniformly in a word, and these multiple errors may gather in particular and unpredictable positions in the word. *Clustered errors* include *burst errors* and *byte errors*. Burst errors occur typically in disks or tape memory. Byte errors are typically found in semiconductor memory. The difference is in the data-recording medium. In disk memory, the data are recorded on a continuous surface. In semiconductor memory, the data are stored in RAM chips, and a data fragment, called a *byte*, is read or stored in each chip. In disk or tape memory, defects or dust particles on the recording surface can cause burst errors to occur anywhere in the continuous recording medium.

Clustered errors may occur in the two-dimensional matrix symbols as well as in the tape or disk memory of a continuous two-dimensional recording medium. In semiconductor memory, on the other hand, byte errors may occur in a fragment of readout data, namely in a single byte, corresponding to the faulty chip. This is because each chip is physically separated and independent, and therefore the presence of a fault in a chip does not extend to the adjacent chips. Figure 1.4 illustrates the different cases of random errors, byte errors, and burst errors.

Another error model consists of mixed clustered and random errors in the operational phase. The clustered errors mentioned above are sometimes caused by physical faults due

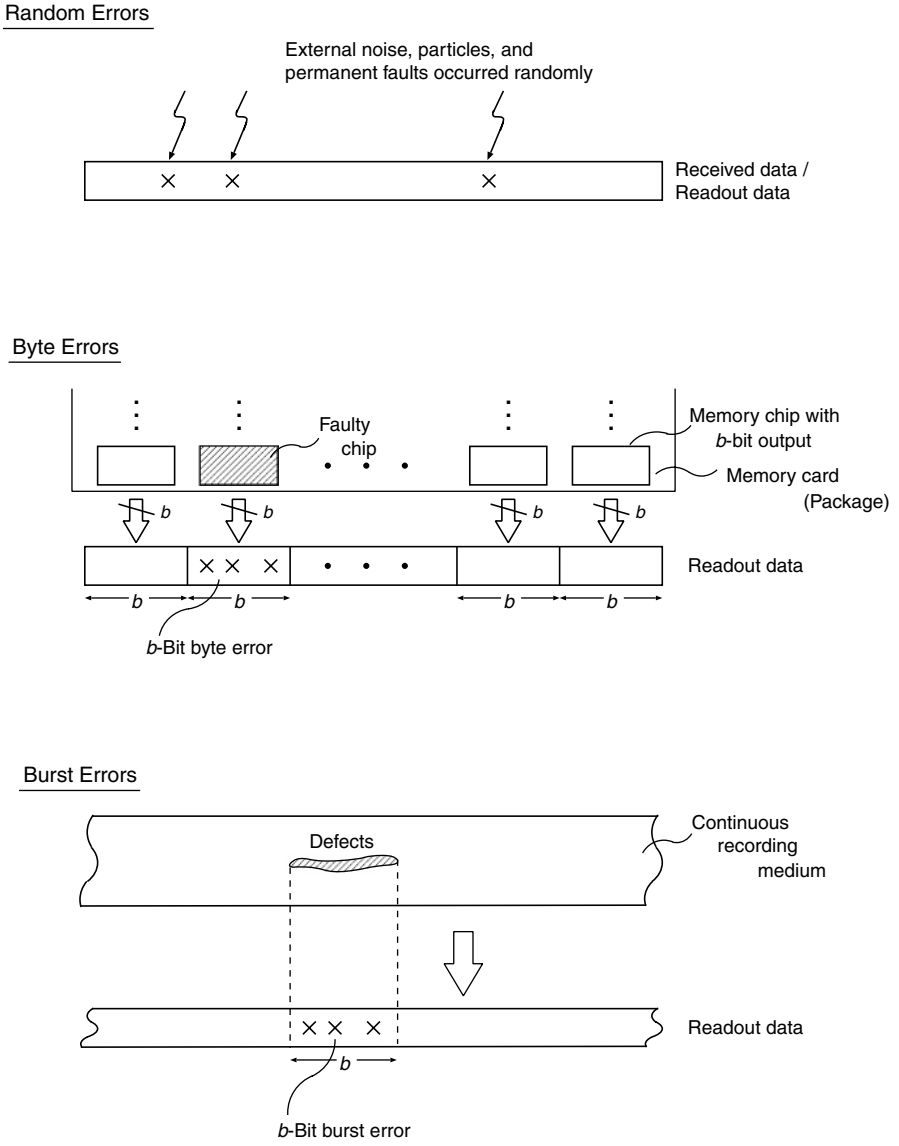


Figure 1.4 *Models of random errors, byte errors, and burst errors.*

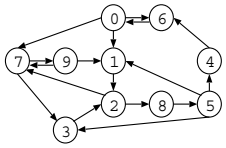
to aging problems. However, systems and devices are more prone to damage from transient faults than from physical faults. Transient faults are source of random errors. Therefore, when a physical fault occurs during the operational phase, both types of error—clustered and random—must be taken into account. For example, in semiconductor memories with byte-organized RAM chips, the major types of errors are transient errors, (i.e., random bit errors) caused by α-particles or external noises. After some time in operation, byte errors will occur due to the aging of RAM chips. Therefore both bit errors and byte errors, meaning both random errors and permanent errors, may occur separately or simultaneously. A similar situation holds for transmission systems, where both random bit errors and burst errors can occur. Chapter 6 deals with the codes which control the mixed type of single-byte errors and random bit errors.

1.2.3 Symmetric Errors, Asymmetric Errors, and Unidirectional Errors

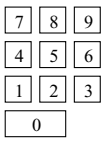
In binary systems the probability of errors that force 0 to 1, called *0-errors*, is, in general, equal to those going from 1 to 0, called *1-errors*. This class of errors is known as *symmetric errors*. When these errors occur with unequal probabilities, they are called *asymmetric errors*. In the binary asymmetric error model, only one type of error, either 0-errors or 1-errors, can occur, and the error type is known a priori. If both error types occur but are not mixed, then this class of errors is said to be *unidirectional errors* [BLAU93]. In binary systems these errors are caused by symmetric faults, asymmetric faults, or unidirectional faults.

In nonbinary systems using numerals, $0, 1, 2, 3, \ldots, 9$, or alpha-numeric symbols, asymmetric errors are the type that occur. That is, the probability of an error that forces one nonbinary symbol A to another symbol B is sometimes different from that of symbol A forced to yet another symbol C. For example, in handwritten character recognition systems, the probability of a 7 being mistaken for a 9 is much higher than that of a 7 being mistaken for a 4, or $p(9|7) \gg p(4|7)$, where $p(B|A)$ means probability of a symbol A being mistaken for another symbol B. This is because the numbers 7 and 9 are close in shape whereas 7 and 4 are not so similar. Likewise in keyboard input systems the symbols located on adjacent keys can be more easily mistyped. Figure 1.5 shows examples of these error models. In the asymmetric error model, the error graphs are not perfect and sometimes not bi-directional. On the other hand, in the symmetric nonbinary error model, they are perfect and bi-directional.

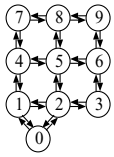
If symbols are removed or added in a word, as is sometimes caused by human mistakes (i.e., *human-made faults*), this class of errors is called *deletion errors* or *insertion errors*, respectively.



(a) Example of an asymmetric error graph for handwritten character (numerals) recognition systems



(Numeric key pad layout)



(b) Asymmetric error graph for keyboard systems

Figure 1.5 *Asymmetric errors in nonbinary systems.* Source: *[KANE04] © 2004 IEEE.*

1.2.4 Unequal Error Probability Model and the Unequal Error Protection Model

The probability of error appearing in any position of a word is usually considered to be equal. However, there is an error model to consider where some positions of a word have higher error probability than other positions. These are sometimes caused in the system by using devices with low reliabilities in the corresponding positions of a word, or by having error-sensitive areas in some positions of a word which are more vulnerable to external noises or have a low noise margin. In such cases the erroneous positions or areas with high error probabilities are known a priori. The type of error model that is relevant here is known as the *unequal error probability model*. The codes based on this error model are called *unequal error control (UEC) codes*. Chapter 10 will discuss the UEC codes and present its application to holographic memory, which has non-uniform error probability in the recording medium.

Some types of computer words or communication messages have a structure such that the information included in one part of the word is more important or more valuable than that in other parts. Control and address information in the computer or communication messages, and pointer information in the database words are good examples. In general, errors in this part, such as errors in control information or in pointer information, will cause much more serious damage to the subsequent processes in the system. Another example is error in the decimal numbers. During processing of digital data of conventional decimal numbers or measurement data, errors in the higher order digits will yield more devastating effects on the subsequent processes in digital systems than errors in the lower order digits. Therefore the higher order digits should be more strongly protected against errors than the lower order digits. This type of error model is known as an *unequal error protection model*. The codes based on this are called *unequal error protection (UEP) codes* and will also be discussed in Chapter 10.

1.3 ERROR RECOVERY TECHNIQUES FOR DEPENDABLE SYSTEMS

Error detection is an essential part of a dependable system design. Ideally, error detection will block the propagation of an error during online operations, before it reaches the system interface and causes a system failure. The error is best be detected immediately as it occurs so that its effect can be minimized.

Upon detecting an error by an error detection mechanism, some error recovery technique must mask the fault or remove it, and thus block the error's propagation. Among such mechanisms, error correcting codes and triple modular redundancy (TMR) correct errors or mask faults directly, that is, without an additional error detection procedure.

Some important error detection techniques and error recovery techniques, comparative to the error control coding techniques, are briefly described below. For more information, the reader is referred to the following excellent texts and papers on dependable systems or design techniques for fault tolerance: [AVIZ78, SIEW82, RENN84, EZHI86, ABRA86, PRAD86, JOHN89, LEE90, AVRE00].

1.3.1 Error Detection / Error Checking

Prediction & Comparison The basic error detecting or checking concept for online operations exists in *prediction & comparison*. That is, the output of the circuit / module is predicted from the input, and then the predicted output and the original circuit / module

output are compared bit by bit. The errors are detected if the actual output is not perfectly matched to the predicted output.

Duplication is an important and popularly used error detection technique in dependable digital systems. This is a special case of prediction & comparison, because the output is generated, or predicted, by a copy of the circuit / module and then compared with that of the original. This concept exists also in *software duplication* where a copy of the same or equivalent software is prepared and executed, and then the outputs are compared.

Parity-prediction is another important and popularly applied technique. The output parity bit is predicted from the input, and then compared with the parity bit generated from the original output.

Error Detecting Codes Error detecting codes typically deal with simple parity-check codes, cyclic codes, checksum codes, and other basic linear codes, as will be explained in Section 2.3. Some further important and newly developed codes will be presented in later chapters.

The application of error detecting codes in online operations is also called *checking* or an *online testing*. The error detection circuit is denoted as a *checker.* These applications will be examined in-depth in Section 12.1 where the self-checking concept is presented. Additional topics on how to detect errors caused by faults in the checker itself and how to design such checkers are covered in Section 12.2 where self-testing checkers are discussed. In summary, Chapter 12 covers error-checking concepts, self-testing checker design methodologies, and concrete checker design for logic circuits and for computer systems.

Watchdog Timer and Watchdog Processor A watchdog timer is very useful for detecting faults in a system. The idea behind this scheme is that some part of the system should act to indicate fault-free status so that absence of this action is indicative of a fault. Also the timer must be repeatedly reset by the system. Failure of the system to perform the reset function results in the system being turned off to prevent a system failure from occurring.

A watchdog timer can be used to detect faults in both the hardware and the software of a system. In many applications software routines are expected to execute within prespecified time frame. In digital control systems, for example, the routines execute repetitively at specified intervals. If a routine suddenly needs more than the expected time to execute, the fault may be in the software's, for example, infinite loop [JOHN89]. In this regard the watchdog timer is an important *control flow check tool*.

A watchdog processor is an extension of the concept of a watchdog timer. This is a special subprocessor that checks the online operations of the processor being checked. The watchdog processor runs the watchdog programs that collect information from the processor being checked and generate *signatures*, such as address and data information, and processor state information, during online operations. The new information is then compared to that already prepared in the watchdog program.

1.3.2 Error Recovery / Error Masking

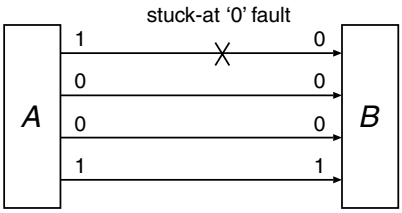
Error recovery techniques are essential to improving system reliability. The important recovery techniques, as was mentioned before, include coding techniques and some modular redundancy techniques, such as TMR, that correct or mask the faults directly. Other effective error detection methods are also available to mask the faults after the detection of errors, for example, self-checked duplication and sift-out redundancy, as discussed below.

Error Correcting Codes Many different error control codes have been studied and developed to correct and / or detect the types of errors mentioned in Section 1.2. Among the most practical matrix codes are those presented in this book.

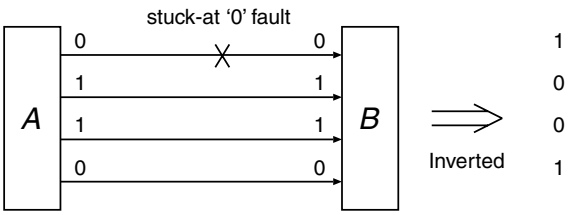
Error correcting codes head the list of the most effective and efficient techniques used to mask faults, both temporal and permanent. The coding approach involves some redundancy, for example, additional check bits, additional hardware in the form of encoding / decoding logic circuits, and additional decoding time delay. Nevertheless, the coding performance is superior to that competitive techniques, especially in quickly masking of temporal faults. For this reason error control codes are still being extensively applied to various digital systems to improve their reliability.

Retry Just as *space redundancy* requires additional hardware resources, the retry method called *time redundancy* which requires additional time to perform multiple identical operations of commands or programs immediately after errors are detected. This very simple technique requires almost no additional hardware but can very effectively recover system operations from temporary faults, meaning transient and intermittent faults. Therefore the retry method is popularly applied to digital systems, including processors, main memories, disk memories, tape memories, and I/O devices.

Alternate data retry, abbreviated by *ADR* [SHED78], is a kind of retry operation that is effective in masking permanent faults besides temporary faults. Figure 1.6 presents the principle behind masking a single permanent fault by ADR. Note that this simple example shows the even-parity encoded bus circuit with four lines, including a parity line. Figure 1.6(a) shows that if a stuck-at '0' permanent fault occurs in the first bus line, then the even-parity encoded data from circuit *A*, here 1001, is received at the input of the circuit *B* as 0001, which is an odd-parity encoded data. Therefore a single error can be detected by examining the parity check of the data. Next, by the ADR method, in Figure 1.6(b), the bit-by-bit complement of the original data, which is 0110, is transmitted from circuit *A* to



(a) Error detection by parity check



(b) Retried by complemented data

Figure 1.6 *Principle of ADR (illustrated by even-parity encoded bus line circuits as an example).*

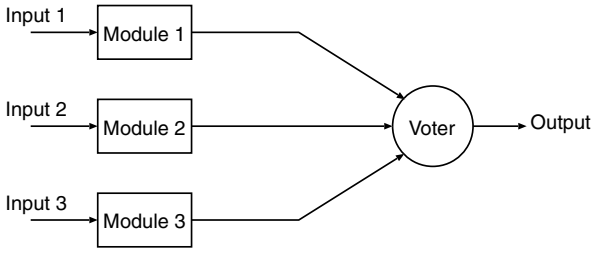
the input of the circuit B. Even though the first line is still preserving a stuck-at '0' fault, the fault is masked because the data on this line are also a '0'. Finally the received data are inverted, and then the original correct data, 1001, are recovered. In this example, a permanent fault is masked at the second stage of ADR, and finally the correct data are recovered at the third stage of ADR. Also, in this example, if the fault in Figure 1.6(a) is a temporary fault, the error it caused can be completely masked and will have no effect because the temporary fault will disappear by the time of the second stage of ADR.

In general, if the logic circuit that performs the function $F(X)$ for the circuit input X satisfies the relation

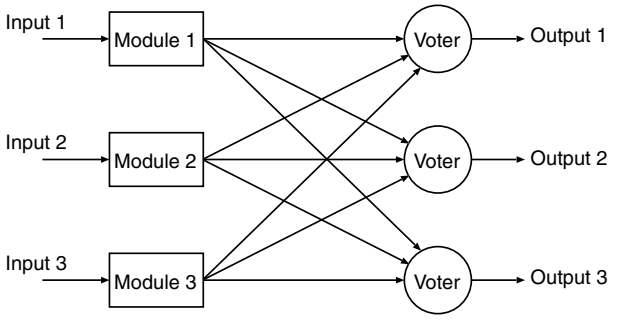
$$F(\overline{X}) = \overline{F(X)},$$

where $\overline{X}$ means the complement of X, then the ADR with bit-by-bit complementary retry at the second stage can be performed successfully. The function F that satisfies the relation above is called a *self-complementary function*, and the circuit that satisfies the relation is called a *self-complementary circuit*. The former even-parity busline circuit is a self-complementary circuit. The adder, the multiplier, and the divider are also good examples of self-complementary circuits.

N-Modular Redundancy (NMR) and Reconfiguration *Triple modular redundancy* (TMR) is the most typical form of N-modular redundancy. The TMR method triplicates the original module and performs a majority vote to determine the output of the system. If one of the modules becomes faulty, the other two fault-free modules mask the results of the faulty one when the majority vote is performed. This is shown in Figure 1.7(a). This voting concept



(a) Triple modular redundancy (TMR)



(B) Triple modular redundancy with triplicated voters

Figure 1.7 *Triple modular redundancy (TMR).*

is applied to TMR software to protect against software faults in any one of three identical or equivalent software programs that perform the same function.

The difficulty in the TMR exists in its voter. That is, if the voter fails, the system completely fails. One approach is to apply TMR to the voter itself such that three voters are used and three independent voting results are provided as shown in Figure 1.7(b). The three modules are functionally identical and receive identical inputs. The results generated by three modules are voted on by the three voters to produce three results. Each result is correct unless more than one module or input is faulty.

N-Modular redundancy (NMR) is a generalization of the TMR and is a typical *space redundancy* technique. In most cases, N is selected as an odd number so that a majority voting principle can be applied. For example, the 5MR system consists of five identical modules and a voter. This system produces correct output in the presence of, or masks, as many as two faulty modules.

The modular redundancy concept has been extended and modified by combining the concept of *reconfiguration*. The following forms show some such combinations.

Self-checked duplication is an extended form of duplication in which each module has its own self-checking mechanism in order to identify the faulty state of the module itself. In this system, two self-checked modules are operated and checked in parallel at all times. If one module is found to have errors by its own error detection mechanism, then the system output is switched to the error-free module, meaning it is reconfigured. This concept is a form of *hot standby sparing* in which the spare module operates synchronously with the online module and is prepared to take over at any time. When the online module is failed, the standby spare module takes over immediately. In contrast to the hot standby sparing, there exists *cold standby sparing* where the spare is unpowered until needed to replace a faulty module.

N-Modular redundancy with spares is also known as hybrid redundancy. It provides a basic core of N modules arranged in a voting system, and in addition spares are provided to replace faulty modules. For example, while the TMR with one spare masks one faulty module, the spare will replace the faulty module immediately upon the detection of the fault. After that spare is used, the system is still capable of masking another faulty module. Therefore two faulty modules can be masked in this system. The aforementioned 5MR requires five modules in order to mask two faulty modules, but the TMR with one spare approach requires only four modules. The system remains in the basic NMR configuration until the disagreement detector determines that a faulty module exists. One approach to fault detection is to compare the output of the voter with the individual outputs of the modules. A module that disagrees with the majority is regarded as faulty and removed from the NMR core. A spare module is then switched in to replace the faulty module. The reliability of the basic NMR system is maintained as long as the pool of spares is not exhausted. This is shown in Figure 1.8.

Self-Purging Redundancy is similar to the NMR with the spare modules approach. The main difference is that all modules operate actively in this redundancy system, unlike the NMR with spares where some spare modules are not an active part of the system until a fault occurs. This is shown in Figure 1.9. Each switch in the self-purging redundancy separates the faulty module if the module output is not equal to the voter output. The reconfiguration is essentially accomplished by the system logically removing the faulty module via the switch and thus reducing the number of N in the reconfigured NMR system.

Sift-out redundancy also requires N identical modules in the system but with every pair of two module outputs compared to identify faulty modules. If there exist $N = 5$ modules,

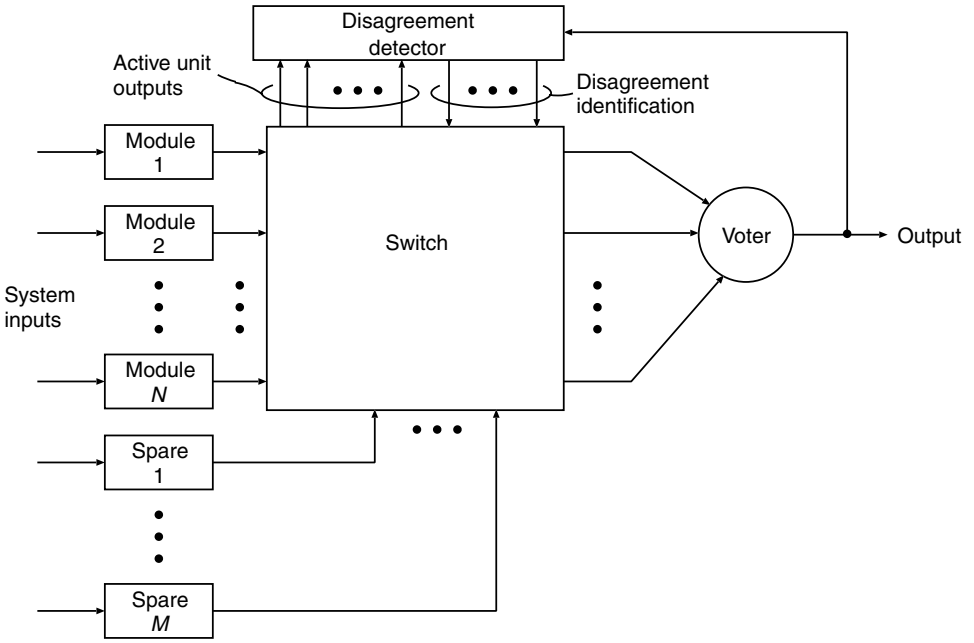


Figure 1.8 *N-modular redundancy with spares (i.e., hybrid redundancy).*

ten comparisons are performed. This redundancy requires an N-way multiplexer instead of an N-input voter, as shown in Figure 1.10. The comparator in this redundancy circuit receives all outputs of the modules and produces comparison outputs of every two modules, that is, $N(N-1)/2$ outputs, and then determines the faulty modules in the detection circuit. Finally the output of the N-way multiplexer is selected based on the faulty indication outputs of the detection circuit. This essentially masks the effects of any faulty modules.

This redundancy can tolerate up to $N-2$ faulty modules. Its tolerance is therefore equal to the TMR system with $N-3$ spares and also to the self-purging system having a voter with threshold level of two.

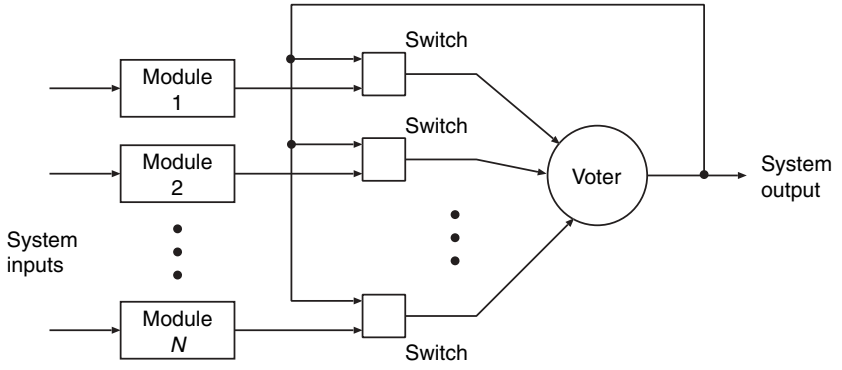


Figure 1.9 *Self-purging redundancy.*

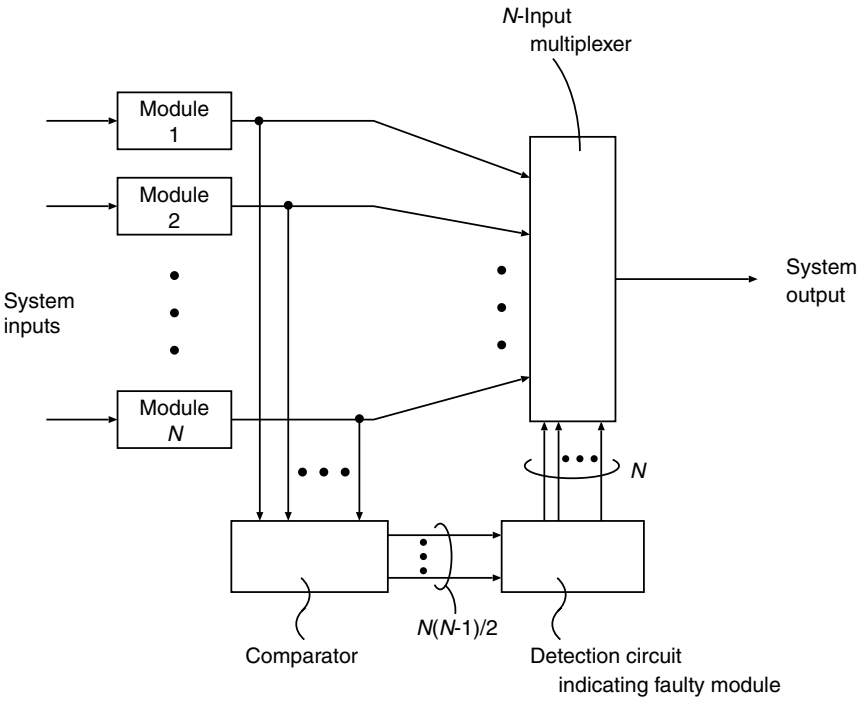


Figure 1.10 *Shift-out redundancy.*

System Recovery by Software Retry techniques require error detection by checkers, and immediately after the error detection the same operations are performed. In contrast, *checkpoint techniques* allow some latency time after error detection because the process can be restored to an earlier point of execution. *Checkpointing* is mostly implemented in software and requires some hardware to store the backup data. The techniques result from a combination of checkpointing and rollback. In checkpointing, complete copy of the system state should be saved at specific points, namely *checkpoints*, during process execution. The information to be stored is the set of system state including data, programs, machine state, and so forth, which is necessary to restart the continued successful execution from the checkpoint. *Rollback* is a part of actual recovery process and occurs after the repair, such as by reconfiguration, that removes faulty modules or equipments from the system, or after the error due to transient faults has died out. An important design criterion is how often checkpoints are to be set, that is, in determining *checkpoint intervals*. If the checkpoints are too infrequent for the actual error rate experienced, too much computation time will be lost due to rollback. On the other hand, too frequent checkpoints result in an unnecessary increase in operation time and memory due to the overhead of saving system states when establishing checkpoints.

1.4 CODE DESIGN PROCESS FOR DEPENDABLE SYSTEMS

What types of dependable techniques are the most effective in the design of dependable systems? In some cases other than coding techniques, or a combination of coding techniques and other dependable techniques, will better meet the reliability requirement or the cost / performance requirement of a system.

Before designing the error control codes, we therefore have to pay attention to a number of preconditions or preparatory measures: Where to apply the code? How to apply the code effectively? How much reliability of the system to improve and satisfy its performance by coding techniques? What are the requirements for decoding speed, and how much decoder hardware? What about the detection capability of errors falling outside the capability of the code? This section addresses all these important questions with respect to the code design process.

1.4.1 Code Functions

Error detection and error correction are the more known code functions. An important code function that lies midway between these two functions is *error location*. The error locating code indicates which blocks, or components of a word contain error but does not indicate the precise erroneous digit position nor the error value. This is a code function that is efficient for retransmission of a word segment, especially in communication systems where whole words do not need retransmitting [WOLF63]. Also in computer systems the error locating code provides the information on where to find the faulty module, faulty package / card, or faulty device, which is very useful for system maintenance. If the system is equipped with spares, then the system can be recovered by removing the faulty blocks and switching to the spares.

Figure 1.11 shows the different functions of these three code types. Because erroneous position, and error value can be determined by use of "error correction", all errors can be

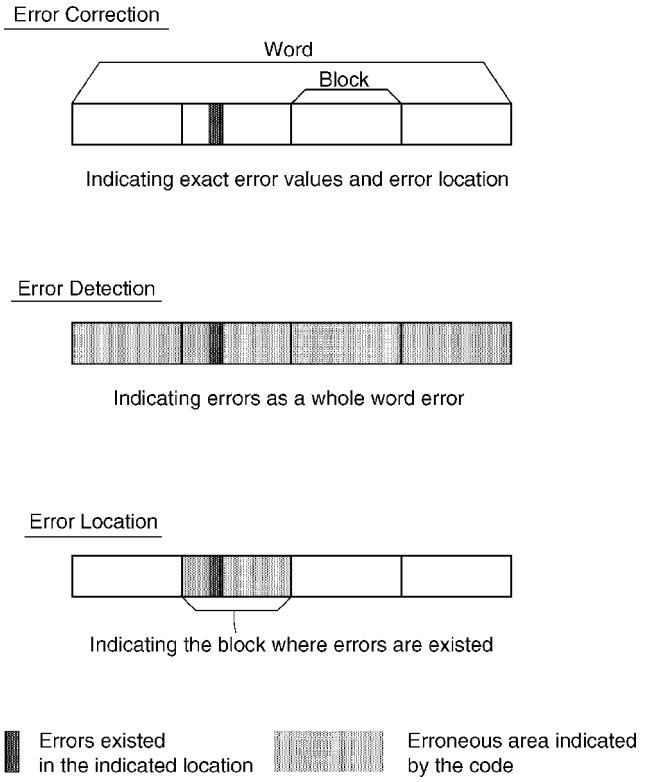


Figure 1.11 *Code functions.*

corrected. Of course, use of ''error detection'' alone does not allow any erroneous position nor error value to be determined; it only indicates the presence of error in a word. For ''error location'', as was mentioned before, only the area where the word includes an erroneous position is indicated by the code. For example, note in Figure 1.11 that the code's information is that errors exist in the second block of the word and no definite error positions in the block nor the error values are determined. Error locating codes will be covered in Chapter 9. Many practical codes, in general, have a mixture of these code functions, for example, single error correction and double error detection.

1.4.2 Code Deisgn Process

Before attempting the design of codes, we need to give the following items our careful consideration:

1. Circumstance where the systems or equipments with the coding techniques are to be applied, for example, the particular needs of medical appliances, nuclear appliances, or digital systems in aircraft or satellite,
2. Fabrication structure, that is, how the systems or the equipments are organized, for example, chip / card (package) organization, bit / byte organization, or binary / nonbinary,
3. Devices, such as memories, logic circuits, or FPGAs that are used in the system to which the coding techniques are to apply.
4. Combination of fault / error masking techniques with coding techniques.

The design process for the error control codes is presented next, and is shown in Figure 1.12. Steps 1 through 3 pertain to the phase of setting code parameters, and steps 4 and 5 are for the phase of code designing.

Step 1. Determine error rates and error types:

- Raw error rate of devices, modules, or systems, and what target error rate to attain
- Whether symmetric error, asymmetric error, or unidirectional error
- Whether equal error or unequal error
- Whether random bit error, byte error, spotty byte error,[a] or burst error
- Whether bit or byte error,[*] or rather, bit plus byte error[b]

Step 2. Determine code parameters and code constraints:

- Information-bit length, and required check-bit length
- Maximum random bit error length—or byte error length, spotty error length,[a] or burst error length
- Required decoding speed
- Required decoder hardware complexity

[a] See Chapter 7.

[b] See Chapter 6.

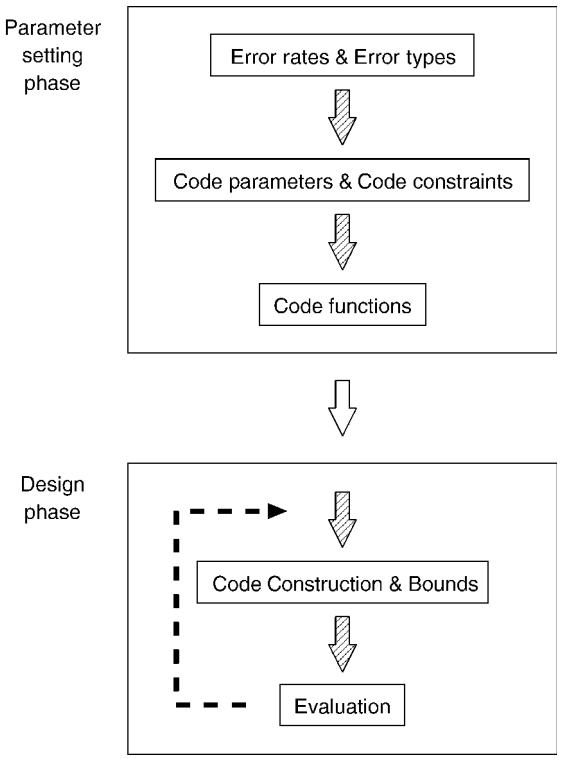


Figure 1.12 *Code design process.*

Step 3. Determine code function:

- Error detection, error correction, error location, or mixed type of these code functions

Step 4. Design code, and calculate code bounds:

- Theoretical bound on code length or check-bit length
- Mathematical knowledge required for code design, for example, algebra, combinatorial mathematics, number theory, graph theory, statistics, and probability theory

Step 5. Evaluate the code designed:

- Check-bit length, and comparison to its bound
- Decoding speed
- Decoder hardware complexity
- Error detection probability of multiple errors beyond the code capability
- If the code does not satisfy the requirements, then go back to step 4

REFERENCES

[ABRA86] J. A. Abraham and W. K. Fuchs, "Fault and Error Models for VLSI," *Proc. IEEE*, 74 (May 1986): 639–654.

[AVIZ78] A. Avizienis, ''Fault Tolerance, the Survival Attribute of Digital Systems'' *Proc. IEEE*, 66 (October 1978): 1109–1125.

[AVIZ04] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, ''Basic Concepts and Taxonomy of Dependable and Secure Computing,'' *IEEE Trans. Depend. Secure Comput.*, 1 (January–March 2004): 11–33.

[AVRE00] D. R. Avresky (ed.), *Dependable Network Computing*, Kluwer Academic Publishers (2000).

[BLAU93] M. Blaum, *Codes for Detecting and Correcting Unidirectional Errors*, IEEE Computer Society Press (1993).

[CALV94]P. Calvel, P. Lamothe, and C. Barillot, ''Space Radiation Evaluation of 16 Mbit DRAMs for Mass Memory Applications,'' *IEEE Trans. Nucl. Sci.*, 41 (December 1994): 2267–2271.

[EZHI86] P. D. Ezhilchevan and S. K. Shrivastava, ''A Characterization of Faults in Systems,'' *Proc. 5th Symp. on Reliability in Distributed Software and Database Systems* (January 1986): 215–222.

[GEFF02] J.-C. Geffroy and G. Motet, *Design of Dependable Computing Systems*, Kluwer Academic Publishers (2002), chs. 1–5.

[HAZU00] P. Hazucha, C. Svensson, and S. A. Wender, ''Cosmic-Ray Soft Error Rate characterization of a Standard 0.6-μm CMOS Process,'' *IEEE J. Solid-State Circ.*, 35 (October 2000): 1422–1429.

[JOHN89] B. W. Johnson, *Design and analysis of Fault Tolerant Digital Systems*, Addison Wesley (1989).

[KANE04] H. Kaneko and E. Fujiwara, ''A Class of *M*-Ary Asymmetric Symbol Error Correcting Codes for Data Entry Devices,'' *IEEE Trans. Comput.*, 53 (February 2004): 159–167.

[KARN04] T. Karnik, P. Hazucha, and J. Patel, ''Characterization of Soft Errors Caused by Single Event Upsets in CMOS Processes,'' *IEEE Trans. Depend. Secure Comput.*, 1 (April–June 2004): 128–143.

[LAPR92] J. C. Laprie (ed.), *Dependability: Basic Concepts and Terminology*, Springer-Verlag (1992).

[LEE90] P. A. Lee and T. Anderson, *Fault Tolerance, Principles and Practice*, Springer-Verlag (1990).

[LEVE95] N. Leveson, *Safeware*, Addison-Wesley (1995).

[LO05] J. C. Lo and E. Fujiwara,''Transient Behavior of the Encoding/Decoding Circuits of Error Control Codes,'' *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems* (October 2005):

[MAKI00] A. Makihara, H. Shindou, N. Nemoto, S. Kuboyama, S. Matsuda, T. Oshima, T. Hirao, H. Itoh, S. Buchner, and A. B. Campbell, ''Analysis of Single-Ion Multiple-Bit Upset in High-Density DRAMs,'' *IEEE Trans. Nucl. Sci.*, 47 (December 2000): 2400–2404.

[MASS96] L. W. Massengill, ''Cosmic and Terrestrial Single-Event Radiation Effects in Dynamic Random Access Memories,'' *IEEE Trans. Nucl. Sci.*, 43 (April 1996): 576–593.

[MAY79] T. C. May, ''Soft Errors in VLSI: Present and Future,'' *IEEE Trans. Comp. Hybrids Manuf. Technol.*, CHMT-2 (December 1979): 377–387.

[MUEL99] M. Mueller, L. C. Alves, W. Fischer, M. L. Fair, and I. Modi, ''RAS Strategy for IBM S/390 G5 and G6,'' *IBM J. Res. Dev.*, 43 (September–November 1999): 875–888.

[NOOR80] D. J. W. Noorlag, L. M. Terman, and A. G. Konheim, ''The Effect of Alpha-Particle-Induced Soft Errors on Memory Systems with Error Correction,'' *IEEE J. Solid-State Circ.*, SC-15 (June 1980): 319–325.

[OGOR96] T. J. O'Gorman, J. M. Ross, A. H. Taber, J. F. Ziegler, H. P. Muhlfeld, C. J. Montrose, H. W. Curtis, and J. L. Walsh, ''Field Testing for Cosmic Ray Soft Errors in Semiconductor Memories,'' *IBM J. Res. Dev.*, 40 (January 1996): 41–50.

[OSAD03] K. Osada, Y. Saitoh, E. Ibe, and K. Ishibashi, ''16.7-fA/Cell Tunnel-Leakage-Suppressed 16-Mb SRAM for Handling Cosmic-Ray-Induced Multierrors,'' *IEEE J. Solid-State Circ.*, 38 (November 2003): 1952–1957.

[OSAD04] K. Osada, K. Yamaguchi, Y. Saitoh, and T. Kawahara, ''SRAM Immunity to Cosmic-Ray-Induced Multierrors Based on Analysis of an Induced Parasitic Bipolar Effect,'' *IEEE J. Solid-State Circ.*, 19 (May 2004): 827–833.

[PRAD86] D. K. Pradhan, *Fault-Tolerant Computing*, Vol. 1 and 2, Prentice-Hall (1986).

[RENN84] D. A. Rennels, ''Fault-Tolerant Computing—Concepts and Examples,'' *IEEE Trans. Comput.*, C-33 (December 1984): 1116–1129.

[SAIH82] G. A. Sai-Halasz, M. R. Wordeman, and R. H. Denard, ''Alpha-Particle-Induced Soft Error Rate in VLSI Circuits,'' *IEEE J. Solid-State Circ.*, SC-17 (April 1982): 355–361.

[SELL68] F. F. Sellers, Jr., M. Y. Hsiao, L. W. Bearnson, *Error Detecting Logic for Digital Computers*, McGraw-Hill (1968).

[SHED78] J. J. Shedletsky, ''Error Correction by Alternate-Data Retry,'' *IEEE Trans. Comput.*, C-27 (February 1978): 106–112.

[SIEW82] D. P. Siewiorek and R. S. Swarz, *The Theory and Practice of Reliable System Design*, Digital Press (1982).

[SRIN96] G. R. Srinivasan, ''Modeling the Cosmic-Ray-Induced Soft-Error Rate in Integrated Circuits: An Overview,'' *IBM J. Res. Dev.*, 40 (January 1996): 77–89.

[WOLF63] J. K. Wolf and B. Elspas, ''Error-Locating Codes—A New Concept in Error Control,'' *IEEE Trans. Info. Theory*, IT-9 (April 1963): 113–117.

[ZIEG96a] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, B. Chin, etc., ''IBM Experiments in Soft Fails in Computer Electronics (1978–1994),'' *IBM J. Res. Dev.*, 40 (January 1996): 3–18.

[ZIEG96b] J. F. Ziegler, ''Terrestrial Cosmic Rays,'' *IBM J. Res. Dev.*, 40 (January 1996): 19–39.

[ZIEG96c] J. F. Ziegler, H. P. Muhlfeld, C. J. Montrose, H. W. Curtis, T. J. O'Gorman, and J. M. Ross, ''Accelerated Testing for Cosmic Soft-Error Rate,'' *IBM J. Res. Dev.*, 40 (January 1996): 51–72.

[ZIEG98] J. F. Ziegler, M. E. Nelson, J. D. Shell, R. J. Peterson, C. J. Gelderloos, H. P. Mahlfeld, and C. J. Montrose, ''Cosmic Ray Soft Error Rates of 16-Mb DRAM Memory Chips,'' *IEEE J. Solid-State Circ.*, 33 (February 1998): 246–252.

CONTENTS

2

Mathematical Background and Matrix Codes

The research in error control codes has relied to a large extent on the powerful structures of modern algebra. A number of important and practical codes based on the structure of rings and Galois fields have been developed. This chapter provides the algebraic structures and the fundamental codes, expressed mostly by matrices, necessary to understand the subsequent chapters and to design codes that fit practical requirements. The level of the discussion is introductory. For a more rigorous treatment, the reader is advised to consult the following excellent texts on coding theory: [PETE72, MACW77, BRAH84, BERL84, PLES98, LIN04].

2.1 INTRODUCTION TO ALGEBRA

The most important ideas in coding theory are based on the arithmetic systems of modern algebra. These systems are not so familiar to most of us, so here we pause to develop a background of this mathematics before we proceed to study coding theory and to design practical codes.

2.1.1 Groups and Rings

A group is a mathematical abstraction of an algebraic structure. A ring is also an abstract set that is an Abelian group and has an additional structure.

Code Design for Dependable Systems: Theory and Practical Applications, by Eiji Fujiwara

Groups

Definition 2.1 Let a, b, and c be the elements of a set $\boldsymbol{G}$ for which an operation $*$ is defined. If $\boldsymbol{G}$ satisfies the following four axioms, then $\boldsymbol{G}$ is called a *group:*

(G1) *Closure*. For every a, b in the set $\boldsymbol{G}$,

$$c = a * b \in \boldsymbol{G}.$$

(G2) *Associativity.* For every a, b, c in the set $\boldsymbol{G}$,

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

(G3) *Identity.* There exists an element e in $\boldsymbol{G}$ called the *identity element* that satisfies

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

(G4) *Inverses*: If a is in the set, then there exists some element b in the set called an *invererse* of a such that

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

□

If the set satisfies the axiom (G1), the set is called a *semigroup*. If the set satisfies the axioms (G1) and (G2), the set is called a *monoid*. Some groups satisfy the additional axiom that for all a, b in the group,

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

This is called a *commutative* axiom. Groups with this additional axiom are called *commutative groups*, or *Abelian groups*. In every group the identity element is unique. Also the inverse of each group element is unique, meaning $(a^{-1})^{-1} = a$. The proofs of these are left to the reader to complete.

Homomorphism and Isomorphism The number of elements in a group is said to be the *order* of the group. A *homomorphism* is a mapping f that preserves the structure of the sets between two groups $\{\boldsymbol{A}, *\}$ and $\{\boldsymbol{A}', *'\}$, meaning $f : \boldsymbol{A} \longrightarrow \boldsymbol{A}'$, which satisfies $f(a * b) = f(a) *' f(b)$ for $a, b \in \boldsymbol{A}$. If there exists a homomorphism from $\boldsymbol{A}$ onto $\boldsymbol{A}'$, then $\boldsymbol{A}'$ is said to be *homomorphic* for $\boldsymbol{A}$. In particular, if there exists one-to-one mapping between $\boldsymbol{A}$ and $\boldsymbol{A}'$, that is, $f' : \boldsymbol{A}' \longrightarrow \boldsymbol{A}$ is also a homomorphism, or *bijection* (*one-to-one* and *onto*), then f is said to be an *isomorphism*, and $\boldsymbol{A}$ and $\boldsymbol{A}'$ are said to be *isomorphic*. The reader is advised to find an isomorphism between the set of integers under addition $\boldsymbol{Z}_4 = \{0, 1, 2, 3\}$ and the multiplicative group $\boldsymbol{G} = \{1, 2, 3, 4\}$.

Subgroups

Definition 2.2 Let $\boldsymbol{G}$ be a group and let $\boldsymbol{F}$ be a subset of $\boldsymbol{G}$. Then $\boldsymbol{F}$ is called a *subgroup* of $\boldsymbol{G}$ if $\boldsymbol{F}$ satisfies all properties of a group with the same operation $*$. □

To prove that a set $\boldsymbol{F}$ is a subgroup of $\boldsymbol{G}$, it is only necessary to check that $a * b$ is in $\boldsymbol{F}$ whenever a and b are in $\boldsymbol{F}$ (i.e., *closure*), and that the inverse of each element in $\boldsymbol{F}$ is also in $\boldsymbol{F}$. The other axioms required of a group will then be inherited from the group $\boldsymbol{G}$.

As an example, we consider the group $\boldsymbol{G} = \{1, 2, 3, 4\}$ under multiplication modulo 5. $\boldsymbol{F} = \{1, 4\}$ is a subgroup of $\boldsymbol{G}$. This is because $1 \cdot 1 = 1, 1 \cdot 4 = 4$, $4 \cdot 1 = 4, 4 \cdot 4 = 1$.

Coset Decomposition This *coset decomposition* illustrates certain relationships between the group $\boldsymbol{G}$ with finite elements, meaning a *finite group*, and the subgroup $\boldsymbol{F}$ with m elements. Let the elements of $\boldsymbol{F}$ be denoted by $a_0, a_1, a_2, \ldots, a_{m-1}$, and choose a_0 to be the identity element. We can construct the array as follows: The first row consists of the elements of $\boldsymbol{F}$, with the identity at the left and every other element of $\boldsymbol{F}$ appearing only once. We choose any element of $\boldsymbol{G}$ not appearing in the first row. We call it b_1 and use it as the first element of the second row. The rest of the elements of the second row are obtained by performing operation $*$ of each subgroup elements by this first element on the left. Similarly we construct a third, fourth, and fifth rows, each time choosing a previously unused group elements for the element in the first column. We stop whenever, after a step, all the group elements, meaning mn elements, appear somewhere in the array. This process stops when $\boldsymbol{G}$ has finite number of elements. The final array is shown as

$$\begin{array}{ccccc}
a_0 = 1 & a_1 & a_2 & \cdots & a_{m-1} \\
b_1 * a_0 = b_1 & b_1 * a_1 & b_1 * a_2 & \cdots & b_1 * a_{m-1} \\
b_2 * a_0 = b_2 & b_2 * a_1 & b_2 * a_2 & \cdots & b_2 * a_{m-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
b_{n-1} * a_0 = b_{n-1} & b_{n-1} * a_1 & b_{n-1} * a_2 & \cdots & b_{n-1} * a_{m-1}
\end{array}$$

The first element on the left of each row is called a *coset leader*. Each row in the array is called a *coset* if the group is Abelian. We can prove that every element of $\boldsymbol{G}$ appears exactly once in a coset decomposition of $\boldsymbol{G}$. We can also prove that the number of elements in $\boldsymbol{F}$ divides the number of elements in $\boldsymbol{G}$ if $\boldsymbol{F}$ is a subgroup of $\boldsymbol{G}$.

Rings The group is an algebra with one operation. We now consider algebra with two operations, namely addition and multiplication.

Definition 2.3 A set is a *ring* $\boldsymbol{R}$ if it satisfies the following axioms:

(R1) $\boldsymbol{R}$ is an Abelian group under addition $(+)$.

(R2) $\boldsymbol{R}$ is closed under multiplication $(\cdot)$.

(R3) $\boldsymbol{R}$ is associative under multiplication.

(R4) (*Distributive law*): Multiplication is distributive with respect to addition, that is, for every $a, b, c \in \boldsymbol{R}$,

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

□

The addition operation in a ring has identity "zero", denoted by 0. But the multiplication operation does not need an identity; if there is an identity, it is unique. This identity is "one" and is denoted by 1. Then $1 \cdot a = a \cdot 1 = a$ for every a in $\boldsymbol{R}$.

A typical example of a ring is a set of integers $\boldsymbol{Z}$ that is commutative.

Definition 2.4 A subset $\boldsymbol{S}$ of a ring $\boldsymbol{R}$ is an *ideal* if it satisfies

(I1) $\boldsymbol{S}$ is a subgroup of $\boldsymbol{R}$ under addition.
(I2) For every $s \in \boldsymbol{S}$ and $r \in \boldsymbol{R}$, $s \cdot r \in \boldsymbol{S}$.

□

In the ring of integer $\boldsymbol{Z}$ all multiples of an integer 3, for example, give an ideal, that is, $\boldsymbol{S} = 3 \cdot \boldsymbol{Z} = \{\cdots, -9, -6, -3, 0, 3, 6, 9, 12, \cdots\}$.

2.1.2 Fields

A ring is a set in which we can add, subtract, and multiply. A more powerful algebraic structure is a *field* in which we can add, subtract, multiply, and divide.

Definition 2.5 A field $\boldsymbol{F}$ is a set that has two operations of addition and multiplication such that the following axioms are defined:

(F1) The set is an Abelian group under addition.
(F2) The field is closed under multiplication, and the set of nonzero elements is an Abelian group under multiplication.
(F3) The distributive law holds for every $a, b, c \in \boldsymbol{F}$,

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

□

A field with finite elements (i.e., q elements) is called a *finite field*, or a *Galois field*, and is denoted by $GF(q)$.

Example 2.1
As a simple example of the finite field with two elements, 0 and 1, meaning $GF(2) = \{0, 1\}$, we have the following addition and multiplication tables:

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

As for $GF(3) = \{0, 1, 2\}$, we have the tables

$+$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

$\cdot$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Another example is $GF(4) = \{0, 1, a, b\}$ in which we have the tables

$+$	0	1	a	b
0	0	1	a	b
1	1	0	b	a
a	a	b	0	1
b	b	a	1	0

$\cdot$	0	1	a	b
0	0	0	0	0
1	0	1	a	b
a	0	a	b	1
b	0	b	1	a

The tables above say that all axioms of the field are satisfied. The readers are recommended to check these. Also carefully to distinguish $GF(4)$ from $\mathbf{Z}_4$, the set of integers modulo 4. In $\mathbf{Z}_4$, $1 + 1 = 2$, whereas in $GF(4)$, $1 + 1 = 0$. Also there does not exist inverse element of 2 in multiplication in $\mathbf{Z}_4$, and hence $\mathbf{Z}_4$ is not a $GF(4)$.

Definition 2.6 Let $\boldsymbol{F}$ be a field. A subset of $\boldsymbol{F}$ is called a *subfield* if it is a field under the inherited addition and multiplication. □

Here the number of elements in the field is called an *order*. The set of elements in the subfield excluding zero element is a subgroup of the original field excluding zero element under multiplication. Therefore, if $GF(p)$ is a subfield of $GF(q)$, then $p - 1$ is a divisor of $q - 1$. Detailed discussion of the subfield is deferred until Subsection 6.2.1.

Definition 2.7 The finite field with prime number p of elements is called a *prime field* $GF(p)$. *Extension field* of the prime field $GF(p)$ is defined as a finite field of $GF(p^m)$ with p^m elements, meaning with mth degree extension of $GF(p)$. □

It can be shown that the set of integers with m elements $\mathbf{Z}_m = \{0, 1, 2, \ldots, m - 1\}$ is a ring and that it is a field if and only if m is a prime. That is, $\mathbf{Z}_p$ with prime p is a prime field, $GF(p)$. This is a relation between the fields and the *integer rings*. Another similar relation exists in *polynomial rings*.

A polynomial over a field $GF(q)$ is expressed as

$$\mathbf{f}(x) = f_{n-1}x^{n-1} + f_{n-2}x^{n-2} + \cdots + f_1x + f_0,$$

where the coefficients f_{n-1}, $f_{n-2}, \ldots, f_1$, f_0 are elements of $GF(q)$, and the indexes and exponents are integers. A *monic polynomial* is a polynomial with leading coefficient f_{n-1} equal to 1. The degree of a nonzero polynomial $\mathbf{f}(x)$, denoted $\deg \mathbf{f}(x)$, is the index of the leading coefficient f_{n-1}. The degree of a nonzero polynomial is always finite. The set of all polynomials over $GF(q)$ is a ring if addition and multiplication are defined as the usual addition and multiplication of polynomials. We define such polynomial ring for each Galois field $GF(q)$. The sum of two polynomials $\mathbf{f}(x)$ and $\mathbf{g}(x)$ in the polynomial ring is another polynomial in this polynomial ring defined by

$$\mathbf{f}(x) + \mathbf{g}(x) = \sum_{i=0}^{\infty} (f_i + g_i)x^i.$$

The degree of the sum is not greater than the larger of these two degrees. For example, over $GF(2)$, $(x^4 + x^2 + x + 1) + (x^3 + x + 1) = x^4 + x^3 + x^2 + (1 + 1)x + (1 + 1) =$

$x^4 + x^3 + x^2$. The product of two polynomials in the polynomial ring is another polynomial in this polynomial ring, defined by

$$\mathbf{f}(x) \cdot \mathbf{g}(x) = \sum_i \left(\sum_{j=0}^{i} f_j g_{i-j} \right) x^i.$$

For example, over $GF(2)$, $(x^4 + x^2 + x + 1) \cdot (x^3 + x + 1) = x^7 + 1$. The degree of a product is equal to the sum of the degrees of the two factors.

We can say that the polynomial $\mathbf{a}(x)$ is divisible by the polynomial $\mathbf{b}(x)$, or that $\mathbf{b}(x)$ is a factor of $\mathbf{a}(x)$, if there is a polynomial $\mathbf{q}(x)$ such that $\mathbf{b}(x) \cdot \mathbf{q}(x) = \mathbf{a}(x)$. A polynomial $\mathbf{p}(x)$ that is divisible only by $\mathbf{p}(x)$ or α, where α is an arbitrary field element in $GF(q)$, is called an *irreducible polynomial*.

The *greatest common divisor* of two polynomials $\mathbf{a}(x)$ and $\mathbf{b}(x)$, denoted by $\mathrm{GCD}(\mathbf{a}(x), \mathbf{b}(x))$, is the monic polynomial of largest degree that divides both of them. The *least common multiple* of two polynomials $\mathbf{a}(x)$ and $\mathbf{b}(x)$, denoted by $\mathrm{LCM}(\mathbf{a}(x), \mathbf{b}(x))$ is the monic polynomial of smallest degree divisible by both of them. If the greatest common divisor of two polynomials is 1, then they are said to be *relatively prime*.

Definition 2.8 For any monic polynomial $\mathbf{p}(x)$ with nonzero degree over the field $\boldsymbol{F}$, the *ring of polynomials modulo* $\mathbf{p}(x)$ is the set of all polynomials with degree smaller than that of $\mathbf{p}(x)$, together with polynomial addition and polynomial multiplication modulo $\mathbf{p}(x)$. □

We can prove that the ring of polynomials modulo $\mathbf{p}(x)$ over $GF(q)$ is an extended field $GF(q^m)$ if and only if $\mathbf{p}(x)$ is a monic irreducible polynomial with m-th degree. As an example, for irreducible polynomial over $GF(2)$ with second degree $\mathbf{p}(x) = x^2 + x + 1$, we have a set $\{0, 1, x, x+1\}$ of $GF(2^2)$. The addition and multiplication tables are as follows:

$+$	0	1	x	$x+1$
0	0	1	x	$x+1$
1	1	0	$x+1$	x
x	x	$x+1$	0	1
$x+1$	$x+1$	x	1	0

$\cdot$	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	$x+1$	1
$x+1$	0	$x+1$	1	x

2.1.3 Representation for Elements of Galois Fields

Polynomial Representation Definition 2.8 mentions that the set of all polynomials with degree smaller than that of the irreducible polynomial $\mathbf{p}(x)$ denotes the ring of polynomials modulo $\mathbf{p}(x)$. That is, by adding zero element to the ring of polynomials modulo $\mathbf{p}(x)$ with degree m over $GF(p)$, we have these elements constitute a finite field of $GF(p^m)$.

Vector Representation Let α be an element of $GF(p^m)$ generated by the irreducible polynomial $\mathbf{p}(x)$ with degree m over $GF(p)$. If $\mathbf{p}(\alpha) = 0$, then α is a root of $\mathbf{p}(x)$. This means that every element in $GF(p^m)$ can be expressed by using α. That is, every element

can be expressed by polynomial of α with degree smaller than or equal to $m-1$. Let an element of $GF(p^m)$ be expressed by $p_{m-1}\alpha^{m-1} + p_{m-2}\alpha^{m-2} + \cdots + p_2\alpha^2 + p_1\alpha + p_0$. Then the coefficient vector of this polynomial can be expressed by $(p_{m-1}, p_{m-2}, \ldots, p_2, p_1, p_0)$ with length m over $GF(p)$. For example, $\{0, 1, \alpha, \alpha+1\}$ is a set of elements in $GF(2^2)$ generated by the primitive polynomial $\mathbf{p}(x) = x^2 + x + 1$ over $GF(2)$, each of which is expressed by a vector with length two over $GF(2)$, that is, $\{(0,0), (0,1), (1,0), (1,1)\}$.

Power Representation A primitive field element of $GF(q)$ is an element α such that every field element except for zero can be expressed as a power of α. For example, in the field $GF(7)$ we have $3^1 = 3$, $3^2 = 2$, $3^3 = 6$, $3^4 = 4$, $3^5 = 5$, $3^6 = 1$, and 3 is a primitive element of $GF(7)$. Primitive elements are used for constructing fields, so we can obtain the elements of fields by multiplying powers of the primitive element.

We can construct $GF(8)$ with the polynomial $\mathbf{p}(x) = x^3 + x + 1$. By using the primitive element $\alpha = x$, we have

$$\begin{aligned}
\alpha &= x,\\
\alpha^2 &= x^2,\\
\alpha^3 &= x + 1,\\
\alpha^4 &= x^2 + x,\\
\alpha^5 &= x^2 + x + 1,\\
\alpha^6 &= x^2 + 1,\\
\alpha^7 &= 1 = \alpha^0.
\end{aligned}$$

In the same way we can construct $GF(16)$ by the polynomial $\mathbf{p}(x) = x^4 + x + 1$. By choosing a special polynomial among irreducible polynomials, called a *primitive polynomial*, we can construct the field. Here the minimum positive integer e that satisfies $\alpha^e = 1$ is called an *order* of α. We can easily prove that $\alpha, \alpha^2, \ldots, \alpha^{e-1}, \alpha^e = 1$ are all distinct. The minimum positive integer e such that the polynomial $\mathbf{p}(x)$ divides $x^e - 1$ is called a *period*, or an *exponent* of $\mathbf{p}(x)$. That is, the period of the irreducible polynomial $\mathbf{p}(x)$ is equal to the order of the root of $\mathbf{p}(x)$. Also it can be proved that there exists an irreducible polynomial with degree m over $GF(p)$ having period $p^m - 1$, called a primitive polynomial with degree m. In other words, an element α of $GF(p^m)$ with order $p^m - 1$ is called a *primitive element* in $GF(p^m)$, and $p^m - 1$ elements of $1, \alpha, \alpha^2, \ldots, \alpha^{p^m-2}$, are all distinct. That is,

$$GF(p^m) = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{p^m-2}\}.$$

For a given polynomial $\mathbf{p}(x)$ with degree m over $GF(2)$, the *reciprocal polynomial* of $\mathbf{p}(x)$, denoted by $\mathbf{p}(x)^*$, is defined as

$$\mathbf{p}(x)^* = x^m \mathbf{p}\left(\frac{1}{x}\right).$$

In this case, $\mathbf{p}(x)^*$ is primitive if and only if $\mathbf{p}(x)$ is primitive.

TABLE 2.1 Primitive Polynomials over *GF* (2)

Degree	Primitive polynomial	Degree	Primitive polynomial
1	$x+1$	17	$x^{17}+x^3+1$
2	x^2+x+1	18	$x^{18}+x^7+1$
3	x^3+x+1	19	$x^{19}+x^5+x^2+x+1$
4	x^4+x+1	20	$x^{20}+x^3+1$
5	x^5+x^2+1	21	$x^{21}+x^2+1$
6	x^6+x+1	22	$x^{22}+x+1$
7	x^7+x+1	23	$x^{23}+x^5+1$
8	$x^8+x^4+x^3+x^2+1$	24	$x^{24}+x^7+x^2+x+1$
9	x^9+x^4+1	25	$x^{25}+x^3+1$
10	$x^{10}+x^3+1$	26	$x^{26}+x^6+x^2+x+1$
11	$x^{11}+x^2+1$	27	$x^{27}+x^5+x^2+x+1$
12	$x^{12}+x^6+x^4+x+1$	28	$x^{28}+x^3+1$
13	$x^{13}+x^4+x^3+x+1$	29	$x^{29}+x^2+1$
14	$x^{14}+x^{10}+x^6+x+1$	30	$x^{30}+x^{23}+x^2+x+1$
15	$x^{15}+x+1$	31	$x^{31}+x^3+1$
16	$x^{16}+x^{12}+x^3+x+1$	32	$x^{32}+x^{22}+x^2+x+1$

Note: Primitive polynomials with minimum number of nonzero coefficients

Table 2.1 shows some primitive polynomials with degree less than or equal to 32. (The reader is referred to Appendix C in [PETE72] for a list of all binary irreducible polynomials with degree less than or equal to 34.) The polynomials provided in Table 2.1 are polynomials with minimum number of nonzero coefficients.

Matrix Representation Each element in $GF(p^m)$ can also be expressed by $m \times m$ matrix over $GF(p)$. The following companion matrix is first defined:

Definition 2.9 Given a primitive polynomial $\mathbf{p}(x)$ of degree m over $GF(p)$, the *companion matrix* $\mathbf{T}$ corresponding to $\mathbf{p}(x)$ is defined as follows:

$$\mathbf{p}(x) = \sum_{i=0}^{m} p_i x^i,$$

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & \cdots & 0 & p_0 \\ & & & & p_1 \\ & \mathbf{I}_{m-1} & & & \vdots \\ & & & & p_{m-1} \end{bmatrix}_{m \times m}$$

$\{p_0, p_1, p_2, \ldots, p_m\}$ $GF(p)$,

$\mathbf{I}_{m-1}$: $(m-1) \times (m-1)$ identity matrix.

□

The set of powered companion matrices and an $m \times m$ zero matrix has the same structure as $GF(p^m)$ and is *field isomorphic* to $GF(p^m)$. Therefore we have

$$\{\mathbf{0}, \mathbf{T}, \mathbf{T}^2, \mathbf{T}^3, \ldots, \mathbf{T}^{p^m-2}, \mathbf{T}^{p^m-1} = \mathbf{I}\} = GF(p^m),$$

where $\mathbf{I}$ is an $m \times m$ identity matrix, and $\mathbf{0}$ is an $m \times m$ zero matrix. The properties of the companion matrix will be mentioned in Subsection 5.1.1.

Example 2.2

Four types of representation for each element of $GF(2^4)$ generated by $\mathbf{p}(x) = x^4 + x + 1$ are presented. The polynomial $\mathbf{p}(x)$ is a primitive polynomial over $GF(2)$. Set $\mathbf{p}(\alpha) = \alpha^4 + \alpha + 1 = 0$. Then $\alpha^4 = \alpha + 1$. From this result we can construct $GF(2^4)$. Some elements of $GF(2^4)$ are

$$\begin{aligned}
\alpha^5 &= \alpha \cdot \alpha^4 = \alpha(\alpha + 1) = \alpha^2 + \alpha, \\
\alpha^8 &= \alpha^4 \cdot \alpha^4 = (\alpha + 1) \cdot (\alpha + 1) = \alpha^2 + 1, \\
\alpha^{13} &= \alpha \cdot \alpha^4 \cdot \alpha^8 = \alpha(\alpha + 1)(\alpha^2 + 1) = \alpha^4 + \alpha^3 + \alpha^2 + \alpha, \\
&= (\alpha + 1) + \alpha^3 + \alpha^2 + \alpha = \alpha^3 + \alpha^2 + 1.
\end{aligned}$$

Table 2.2 shows the previous four representations of the elements of $GF\ (2^4)$.

2.1.4 Properties of Galois Field $GF(2^m)$

It is important to have a knowledge on some properties of basic Galois field of $GF(2)$ and its extension field of $GF(2^m)$ for designing the codes in binary systems.

A polynomial with coefficients from $GF(2)$ may not have roots from $GF(2)$ but has roots from the an extension field of $GF(2)$. For example, $x^4 + x + 1$ is an irreducible polynomial 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 $GF(2^4)$ into $x^4 + x^3 + 1$, we find α^7, α^{11}, α^{13}, and α^{14} are the roots, where α is a root of $x^4 + x + 1$. The reader

TABLE 2.2 Four Representations for Elements of $GF(2^4)$ Generated by $p(x) = x^4+x+1$

Power representation	Polynomial representation	Vector representation	Matrix representation
0	0	(0 0 0 0)	$\mathbf{0}$
1	1	(1 0 0 0)	$\mathbf{T}^0 = \mathbf{I}$
α	α	(0 1 0 0)	$\mathbf{T}$
α^2	α^2	(0 0 1 0)	$\mathbf{T}^2$
α^3	α^3	(0 0 0 1)	$\mathbf{T}^3$
α^4	$1+\alpha$	(1 1 0 0)	$\mathbf{T}^4$
α^5	$\alpha+\alpha^2$	(0 1 1 0)	$\mathbf{T}^5$
α^6	$\alpha^2+\alpha^3$	(0 0 1 1)	$\mathbf{T}^6$
α^7	$1+\alpha+\alpha^3$	(1 1 0 1)	$\mathbf{T}^7$
α^8	$1+\alpha^2$	(1 0 1 0)	$\mathbf{T}^8$
α^9	$\alpha+\alpha^3$	(0 1 0 1)	$\mathbf{T}^9$
α^{10}	$1+\alpha+\alpha^2$	(1 1 1 0)	$\mathbf{T}^{10}$
α^{11}	$\alpha+\alpha^2+\alpha^3$	(0 1 1 1)	$\mathbf{T}^{11}$
α^{12}	$1+\alpha+\alpha^2+\alpha^3$	(1 1 1 1)	$\mathbf{T}^{12}$
α^{13}	$1+\alpha^2+\alpha^3$	(1 0 1 1)	$\mathbf{T}^{13}$
α^{14}	$1+\alpha^3$	(1 0 0 1)	$\mathbf{T}^{14}$

Note: $\mathbf{O} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ $\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ $\mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

should check all these roots. Since these 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 $x^4 + x^3 + 1$. That is,

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

Let $\mathbf{f}(x)$ be a polynomial with coefficients from $GF(2)$. If β, which is an element in $GF(2^m)$, is a root of $\mathbf{f}(x)$, the polynomial $\mathbf{f}(x)$ may have other roots from $GF(2^m)$. That is, β^{2^z} is also a root of $\mathbf{f}(x)$. This is because from $[\mathbf{f}(x)]^{2^z} = \mathbf{f}(x^{2^z})$, and by substituting β into this equation, we have $[\mathbf{f}(\beta)]^{2^z} = \mathbf{f}(\beta^{2^z})$, and then from $\mathbf{f}(\beta) = 0$, we have $\mathbf{f}(\beta^{2^z}) = 0$. Therefore β^{2^z} is also a root of $\mathbf{f}(x)$. The element β^{2^z} is called a *conjugate* of β. For example, the polynomial $\mathbf{f}(x) = x^6 + x^5 + x^4 + x^3 + 1$ has α^4, an element in $GF(2^4)$, as a root. The conjugates of α^4 are

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

That is, α^4, α^8, α, and α^2 are the roots of $\mathbf{f}(x) = x^6 + x^5 + x^4 + x^3 + 1$. We can further check that α^5 and its conjugates α^{10} are roots of $\mathbf{f}(x) = x^6 + x^5 + x^4 + x^3 + 1$. Therefore $\mathbf{f}(x)$ has six distinct roots in $GF(2^4)$.

Let β be a nonzero element in the $GF(2^m)$. In this case $\beta^{2^m-1} = 1$, as was mentioned before. Adding 1 to both sides, we have

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

This says that β is a root of the polynomial $x^{2^m-1} + 1$. Hence the $2^m - 1$ nonzero elements of $GF(2^m)$ form all the roots of $x^{2^m-1} + 1$. This can also be said that the elements of $GF(2^m)$ form all the roots of $x^{2^m} + x$.

Minimal Polynomial 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 $\mathbf{m}(x)$ be the polynomial of smallest degree over $GF(2)$ such that $\mathbf{m}(\beta) = 0$. This polynomial $\mathbf{m}(x)$ is called the *minimal polynomial* of β. Next it can be proved that the minimal polynomial $\mathbf{m}(x)$ is irreducible. If $\mathbf{m}(x)$ is not irreducible, then $\mathbf{m}(x) = \mathbf{m}_1(x)\mathbf{m}_2(x)$, where both $\mathbf{m}_1(x)$ and $\mathbf{m}_2(x)$ have degree larger than 0 and less than the degree of $\mathbf{m}(x)$. Since $\mathbf{m}(\beta) = \mathbf{m}_1(\beta)\mathbf{m}_2(\beta) = 0$, either $\mathbf{m}_1(\beta) = 0$ or $\mathbf{m}_2(\beta) = 0$. This contradicts the hypothesis that $\mathbf{m}(x)$ is a polynomial of smallest degree such that $\mathbf{m}(\beta) = 0$. Therefore $\mathbf{m}(x)$ must be irreducible.

The following is also an important relation as to the minimal polynomial. Let $\mathbf{f}(x)$ be a polynomial over $GF(2)$, and also let $\mathbf{m}(x)$ be the minimal polynomial of a field element β. If β is a root of $\mathbf{f}(x)$, then $\mathbf{f}(x)$ is divisible by $\mathbf{m}(x)$. This is shown as follows: dividing $\mathbf{f}(x)$ by $\mathbf{m}(x)$, we obtain

$$\mathbf{f}(x) = \mathbf{a}(x)\mathbf{m}(x) + \mathbf{r}(x),$$

where the degree of the remainder $\mathbf{r}(x)$ is less than the degree of $\mathbf{m}(x)$. Substituting β into the equation above and using the fact that $\mathbf{f}(\beta) = \mathbf{m}(\beta) = 0$, we have $\mathbf{r}(\beta) = 0$. Hence $\mathbf{r}(x)$ must be zero and $\mathbf{m}(x)$ divides $\mathbf{f}(x)$.

Next we show how to find the minimal polynomial of a field element; Let $\mathbf{m}(x)$ be the minimal polynomial of en element β in $GF(2^m)$, and also let e be smallest integer such that $\beta^{2^e} = \beta$. Then

$$\mathbf{m}(x) = \prod_{i=0}^{e-1}(x + \beta^{2^i}).$$

For example, let's have the minimal polynomial of $\beta = \alpha^3$ in $GF(2^4)$ defined by the primitive polynomial $\mathbf{p}(x) = x^4 + x + 1$. The conjugates of β are $\beta^2 = \alpha^6, \beta^{2^2} = \alpha^{12}$, and $\beta^{2^3} = \alpha^{24} = \alpha^9$. The minimal polynomial of $\beta = \alpha^3$ is

$$\begin{aligned}\mathbf{m}(x) &= (x + \alpha^3)(x + \alpha^6)(x + \alpha^{12})(x + \alpha^9).\\ &= x^4 + x^3 + x^2 + x + 1.\end{aligned}$$

Table 2.3 shows the minimal polynomials of the elements in $GF(2^4)$ generated by $\mathbf{p}(x) = x^4 + x + 1$. Multiplying the polynomials of Table 2.3, we have

$$\begin{aligned}&x \cdot \mathbf{m}_0(x) \cdot \mathbf{m}_1(x) \cdot \mathbf{m}_3(x) \cdot \mathbf{m}_5(x) \cdot \mathbf{m}_7(x)\\ &= x(x+1)(x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)(x^4 + x^3 + 1)\\ &= x^{16} + x.\end{aligned}$$

Thus the roots of $x^{16} + x$ are all the elements of $GF(2^4)$. In general, for α, which is a primitive element in $GF(q)$, we have

$$x^q + x = x\prod_{j=0}^{q-2}(x + \alpha^j).$$

2.2 LINEAR CODES

In this section basic concepts of linear block codes are introduced. First, linear codes are defined as a subspace of vector space and are described in terms of parity-check matrices

TABLE 2.3 Minimal Polynomials of Elements in $GF(2^4)$ Generated by $\mathbf{p(x) = x^4+x+1}$

Conjugate roots	Minimal polynomials	
0	x	
1	$x+1$	$\mathbf{m}_0(x)$
$\alpha\ \alpha^2\ \alpha^4\ \alpha^8$	x^4+x+1	$\mathbf{m}_1(x)$
$\alpha^3\ \alpha^6\ \alpha^{12}\ \alpha^9$	$x^4+x^3+x^2+x+1$	$\mathbf{m}_3(x)$
$\alpha^5\ \alpha^{10}$	x^2+x+1	$\mathbf{m}_5(x)$
$\alpha^7\ \alpha^{14}\ \alpha^{13}\ \alpha^{11}$	x^4+x^3+1	$\mathbf{m}_7(x)$

and generator matrices. The parity check equations for a systematic code are derived. The concept of syndrome is introduced, and the principle of error detection and / or correction by syndrome is discussed. Also we define the minimum distance of a block code, and show that the error correction and / or detection capabilities of the code are determined by its minimum distance. Standard array and its decoding are presented as an introduction to the decoding concept.

2.2.1 Vector Space and Subspace

Definition 2.10 A *vector space* $\boldsymbol{V}$ is a set of vectors over a field F satisfying the following axioms:

(V1) $\boldsymbol{V}$ is an Abelian group under addition.
(V2) $c\boldsymbol{v} \in \boldsymbol{V}$ for all $\boldsymbol{v} \in \boldsymbol{V}$ and for all scalar $c \in F$.
(V3) $c(\boldsymbol{u} + \boldsymbol{v}) = c\boldsymbol{u} + c\boldsymbol{v}$ for all $\boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{V}$ and for all scalar $c \in F$.
(V4) $(a + b)\boldsymbol{v} = a\boldsymbol{v} + b\boldsymbol{v}$ for all $\boldsymbol{v} \in \boldsymbol{V}$ and for all scalars $a, b \in F$.
(V5) $(ab)\boldsymbol{v} = a(b\boldsymbol{v})$, $1\boldsymbol{v} = \boldsymbol{v}$, and $0\boldsymbol{v} = 0$ for all $\boldsymbol{v} \in \boldsymbol{V}$ and for all scalars $a, b \in F$.

□

Axioms (V3) and (V4) are distributive laws, and (V5) is an associative law in the vector space. The vector space plays an important role in coding theory.

Definition 2.11 A subset $\boldsymbol{S}$ of a vector space $\boldsymbol{V}$ over a field F is called a *subspace* if it satisfies all the axioms of a vector space. In order to check whether a subset $\boldsymbol{S}$ of a vector space $\boldsymbol{V}$ is a subspace, it is necessary only to check the following axioms:

(S1) $\boldsymbol{u} + \boldsymbol{v} \in \boldsymbol{S}$ for all $\boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{S}$.
(S2) $c\boldsymbol{v} \in \boldsymbol{S}$ for all $\boldsymbol{v} \in \boldsymbol{S}$ and for all scalar $c \in F$.

□

A *linear combination* of k vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$, is a sum of the form

$$\boldsymbol{u} = c_1\boldsymbol{v}_1 + c_2\boldsymbol{v}_2 + \ldots + c_k\boldsymbol{v}_k,$$

where c_i's are scalars, that is, field elements.

From the above, it can be proved that the set of all linear combinations of a set of k vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$, in $\boldsymbol{V}$ over F is a subspace of $\boldsymbol{V}$.

Linear Dependence / Independence

Definition 2.12 A set of vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$, is *linearly dependent* if and only if there are scalars $c_1, c_2, \ldots, c_k$, not all zeros over F, such that $c_1\boldsymbol{v}_1 + c_2\boldsymbol{v}_2 + \cdots + c_k\boldsymbol{v}_k = 0$.

□

If the set is linearly dependent, then one of these vectors can be obtained as a linear combination of the others. For instance, if $c_k \neq 0$, then we have

$$\begin{aligned}\boldsymbol{v}_k &= c_k^{-1}(c_1\boldsymbol{v}_1 + c_2\boldsymbol{v}_2 + \cdots + c_{k-1}\boldsymbol{v}_{k-1}) \\ &= d_1\boldsymbol{v}_1 + d_2\boldsymbol{v}_2 + \cdots + d_{k-1}\boldsymbol{v}_{k-1}\end{aligned}$$

for $d_i = c_k^{-1}c_i$ and $i = 1, 2, \ldots, k-1$.

Definition 2.13 A set of vectors is *linearly independent* if it is not linearly dependent. That is, $c_1\boldsymbol{v}_1 + c_2\boldsymbol{v}_2 + \cdots + c_k\boldsymbol{v}_k \neq 0$.

□

In such a set, any one of those cannot be obtained as a linear combination of the others.

Definition 2.14 A set of vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$, is said to *span* a vector space $\boldsymbol{V}$ if and only if every vector in $\boldsymbol{V}$ equals a linear combination of the vectors in the set.

□

In this case, if a set of vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$, span $\boldsymbol{V}$, then any set that is linearly independent in $\boldsymbol{V}$ can have at most k vectors. Readers are recommended to prove this. This states that there can be at most k linearly independent vectors in a vector space that is spanned by k vectors. Therefore, if k linearly independent vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$ span $\boldsymbol{V}$, then k is said to be the *dimension* of $\boldsymbol{V}$ and the set $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$ is a *basis* of $\boldsymbol{V}$. Also in a k-dimensional space any basis must be exactly of k vectors, and the vectors are linearly independent. If $\boldsymbol{V}$ is any k-dimensional vector space, then any set of k linearly independent vectors is a basis of $\boldsymbol{V}$.

Orthogonality and Null Spaces

Definition 2.15 An *inner product* or *dot product* of two k-tuple vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ is a scalar and is defined as follows:

$$\boldsymbol{u} \cdot \boldsymbol{v} = (u_1\ u_2\ \ldots\ u_k) \cdot (v_1\ v_2\ \ldots\ v_k) = u_1v_1 + u_2v_2 + \cdots + u_kv_k.$$

□

It is easily verified that $\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{v} \cdot \boldsymbol{u}$ and that $\boldsymbol{w} \cdot (\boldsymbol{u} + \boldsymbol{v}) = \boldsymbol{w} \cdot \boldsymbol{u} + \boldsymbol{w} \cdot \boldsymbol{v}$ for any k-tuple vectors $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{w}$.

Definition 2.16 If the inner product of two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ is zero, that is, $\boldsymbol{u} \cdot \boldsymbol{v} = \boldsymbol{v} \cdot \boldsymbol{u} = 0$, then they are said to be *orthogonal*.

□

Let $\boldsymbol{S}_1$ be a k-dimensional subspace of $\boldsymbol{V}_n$, where $\boldsymbol{V}_n$ is the vector space of all n-tuple vectors, such that for any $\boldsymbol{u} \in \boldsymbol{S}_2$, $\boldsymbol{u}$ is orthogonal to every vector in $\boldsymbol{S}_1$. Then the set of vectors $\boldsymbol{S}_2$ orthogonal to every vector in subspace $\boldsymbol{S}_1$ of the vector space $\boldsymbol{V}_n$ is also a subspace. In this case $\boldsymbol{S}_2$ is called the *null space* of $\boldsymbol{S}_1$. If $\boldsymbol{S}_2$ is the null space of $\boldsymbol{S}_1$, then it is easy to see that $\boldsymbol{S}_1$ is the null space of $\boldsymbol{S}_2$. The dimension of $\boldsymbol{S}_2$ is given by the following: if $\boldsymbol{S}_2$ is the null space of a subspace $\boldsymbol{S}_1$ with dimension k in $\boldsymbol{V}_n$, then $\boldsymbol{S}_2$ has dimension $n - k$.

2.2.2 Linear Codes as Vector Spaces

Let the vector space $\boldsymbol{V}_n$ of all n-tuples over $GF(q)$ be expressed as

$$\boldsymbol{V}_n = \{(a_0\ a_1\ a_2\ \ldots\ a_{n-1}) \mid a_i \in GF(q)\}.$$

We define linear codes as follows:

Definition 2.17 A subset $\boldsymbol{C}$ of $\boldsymbol{V}_n$ is a *linear code* over $GF(q)$ if and only if it is a subspace. □

That is, for all $\mathbf{w}_0$, $\mathbf{w}_1 \in \boldsymbol{C}$, and for all $c_0, c_1 \in GF(q)$ if

$$c_0\mathbf{w}_0 + c_1\mathbf{w}_1$$

is also included in $\boldsymbol{C}$, then $\boldsymbol{C}$ is called a linear code over $GF(q)$.

For $q = 2$, a word $\mathbf{0} = (0, 0, \ldots, 0)$ is a codeword in $\boldsymbol{C}$ over $GF(2)$. Also, if $\mathbf{w}_1 + \mathbf{w}_2$ is a codeword for all $\mathbf{w}_1$, $\mathbf{w}_2 \in \boldsymbol{C}$, then $\boldsymbol{C}$ is a linear code. As a simple example, set of $\{(0\,0\,0), (0\,1\,1), (1\,0\,1), (1\,1\,0)\}$ is a linear code because $\mathbf{0} = (0\,0\,0)$ is included in the set, and sum of any two codewords is also a codeword.

Using k codewords $\mathbf{w}_0$, $\mathbf{w}_1, \ldots, \mathbf{w}_{k-1}$ in $\boldsymbol{C}$, a linear combination of these, that is,

$$\mathbf{w} = c_0\mathbf{w}_0 + c_1\mathbf{w}_1 + \cdots + c_{k-1}\mathbf{w}_{k-1}, \qquad c_0, c_1, \ldots, c_{k-1} \in GF(q), \tag{2.1}$$

is a codeword of $\boldsymbol{C}$ over $GF(q)$ from the definition of the linear code. Here we assume that $\mathbf{w}_0$, $\mathbf{w}_1, \ldots, \mathbf{w}_{k-1}$, are linearly independent over $GF(q)$. Then, if any codeword of $\boldsymbol{C}$ can be expressed by Eq. (2.1), we call $\{\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_{k-1}\}$ a *basis* of the linear code $\boldsymbol{C}$. The number of codewords k in the basis is called a *dimension* of the linear code $\boldsymbol{C}$, denoted $\dim(\boldsymbol{C})$. In the simple example above, there are three kinds of basis, $\{(0\,1\,1), (1\,0\,1)\}$, $\{(0\,1\,1), (1\,1\,0)\}$, $\{(1\,0\,1), (1\,1\,0)\}$, and $\dim(\boldsymbol{C}) = 2$. It can be proved that since $\boldsymbol{w}_0$, $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{k-1}$, are linearly independent, the coefficients $c_0, c_1, \ldots, c_{k-1}$, are uniquely determined for each codeword. Therefore the q-ary linear code with dimension $\dim(\boldsymbol{C}) = k$ has q^k distinct codewords.

2.2.3 Matrix Algebra

Basic knowledge of matrix algebra is important for designing matrix codes. This subsection presents a brief introduction to matrix algebra, and includes the important matrices that will be applied to block codes in later chapters.

An $n \times m$ matrix is an ordered set of nm elements over $GF(q)$ in a rectangular array of n rows and m columns:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots\cdots & a_{n,m} \end{bmatrix} = [a_{i,j}],$$

where $i = 1, 2, \ldots, n$, and $j = 1, 2, \ldots, m$.

The n rows can be thought of as n m-tuples or vectors, and similarly, the m columns can be thought of as m n-tuples or vectors. The set of elements $a_{i,j}$ for which the column number and row number are equal is called the *main diagonal*.

The *row space* of an $n \times m$ matrix is the set of all linear combinations of row vectors of the matrix. They form a subspace of the vector space of m-tuples. The dimension of the row space is called the *row rank*. Similarly the set of all linear combinations of column vectors of the matrix forms the *column space*, whose dimension is called the *column rank*. It can be shown that if row rank equals column rank, then this is referred to as *rank of the matrix*.

Elementary Row Operations There is a set of *elementary row operations* defined for matrices:

(a) Interchange any two rows.
(b) Multiply any row by a nonzero element over $GF(q)$.
(c) Add any multiple of one row to another.

It can be proved that if one matrix is obtained from another by a succession of elementary operations, both matrices have the same row space. Two matrices are said to be *row equivalent* if their row spaces are the same. These row operations are useful for designing the matrix codes.

Echelon Canonical Form Elementary row operations obtain a simplified form of matrix that is a standard form. This form is called an *echelon canonical form* and has the following characteristics:

1. Every leading element of a nonzero row is 1.
2. Every column containing such a leading element has all its other entries zero.
3. The leading element 1 of any row is to the right of the leading element in every preceding row. All zero rows are below the nonzero rows.

Example 2.3
For the following matrix over $GF(3)$, we go through six elementary row operations:

$$\begin{bmatrix} 2 & 0 & 1 & 2 & 1 & 0 & 1 \\ 1 & 2 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 2 & 1 & 1 & 2 \end{bmatrix}.$$

Step 1. Multiply the first row by 2:

$$\begin{bmatrix} 1 & 0 & 2 & 1 & 2 & 0 & 2 \\ 1 & 2 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 2 & 1 & 1 & 2 \end{bmatrix}.$$

Step 2. Add two times the first row to the second:

$$\begin{bmatrix} 1 & 0 & 2 & 1 & 2 & 0 & 2 \\ 0 & 2 & 2 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 2 & 1 & 1 & 2 \end{bmatrix}.$$

Step 3. Multiply the second row by 2:

$$\begin{bmatrix} 1 & 0 & 2 & 1 & 2 & 0 & 2 \\ 0 & 1 & 1 & 0 & 2 & 2 & 2 \\ 0 & 1 & 1 & 2 & 1 & 1 & 2 \end{bmatrix}.$$

Step 4. Add two times the second row to the third row:

$$\begin{bmatrix} 1 & 0 & 2 & 1 & 2 & 0 & 2 \\ 0 & 1 & 1 & 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 2 & 2 & 2 & 0 \end{bmatrix}.$$

Step 5. Multiply the third row by 2:

$$\begin{bmatrix} 1 & 0 & 2 & 1 & 2 & 0 & 2 \\ 0 & 1 & 1 & 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

Step 6. Add two times the third row to the first:

$$\begin{bmatrix} 1 & 0 & 2 & 0 & 1 & 2 & 2 \\ 0 & 1 & 1 & 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

In step 6 we finally get the echelon canonical form of the matrix. The leading elements of the above final matrix are in column positions 1, 2, and 4. Nonzero rows of a matrix in echelon canonical form are linearly independent, and thus the number of nonzero rows is the dimension of the row space. Note that in the final matrix there are no zero rows, and hence the dimension of the row space is 3.

In the above example we can get a 3×3 identity matrix at the left of the final matrix by exchanging columns 3 and 4, that is,

$$\begin{bmatrix} 1 & 0 & 0 & 2 & 1 & 2 & 2 \\ 0 & 1 & 0 & 1 & 2 & 2 & 2 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

This form of matrix is called a *reduced echelon canonical form*, or *systematic form*. Note that systematic form cannot always be obtained only by row operations. Sometimes column operations are also required. This form of parity-check matrix is essential for encoding the input information, which will be discussed in later chapters.

A systematic form of the parity-check matrix can have the identity matrix at the right most position. Or the weight-1 columns of the identity matrix can be distributed among other columns, as will be seen in the matrix codes in later chapters.

Nonsingular Matrix

Definition 2.18 The square matrix is said to be *nonsingular* if it has an inverse matrix, or if the determinant of the matrix is nonzero. □

This can be said another way, from an algebraic standpoint, that if the rows or columns of an $n \times n$ square matrix are linearly independent, the matrix is said to be nonsingular. It is apparent that an $n \times n$ identity matrix is nonsingular. The companion matrix defined by

Definition 2.9 is also nonsingular. Any nonsingular matrix can be transformed into an identity matrix by elementary row operations.

It can be proved that if $\mathbf{M}$ is an $n \times m$ matrix and $\mathbf{S}$ is a nonsingular $n \times n$ matrix, then the product of $\mathbf{S}$ and $\mathbf{M}$ has the same row space as $\mathbf{M}$ has. The nonsingular matrices will be extensively used for designing matrix codes in later chapters.

Transposed Matrix The *transpose* of an $n \times m$ matrix $\mathbf{M}$ is an $m \times n$ matrix, denoted $\mathbf{M}^T$, whose rows are the columns of $\mathbf{M}$ and thus whose columns are the rows of $\mathbf{M}$.

Vandermonde Matrix The following square matrix with elements a_i's from the finite or infinite field is called a *Vandermonde matrix*, and its determinant is nonzero if the elements a_i's are distinct:

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ a_1 & a_2 & a_3 & \cdots & a_n \\ {a_1}^2 & {a_2}^2 & {a_3}^2 & \cdots & {a_n}^2 \\ \vdots & \vdots & \vdots & & \vdots \\ {a_1}^{n-1} & {a_2}^{n-1} & {a_3}^{n-1} & \cdots & {a_n}^{n-1} \end{bmatrix} n \times n$$

Since the determinant of this square matrix with distinct a_i's is nonzero, the matrix is nonsingular. This matrix will be used again in later chapters, in particular, in Chapter 7.

2.2.4 Distance and Error Control Capability

We begin with the important concepts of Hamming weight and Hamming distance.

Definition 2.19 **The** *Hamming weight* of a vector $\mathbf{u} = (u_0, u_1, \ldots, u_{n-1})$, denoted by $w(\mathbf{u})$, is the number of nonzero elements of $\mathbf{u}$. □

Definition 2.20 **The** *Hamming distance* between two vectors $\mathbf{u}$ and $\mathbf{v}$, denoted by $d(\mathbf{u}, \mathbf{v})$, is the Hamming weight of $\mathbf{u} - \mathbf{v}$. The Hamming distance also equals the number of positions by which the two vectors differ. That is,

$$\begin{aligned} d(\mathbf{u}, \mathbf{v}) &= w(\mathbf{u} - \mathbf{v}) = w(\mathbf{v} - \mathbf{u}) \\ &= \text{number of differing positions of } \mathbf{u} \text{ and } \mathbf{v}. \end{aligned} \tag{2.2}$$

□

The Hamming distance is a *metric* in the sense that it is a real number satisfying the following:

(1) $d(\mathbf{u}, \mathbf{v}) > 0$ for $\mathbf{u} \neq \mathbf{v}$ (positive definiteness)
$= 0$ for $\mathbf{u} = \mathbf{v}$
(2) $d(\mathbf{u}, \mathbf{v}) = d(\mathbf{v}, \mathbf{u})$ (symmetry)
(3) $d(\mathbf{u}, \mathbf{v}) + d(\mathbf{v}, \mathbf{w}) \geq d(\mathbf{u}, \mathbf{w})$ (triangle inequality).

The Hamming distance and the Hamming weight can often help us understand the error control capability of a code. When the codeword $\mathbf{v}$ is transmitted, and hence we receive the

erroneous word $\mathbf{r}$, the Hamming distance between $\mathbf{v}$ and $\mathbf{r}$, meaning $d(\mathbf{v}, \mathbf{r})$, is equal to the number of errors. The Hamming weight of the error vector $\mathbf{e} = \mathbf{r} - \mathbf{v}$, meaning $w(\mathbf{e})$, provides the number of errors.

Definition 2.21 **The** *minimum Hamming distance* (or *minimum distance*) $d_{\min}$ of a code $\boldsymbol{C}$ is the minimum of the distances between all pairs of codewords. □

Since vectors $\mathbf{u}$ and $\mathbf{v}$ are codewords in a linear code in Eq. (2.2), then $\mathbf{u} - \mathbf{v}$ or $\mathbf{v} - \mathbf{u}$ is also another codeword. Therefore the minimum distance of a linear code is equal to the minimum weight of its nonzero codewords.

The minimum distance of a code is an important parameter by which we decide the error control capability of the code. As was mentioned before, every linear code contains a zero codeword.

Now we consider $\boldsymbol{V}$, the set of all n-tuples over $GF(2)$, and let a subset $\boldsymbol{C} \subset \boldsymbol{V}$ be a code with minimum distance d. The codeword is used as a transmitted word. At the receiver, the received word is checked to see whether or not it is a codeword. If it is a codeword, then it is accepted; otherwise, it is considered to be an erroneous word, and the errors are detected.

Let us consider the relation between the minimum distance and the code capability.

Error Detection If the code has the minimum distance at least d, then the code detects any error pattern of weight $d - 1$ or less. No pattern of $d - 1$ or fewer errors can change the transmitted codeword into another codeword. This is because $d_{\min}$ is d. Therefore such errors can be detected. This means that the code with $d_{\min} = d$ guarantees detection of $d - 1$ or fewer errors.

Error Correction If the code has the minimum distance $d_{\min}$ larger than or equal to $2t + 1$, then the code can correct all patterns of t or fewer errors. Here we consider the n-dimensional spheres of radius t for each codeword as its center. These spheres are all disjoint, and for any t or fewer errors from a codeword, the erroneous words are present within the bounds of the respective sphere. On the other hand, if the minimum distance is less than $2t + 1$, the t-error patterns have cases to result in a received word at least as close to an incorrect codeword as it is to the correct codeword. From these, as far as any erroneous words are present within the bounds of the sphere, these can be recovered to the correct words.

Error Correction and Detection If the minimum distance of the code $d_{\min}$ is larger than or equal to $t + d + 1$, then the code can correct any combination of t errors and detect up to d errors where d is larger than or equal to t. We consider again spheres of radius t from each codeword as its center. For a codeword, t or fewer errors cause the received word to fall within its own sphere. That is, these errors can be corrected. Any errors larger than t but less than $d(\geq t)$ apart from the center of the spheres can be detected because the received words are outside all these spheres. The spheres are basically all disjoint, because if there is an intersection containing a word, then this word is, at most, distance t from two codewords, whereas the two codewords are apart by more than $2t + 1$. This violates triangle inequality, and is therefore impossible. Similarly, if the number of errors are larger than t but not larger than d, then the received word is outside these spheres, so the errors can be detected but not corrected. The above relations between the minimum distance and the code capability are illustrated in Figure 2.1.

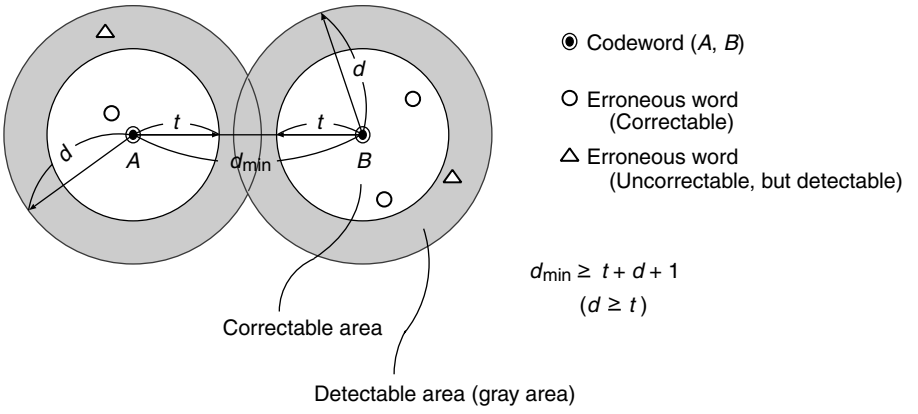


Figure 2.1 *Decoding spheres.*

Erasure Correction Erasures usually correspond to detected signals that are considered to be in a certain "no-confidence zone." In the binary systems the erasure zone is intermediate between the 1-zone and the 0-zone. In general, an erasure implies an unknown bit or symbol at a known location. With a t-error correcting code, any pattern of $2t$ erasures is correctable. This follows from the fact that with $2t$ erasures any two n-tuple words resulting from different substitutions can differ at most $2t$ positions. In this case a t-error correcting code has a distance at least $2t + 1$, which means these n-tuple words cannot both be codewords. In general, error control codes with minimum Hamming distance $d_{\min} = d$ can correct $d - 1$ erasures. For example, code $\boldsymbol{C}$ with $d_{\min} = 3$ and $n = 3$, meaning $\boldsymbol{C} = \{111, 000\}$, so at most two erasures can be corrected as shown in the following:

111	000
x11	x00
1x1	0x0
11x	00x
xx1	xx0
x1x	x0x
1xx	0xx

That is, any received word in a column of the above array can be decoded by simply going to the top word, which is the codeword of the column.

In real systems, erasures are often compounded with nonerasure errors. There is a tradeoff between the number of correctable errors and erasures. For example, a multiple error correcting code is capable of correcting any combination of t errors and e erasures as long as the minimum distance of the code $d_{\min}$ is at least $2t + e + 1$.

2.2.5 Parity-Check Matrices for Linear Codes

In block coding, the binary information sequence is segmented into message block with fixed length of k information bits. This means that there are 2^k distinct messages. According to certain rules, the encoder transforms each input message into a binary n-tuple word where $n > k$. This binary n-tuple word is referred to as a *codeword*. Therefore there are 2^k

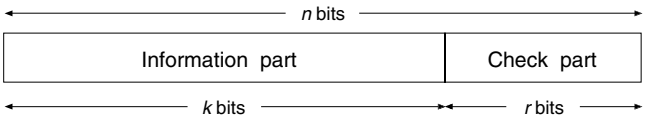


Figure 2.2 *Systematic structure of codeword.*

codewords corresponding to the 2^k possible messages. This set of codewords is called a *block code*. There is one-to-one correspondence between a message and its codeword.

Definition 2.22 A block code with length n and having 2^k codewords is called a *linear* (n, k) *code* if and only if its 2^k codewords form a k-dimensional subspace of the vector space of all the n-tuples over $GF(2)$. □

A binary block code is *linear* if and only if the modulo-2 sum of two codewords is also a codeword. Codeword of a linear block code has the *systematic structure* as shown in Figure 2.2 where a codeword is divided into two parts, the *information part* and the *check part*. The information part consists of k information bits and the check part consists of $r = n - k$ *parity-check bits*. Each check bit is determined by linear sum of *information bits*. A linear block code having this structure is called a *linear systematic block code*.

Parity-Check Matrix Here we define the $r \times k$ binary *encoding matrix* $\mathbf{H}_e$ that determines r check bits by linear sum of information bits. That is, the encoding matrix determines which information bits are added over $GF(2)$ in order to generate check bits.

$$\mathbf{H}_e = \begin{bmatrix} h_{0,0} & h_{0,1} & \cdots\cdots & h_{0,k-1} \\ h_{1,0} & h_{1,1} & \cdots\cdots & h_{1,k-1} \\ \vdots & \vdots & \ddots & \vdots \\ h_{r-1,0} & h_{r-1,1} & \cdots & h_{r-1,k-1} \end{bmatrix},$$

$$h_{i,j} \in GF(2), \qquad 0 \le i \le r-1, \qquad 0 \le j \le k-1.$$

Let the vector with r check bits be denoted as $\mathbf{c} = (c_0\ c_1\ \ldots\ c_{r-1})$ and the vector with k information bits be as $\mathbf{d} = (d_0\ d_1\ d_2\ \ldots\ d_{k-1})$. Then r check bits are determined by the following relation:

$$\mathbf{c} = \mathbf{d} \cdot \mathbf{H}_e{}^T,$$

where $\mathbf{H}_e{}^T$ means transpose of $\mathbf{H}_e$. By appending these r check bits to the input k information bits, the n-bit codeword $\mathbf{v}$ can be generated, meaning $\mathbf{v} = [\mathbf{d}\ \mathbf{c}]$. That is,

$$\begin{aligned} c_0 &= d_0 h_{0,0} + d_1 h_{0,1} + \cdots + d_{k-1} h_{0,k-1} \\ c_1 &= d_0 h_{1,0} + d_1 h_{1,1} + \cdots + d_{k-1} h_{1,k-1} \\ &\cdots \\ c_{r-1} &= d_0 h_{r-1,0} + d_1 h_{r-1,1} + \cdots + d_{k-1} h_{r-1,k-1}, \end{aligned}$$

and finally we have a codeword of $\mathbf{v} = (d_0\, d_1 \ldots d_{k-1}\ c_0\, c_1 \ldots c_{r-1})$.

Next we define the $r \times n$ matrix $\mathbf{H}$, where the $r \times r$ identity matrix $\mathbf{I}_r$ is appended to the $r \times k$ matrix $\mathbf{H}_e$, meaning $\mathbf{H} = [\mathbf{H}_e \ \mathbf{I}_r]$. This matrix is called a *parity-check matrix* or an $\mathbf{H}$ *matrix* that expresses the systematic error control code. The codes expressed by the parity-check matrix are called *matrix codes* in this book.

$$\mathbf{H} = [\mathbf{H}_e \quad \mathbf{I}_r] = \begin{bmatrix} \mathbf{h}_0 \\ \mathbf{h}_1 \\ \vdots \\ \mathbf{h}_{r-1} \end{bmatrix} = \begin{bmatrix} h_{0,0} & h_{0,1} & \cdots & h_{0,k-1} & 1 & 0 & \cdots & 0 \\ h_{1,0} & h_{1,1} & \cdots & h_{1,k-1} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h_{r-1,0} & h_{r-1,1} & \cdots & h_{r-1,k-1} & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{matrix} \uparrow \\ \\ \\ \downarrow \end{matrix} r,$$

$$\overleftarrow{\qquad\qquad\qquad\qquad n \qquad\qquad\qquad\qquad}$$

$$\overleftarrow{\qquad\qquad k \qquad\qquad}\overleftarrow{\qquad r \qquad}$$

$$\mathbf{h}_i = (h_{i,0}\ h_{i,1} \cdots h_{i,k-1} \overbrace{0\,0 \cdots 0\,1}^{i}\ 0 \cdots 0),$$

$$\overleftarrow{\qquad k \qquad}\overleftarrow{\qquad r \qquad}$$

$$h_{i,j} \in GF(2), \qquad 0 \le i \le r-1, \quad 0 \le j \le k-1.$$

From the above, we have the following important relation:

$$\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}, \tag{2.3}$$

where $\mathbf{v}$ is an n-bit codeword and $\mathbf{0}$ is an r-bit zero row vector. That is, we obtain

$$d_0 h_{j,0} + d_1 h_{j,1} + \cdots + d_{k-1} h_{j,k-1} + c_j = 0 \quad \text{for} \quad j = 0, 1, \cdots, r-1.$$

Therefore an (n, k) linear code is completely specified by its parity-check matrix $\mathbf{H}$. There exists an $r \times n$ matrix $\mathbf{H}$ such that an n-tuple vector $\mathbf{v}$ is a codeword in $\boldsymbol{C}$ if and only if $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$.

The linear block codes do not necessarily have the previous systematic form of $\mathbf{H}$. In this case nonsystematic form of $\mathbf{H}$ can be transformed into systematic form by performing the elementary row operation shown in the previous Subsection 2.2.3.

There exist important relations between $\mathbf{H}$ matrix column vectors, linear dependence / independence, and minimum Hamming distance $d_{\min}$.

Theorem 2.1 *For any codeword* $\boldsymbol{v}$ *having weight* d *in a linear code* $\boldsymbol{C}$*,* d *columns of its* $\boldsymbol{H}$ *matrix are linearly dependent.*

It can easily be proved that d columns of $\mathbf{H}$ are linearly dependent because only d elements of $\mathbf{v}$ are nonzero in Eq. (2.3).

Note that the distance of the code is also its minimum weight and every column in $\mathbf{H}$ is a nonzero vector. The minimum weight is d if and only if no $d-1$ or fewer columns of $\mathbf{H}$ are linearly dependent. From this, the following theorem holds:

Theorem 2.2 *A linear code* $\boldsymbol{C}$ *has minimum distance* d_{min} *if and only if every* $d_{min}-1$ *or fewer columns of its* $\boldsymbol{H}$ *matrix are linearly independent.*

Theorem 2.2 is important not only in determining the distance of a code but also in constructing the matrix codes with distance d.

Example 2.4
The following $\mathbf{H}$ matrix over $GF(3)$ is assumed to be given:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 2 & 2 & 2 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 \end{bmatrix}.$$

First, we note that every column is nonzero vector. If the received vector $\mathbf{r}$ is of weight 1, then $\mathbf{r} \cdot \mathbf{H}^T$ is equal to the column vector pattern corresponding to the nonzero element in $\mathbf{r}$, and hence is nonzero. Therefore $\mathbf{r}$ is not a codeword. Consider $\mathbf{r} = (0001000200000)$ with weight two, meaning $w(\mathbf{r}) = 2$. Then $\mathbf{r} \cdot \mathbf{H}^T = (121) + 2(102) = (022) \neq (000)$. Therefore $\mathbf{r}$ is not a codeword. If two columns are linearly dependent, then one column vector is a multiple of another column vector; this is not the case in the $\mathbf{H}$ matrix. Therefore there cannot be a codeword with weight two. If $\mathbf{r} = (0001000201000)$, then $\mathbf{r} \cdot \mathbf{H}^T = (121) + 2(102) + (011) = (000)$, so $\mathbf{r}$ is a codeword. That is, these three columns of $\mathbf{H}$ are linearly dependent. Therefore we conclude that the minimum distance of the example code is 3.

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

$$\mathbf{v} = d_0\mathbf{g}_0 + d_1\mathbf{g}_1 + \cdots + d_{k-1}\mathbf{g}_{k-1},$$

where $d_i \in \{0, 1\}$ for $i = 0, 1, \cdots, k-1$. These k linearly independent codewords are arranged as the rows of a $k \times n$ matrix as follows:

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & g_{0,1} & \cdots & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \cdots & g_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1,0} & g_{k-1,1} & \cdots & g_{k-1,n-1} \end{bmatrix},$$

where $\mathbf{g}_i = (g_{i,0}\ g_{i,1} \ldots g_{i,n-1})$ for $0 \leq i \leq k-1$. Therefore the codewords can be generated by the following relation:

$$\begin{aligned} \mathbf{v} &= \mathbf{d} \cdot \mathbf{G} \\ &= (d_0\, d_1 \cdots d_{k-1}) \cdot \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_{k-1} \end{bmatrix}. \end{aligned}$$

The rows of $\mathbf{G}$ span the code space and hence generate the (n, k) linear code $\boldsymbol{C}$. Any k linearly independent codewords of $\boldsymbol{C}$ can be used to form this matrix. From this relation, the matrix $\mathbf{G}$ is called a *generator matrix* for $\boldsymbol{C}$.

Let us turn to the relation between the matrices $\mathbf{H}$ and $\mathbf{G}$. From Eq. (2.3) any vector in the row space of $\mathbf{G}$, meaning any codeword, is orthogonal to the rows of $\mathbf{H}$. That is, any vector that is orthogonal to the rows of $\mathbf{H}$ is in the row space of $\mathbf{G}$. This implies that

$$\mathbf{G} \cdot \mathbf{H}^T = \mathbf{0}.$$

In other words, the $\mathbf{H}$ matrix can be generated by the $\mathbf{G}$ matrix, and also any linear code can be expressed by the $\mathbf{H}$ matrix.

Syndrome Let $\mathbf{v} = (d_0\, d_1 \ldots d_{k-1}\; c_0\, c_1 \ldots c_{r-1})$ be a codeword that is transmitted under noisy circumstances, that is, transmitted through the medium of air, storage, line, and so forth, that may cause some errors. Also let $\mathbf{r} = (d'_0\; d'_1 \ldots d'_{k-1}\; c'_0\; c'_1 \ldots c'_{r-1})$ be the *received word* at the output of the medium that may contain errors. Because of the errors, $\mathbf{r}$ may be different from $\mathbf{v}$. If the errors are added to $\mathbf{v}$, then we have the vector sum

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

where $\mathbf{e} = (e_0\; e_1 \ldots e_{n-1})$ is an n-tuple vector and is called an *error vector* or an *error pattern*. In this case $e_i = 1$ means that the value of i-th element of $\mathbf{r}$ is not equal to that of the corresponding i-th element of $\mathbf{v}$, that is , error is existed in the i-th element of $\mathbf{r}$. On the other hand, $e_i = 0$ means that both i-th elements of $\mathbf{r}$ and $\mathbf{v}$ have an equal value, that is, there exists no error in the i-th element of $\mathbf{r}$.

Upon receiving $\mathbf{r}$, the decoder determines whether $\mathbf{r}$ contains errors or not, and then takes the action of detection, location, or correction if errors are existed. When $\mathbf{r}$ is received, the decoder performs the following computation:

$$\begin{aligned}\mathbf{S} &= \mathbf{r} \cdot \mathbf{H}^T \\ &= (S_0\; S_1 \ldots S_{r-1}).\end{aligned}$$

The output of $\mathbf{S}$ is called the *syndrome* of $\mathbf{r}$. If $\mathbf{S} = \mathbf{0}$, then $\mathbf{v}$ is a codeword, and there exist no errors in $\mathbf{r}$; if $\mathbf{S} \neq \mathbf{0}$, then $\mathbf{r}$ is not a codeword. The syndrome $\mathbf{S}$ computed from the received vector $\mathbf{r}$ depends only on the error pattern $\mathbf{e}$, and not on the transmitted codeword $\mathbf{v}$. This is because $\mathbf{r}$ is the vector sum of $\mathbf{v}$ and $\mathbf{e}$. So it follows from the above computation that

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

Since $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$ from Eq. (2.3), the syndrome $\mathbf{S}$ depends only on $\mathbf{e}$ as

$$\mathbf{S} = \mathbf{e} \cdot \mathbf{H}^T. \tag{2.4}$$

Syndrome Decoding for Standard Array Once we have calculated the syndrome, we need to identify the original codeword, which is the actual transmitted codeword, from

the syndrome. Because it helps in understanding the decoding concept, we introduce the standard array. The standard array is the same as the array in the coset decomposition discussed in Subsection 2.1.1 whose elements are n-tuple words over $GF(2)$ and the operation $*$ is an addition on $GF(2)$.

Let $\boldsymbol{C}$ be an (n, k) linear code. Let $\mathbf{v}_0$, $\mathbf{v}_1, \ldots, \mathbf{v}_{2^k-1}$ be the codewords of $\boldsymbol{C}$. No matter what codeword is transmitted under noisy circumstances, the received word $\mathbf{r}$ will be any of the 2^n n-tuple words over $GF(2)$. The decoding scheme at the receiver uses a rule to partition the 2^n possible received words into disjoint subsets $\mathbf{D}_0$, $\mathbf{D}_1, \ldots, \mathbf{D}_{2^k-1}$ such that the codeword $\mathbf{v}_i$ is contained in the subset $\mathbf{D}_i$ for $0 \leq i \leq 2^k - 1$. Thus each subset $\mathbf{D}_i$ has one-to-one correspondence to a codeword $\mathbf{v}_i$. If the received word $\mathbf{r}$ is found in the subset $\mathbf{D}_i$, then $\mathbf{r}$ is decoded into $\mathbf{v}_i$. Correct decoding is performed if and only if the received word $\mathbf{r}$ is in the subset $\mathbf{D}_i$ that corresponds to the actual codeword transmitted.

The method to partition the 2^n possible received words into 2^k disjoint subsets depends on the coset decomposition concept. Each subset contains one and only one codeword. The partition is based on the linear structure of the code. The 2^k codewords of $\boldsymbol{C}$ are placed in a row including an all-zero codeword $\mathbf{v}_0 = (0\ 0\ 0\ \ldots\ 0)$ in the first left-most position. From the remaining $2^n - 2^k$ n-tuple words, an n-tuple $\mathbf{e}_1$ is chosen and is placed under $\mathbf{v}_0$. Next a second row is formed by adding $\mathbf{e}_1$ to each codeword $\mathbf{v}_i$ in the first row and placing the sum $\mathbf{e}_1 + \mathbf{v}_i$ under $\mathbf{v}_i$. Having completed the second row, an unused n-tuple $\mathbf{e}_2$ is chosen from the remaining n-tuple words and is placed under $\mathbf{e}_1$. A third row is formed by adding $\mathbf{e}_2$ to each codeword $\mathbf{v}_i$ in the first row and placing $\mathbf{e}_2 + \mathbf{v}_i$ under $\mathbf{v}_i$. We continue this process until all the n-tuple words are used. Then we have an array of rows and columns as shown in Figure 2.3. This array is called a *standard array* of the given linear code $\boldsymbol{C}$.

The construction rule of a standard array says that the sum of any two words in the same row is a codeword in $\boldsymbol{C}$. Also it can be proved that no two n-tuple words in the same row are identical. Also every n-tuple word appears in one and only one row. There are then

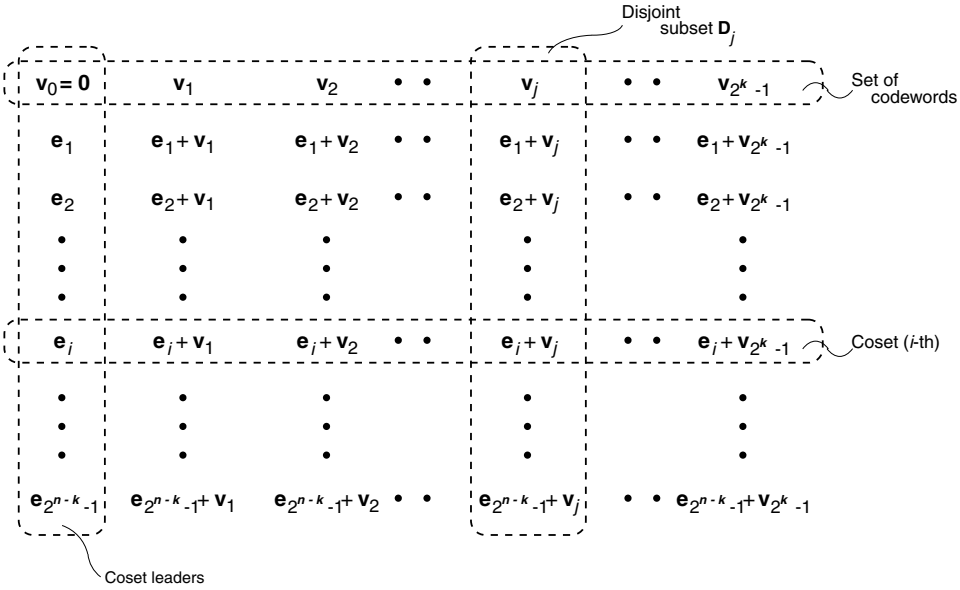


Figure 2.3 *Standard array for an (n, k) linear code.*

Coset leaders								Syndromes
$\mathbf{v}_0$	$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$	$\mathbf{v}_4$	$\mathbf{v}_5$	$\mathbf{v}_6$	$\mathbf{v}_7$	
000000	100110	010101	001011	110011	101101	011110	111000	000
$\mathbf{e}_1$: 100000	000110	110101	101011	010011	001101	111110	011000	110
$\mathbf{e}_2$: 010000	110110	000101	011011	100011	111101	001110	101000	101
$\mathbf{e}_3$: 001000	101110	011101	000011	111011	100101	010110	110000	011
$\mathbf{e}_4$: 000100	100010	010001	001111	110111	101001	011010	111100	100
$\mathbf{e}_5$: 000010	100100	101111	001001	110001	101111	011100	111010	010
$\mathbf{e}_6$: 000001	100111	010100	001010	110010	101100	011111	111001	001
$\mathbf{e}_7$: 100001	000111	110100	101010	010010	001100	111111	011001	111

Figure 2.4 *Standard array for (6, 3) code.*

$2^n/2^k = 2^{n-k} = 2^r$ distinct rows in the standard array, and each row consists of 2^k distinct words. These 2^r rows are called *cosets* of the code $\boldsymbol{C}$ and the first n-tuple $\mathbf{e}_j$ of each coset is called a *coset leader*, mentioned in Subsection 2.1.1.

Example 2.5
We consider the $(6, 3)$ linear code expressed by the following $\mathbf{H}$ matrix:

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

The standard array of this code is shown in Figure 2.4.

A standard array of an (n, k) linear code $\boldsymbol{C}$ consists of 2^k disjoint columns, meaning 2^k disjoint subsets. Each column consists of $2^{n-k} = 2^r$ n-tuple words, with the topmost one as a codeword in $\boldsymbol{C}$. Let $\mathbf{D}_j$ denote the j-th column of the standard array. Then

$$\mathbf{D}_j = \{\mathbf{v}_j \quad \mathbf{e}_1 + \mathbf{v}_j \quad \mathbf{e}_2 + \mathbf{v}_j \quad \ldots \quad \mathbf{e}_{2^{n-k}-1} + \mathbf{v}_j\}, \tag{2.5}$$

where $\mathbf{v}_j$ is a codeword of $\boldsymbol{C}$ and $\mathbf{e}_0(= \mathbf{v}_0), \mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_{2^{n-k}-1}$, are the coset leaders. The 2^k distinct columns $\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_{2^k-1}$, can be used for decoding the code $\boldsymbol{C}$. Assume that the codeword $\mathbf{v}_j$ is transmitted. From Eq. (2.5) the received word $\mathbf{r}$ is in $\mathbf{D}_j$ if the error pattern is a coset leader. In this case the received word $\mathbf{r}$ is decoded correctly into the transmitted codeword $\mathbf{v}_j$. On the other hand, if the error pattern is not a coset leader, an erroneous decoding will be performed. That is, the decoding is correct if and only if the error pattern is a coset leader. For this reason the $2^{n-k} = 2^r$ coset leaders including the zero codeword are called the *correctable error patterns*. Every word included in the same coset has the same syndrome, and in addition no two syndromes in different cosets are equal. Therefore every (n, k) linear block code is capable of correcting 2^{n-k} error patterns.

The decoding the received word in the standard array is performed in the following steps:

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

Step 2. Locate the coset leader $\mathbf{e}_i$ whose syndrome is equal to $\mathbf{r} \cdot \mathbf{H}^T$. Then $\mathbf{e}_i$ is assumed to be the error pattern.

Step 3. Decode the received word $\mathbf{r}$ into the codeword $\mathbf{v} = \mathbf{r} + \mathbf{e}_i$.

2.3 BASIC MATRIX CODES

In this section the typical error control codes are introduced. These codes are basic to the design of practical codes that should fit the requirements of future applications.

As was mentioned before, a linear code can be expressed by a parity-check matrix or by a generator polynomial. Here we will use matrix form of expression for the basic codes.

2.3.1 Simple Parity-Check Codes

In digital systems parity check is usually used to detect errors because this requires only one check bit and is implemented by very simple encoder / decoder hardware. Parity check has been extensively applied to logic systems and memory systems, including data-path logic circuits, arithmetic logic circuits, high-speed memories, and so forth. Among the variety of error control codes the *simple parity-check code* is the easiest to use, and it is first presented precisely to help the reader understand the matrix code.

A parity-check bit is determined to make the total number of 1's in a codeword even. For example, assume that an eight-bit input word $\mathbf{d} = (0\,1\,1\,1\,0\,1\,0\,1)$ is given. Since $\mathbf{d}$ includes five 1's, a parity bit p is determined to be 1 in order to make the total number of 1's even. The parity bit p is appended to $\mathbf{d}$, and this results in the codeword $\mathbf{v} = (\mathbf{d}\ p) = (0\,1\,1\,1\,0\,1\,0\,1\,1)$ having even number of 1's. For this reason a simple parity-check code is sometimes called an *even parity code*. Alternatively, this encoding procedure can also be performed for odd number of 1's; then it is called an *odd parity code*.

The above encoding procedure can be expressed in mathematical form. That is, for a given k-bit input word $\mathbf{d} = (d_0\ d_1\ \ldots\ d_{k-1})$, where $d_i \in GF(2)$, $0 \le i \le k-1$, a parity bit p is generated by

$$\begin{aligned} p &= \mathbf{d} \cdot \mathbf{H}_e^T, \\ &= d_0 + d_1 + \cdots + d_{k-1}, \end{aligned}$$

where $\mathbf{H}_e = (1\,1 \ldots 1)$ with the row vector consisting of k 1's. The calculation by addition is performed over $GF(2)$, so the "+" denotes modulo-2 addition. The codeword of the simple parity-check code $\boldsymbol{C}$ is $\mathbf{v} = (\mathbf{d}\ p) = (d_0\ d_1\ \ldots\ d_{k-1}\ p)$.

The decoding procedure is as follows: for the received word $\mathbf{r}$ with $k+1$ bits, meaning $\mathbf{r} = (\mathbf{d}'\ p') = (d_0'\ d_1'\ \ldots\ d_{k-1}'\ p')$, a parity check is performed as

$$\begin{aligned} \mathbf{S} &= \mathbf{r} \cdot \mathbf{H}^T \\ &= (d_0'\ d_1' \cdots d_{k-1}'\ p') \cdot (1\,1 \cdots 1\,1\ )^T \\ &= d_0' + d_1' + \cdots + d_{k-1}' + p', \end{aligned}$$

where $\mathbf{H}$ is a row vector with $k+1$ 1's, or

$$\mathbf{H} = [\underbrace{1\,1\ \cdots\ 1\,1}_{k+1}]. \tag{2.6}$$

Calculation is then performed over $GF(2)$, and $\mathbf{S}$ is the parity-checked output, called a *syndrome*. If an error exists in the received word, the error $\mathbf{e} = (e_0, e_1, \ldots, e_{k-1}, e_p)$,

where $e_i \in GF(2)$, $(i = 0, 1, \ldots, k-1, p)$, is added to the transmitted word $\mathbf{v}$. Hence $\mathbf{r} = \mathbf{v} + \mathbf{e} = (d_0 + e_0 \;\; d_1 + e_1 \;\; \ldots \;\; d_{k-1} + e_{k-1} \;\; p + e_p)$, where $r_i = d_i' = d_i + e_i$ and $r_p = p' = p + e_p$ for $0 \leq i \leq k-1$. Then the syndrome $\mathbf{S}$ is expressed as

$$\begin{aligned}\mathbf{S} &= \mathbf{r} \cdot \mathbf{H}^T \\ &= (\mathbf{v} + \mathbf{e}) \cdot \mathbf{H}^T = \mathbf{v} \cdot \mathbf{H}^T + \mathbf{e} \cdot \mathbf{H}^T \\ &= \mathbf{e} \cdot \mathbf{H}^T = e_0 + e_1 + \cdots + e_{k-1} + e_p \qquad (\because\ \mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}).\end{aligned} \tag{2.7}$$

In the above equations, $\mathbf{S} = 1$ indicates an error detection if there exists an odd number of 1's in $\mathbf{r}$, or in $\mathbf{e}$. Therefore addition over $GF(2)$ turns out to be 1.

Equation (2.7) indicates that there exists a one-to-one correspondence between an element of $\mathbf{r}$ and a column of $\mathbf{H}$. An element "1" in $\mathbf{H}$ means that the corresponding element of $\mathbf{r}$ is added over $GF(2)$, that is, *checked*.

On this basis the simple parity-check code can be defined as a matrix code by the following definition:

Definition 2.23 A $(k+1, k)$ *simple parity-check code* is defined and expressed by $1 \times (k+1)$ parity-check matrix having $k+1$ 1's, that is, one row vector composed of $k+1$ 1's. □

Let us explore the error control capability of a simple parity-check code $\boldsymbol{C}$. It can be easily understood that the minimum Hamming distance of $\boldsymbol{C}$ is two, meaning $d_{\min} = 2$. From Subsection 2.2.4 we know that a simple parity-check code can detect single-bit errors. The following theorem, however, clarifies the error detection capability of the simple parity-check code.

Theorem 2.3 *Simple parity-check codes detect any odd number of bit errors but never detect even number of bit errors.*

Theorem 2.3 can be easily proved by Eq. (2.7). Recall that an odd number of 1's in $\mathbf{e}$, meaning $w(\mathbf{e}) =$ odd number, always leads to $\mathbf{S} = 1$, whereas an even number of 1's in $\mathbf{e}$ always leads to $\mathbf{S} = 0$. Theorem 2.3 tells us that only one additional check bit to the original input word will lead to an error detection of any odd number of bit errors. This is why the simple parity-check codes are so efficient and hence so much in use in digital systems.

The simple parity-check code is usually expressed over $GF(2)$. The corresponding nonbinary check code is called a *checksum code* over integer set modulo q, meaninig $\mathbf{Z}_q$, where q is an integer larger than 2. One check symbol is determined by sum modulo q of the other information symbols of the input word. This code will be mentioned in more detail in Subsection 12.3.2.

2.3.2 Hamming Single Error Correcting (SEC) Codes

From Subsection 2.2.4 we know that a code with minimum Hamming distance $d_{\min} = 3$, called *distance-3 code*, can correct single errors or detect double errors in the received word. Further from Theorem 2.2 of Subsection 2.2.5 we have that any two column vectors in the $\mathbf{H}$ matrix (i.e., $\mathbf{H}_{\text{SEC}}$) of the distance-3 code are linearly independent. The distance-3

codes can be used either as single error correcting (SEC) codes or as double error detecting (DED) codes. Here we will study the single error correcting Hamming code [HAMM50].

Definition 2.24 An (n, k) single error-correcting code is expressed by an $(n-k) \times n$ **H** matrix in which any two nonzero column vectors are linearly independent. □

Example 2.6
The binary $(7, 4)$ SEC code expressed by the following **H** matrix:

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

The corresponding codeword $\mathbf{v} = (d_0\ d_1\ d_2\ d_3\ c_0\ c_1\ c_2)$ has four data bits in the former four positions in **v** and three check bits in the latter three positions. The check bits are computed by the following equations over $GF(2)$:

$$\begin{aligned} c_0 &= d_0 + d_1 + d_2, \\ c_1 &= d_0 + d_2 + d_3, \\ c_2 &= d_0 + d_1 + d_3. \end{aligned}$$

At the start of decoding the following syndrome calculation is performed for the received word $\mathbf{r} = (d'_0\ d'_1\ d'_2\ d'_3\ c'_0\ c'_1\ c'_2)$:

$$\begin{aligned} S_0 &= d'_0 + d'_1 + d'_2 + c'_0, \\ S_1 &= d'_0 + d'_2 + d'_3 + c'_1, \\ S_2 &= d'_0 + d'_1 + d'_3 + c'_2. \end{aligned}$$

The syndrome obtained from the calculation above is denoted as $\mathbf{S} = (S_0\ S_1\ S_2)$, which indicates the erroneous bit positions if single-bit errors occur. That is, if the binary syndrome pattern is identical to a particular binary column vector in **H**, then the bit corresponding to the column vector is determined to be in error. The indicated bit is inverted and finally corrected. Suppose that in the example code above the second bit d_1 is in error. We then obtain the syndrome $\mathbf{S} = (1\,0\,1)$, which is identical to the second column vector in **H**. Therefore the second bit in **r** is inverted and finally corrected. Figure 2.5 shows the encoder and the decoder of this code, where $\widehat{d_0}$, $\widehat{d_1}$, $\widehat{d_2}$, and $\widehat{d_3}$ are the decoded output data.

If double-bit errors occur, there is the risk of *miscorrection*. In the $(7, 4)$ code of this example, if the d_2 and d_3 bits are in error, then $\mathbf{S} = (110) + (011) = (101)$, which indicates that the d_1 bit is in error. So the d_1 bit will be corrected, that is, miscorrected. In this example code, if the second column is deleted from the $\mathbf{H}_{\text{SEC}}$, which then expresses a $(6, 3)$ SEC code, the double-bit errors given above will lead to no miscorrection. In general, if the nonzero syndrome pattern is not identical to any column vectors in $\mathbf{H}_{\text{SEC}}$, then the errors can be detected.

From the example above we can easily design the binary SEC code. The **H** matrix of the binary SEC code is constructed by choosing distinct nonzero binary column vectors.

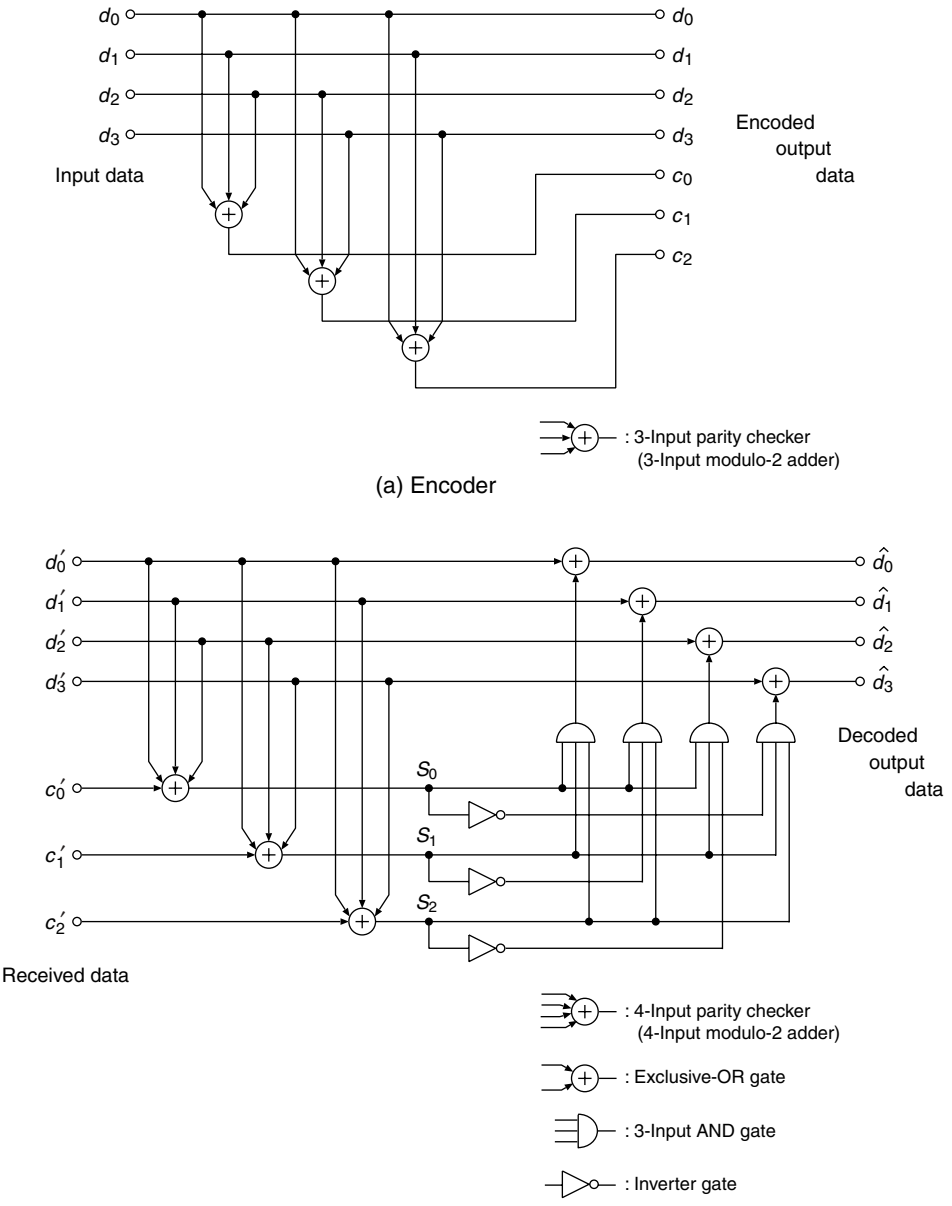


Figure 2.5 *Encoder and decoder of the Hamming (7, 4) SEC code.*

The maximum number of these columns is $2^r - 1$. In other words, the maximum code length of a binary (n, k) SEC code is $n = 2^{n-k} - 1$, where $n - k = r$. The code whose code length is less than the maximum length is called a *shortened code*. The shortened code has a possibility to detect some additional errors beyond the guaranteed error control capability of the code.

The (n, k) binary SEC codes, in general, have the following code parameters:

Check-bit length $r = n - k$.
Maximum code length in bits $n = 2^r - 1$.
Maximum information-bit length (or data length in bits) $k = n - r = 2^r - 1 - r$.

2.3.3 Hamming Single Error Correcting and Double Error Detecting (SEC-DED) Codes

The distance-4 code is a single error correcting and double error detecting code that is called a Hamming *SEC-DED code* [HAMM50]. The code is designed by adding a simple parity check of n bits to the SEC codes. That is, the **H** matrix of this code is designed by adding a row vector with all 1's, and also adding a weight-1 column vector with upper $r - 1$ all 0's to the $(r - 1) \times (n - 1)$ **H** matrix of the binary SEC code. The **H** matrix, denoted as $\mathbf{H}_{\text{SEC}-\text{DED}}$, is written as

$$\mathbf{H}_{\text{SEC-DED}} = \begin{bmatrix} \boxed{\mathbf{H}_{\text{SEC}}} & \begin{matrix} 0 \\ 0 \\ \vdots \\ 0 \end{matrix} \\ \underbrace{1 \quad 1 \quad \cdots \quad 1}_{n-1} & \underbrace{1}_{1} \end{bmatrix} \begin{matrix} \uparrow \\ \downarrow \end{matrix} r - 1 \quad , \tag{2.8}$$

where $\mathbf{H}_{\text{SEC}}$ is an **H** matrix of a single-error correcting (SEC) code, and $r = n - k$.

Definition 2.25 An (n, k) single-bit error correcting and double-bit error detecting (SEC-DED) code is defined and expressed by an $r \times n$ **H** matrix shown in Eq.(2.8). □

Unlike the SEC code this code requires an extra overall parity check of its encoding and decoding. In the encoding the following overall parity bit c_r is generated and appended to the remaining $r - 1$ generated check bits:

$$c_{r-1} = d_0 + d_1 + \cdots + d_{k-1} + c_0 + c_1 + \cdots + c_{r-2}.$$

The syndrome calculation is performed in the same way by making a parity calculation:

$$S_{r-1} = d'_0 + d'_1 + \cdots + d'_{k-1} + c'_0 + c'_1 + \cdots + c'_{r-1}.$$

The decoding of the SEC-DED codes is performed by the calculated syndrome $\mathbf{S} = (S_0, S_1, \ldots, S_{r-1})$ as follows:

Step 1. If $\mathbf{S} = \mathbf{0}$, then the received word is error-free.

Step 2. If $\mathbf{S} \neq \mathbf{0}$ and $S_{r-1} = 1$, then an odd number of errors, likely single-bit errors, have occurred. If the syndrome pattern is identical to a column vector in **H**, the corresponding bit is in error and then inverted, that is, corrected. If the syndrome pattern is not identical

to any column vector in **H**, then three or the larger odd number of errors is detected. That is, uncorrectable errors are detected.

Step 3. If $\mathbf{S} \neq \mathbf{0}$ and $S_{r-1} = 0$, then the even number of errors, likely double-bit errors, are detected. As a result uncorrectable errors are detected.

A modified Hamming SEC-DED code will be discussed in Chapter 4. It is constructed by using all odd-weight-column vectors in the **H** matrix. That is, there is no overall parity check. This code is called an *odd-weight-column SEC-DED code*, or a *modified Hamming SEC-DED code*, which brings smaller decoder hardware and higher decoding speed than the Hamming SEC-DED code [HSIA70].

It can be easily proved that the (n, k) binary SEC-DED codes have the maximum code length in bits $n = 2^{n-k-1}$.

2.3.4 Cyclic Codes

Cyclic codes are a class of typical linear polynomial codes. Encoding and decoding are performed mainly by *linear feedback shift register* (*LFSR*) circuits and therefore operated serially bit by bit. There are efficient cyclic codes for detecting and / or correcting random errors, burst errors, and byte errors. They have inherent algebraic structure, and hence there exist various serial decoding methods. Usually the codes are expressed by generator polynomials.

Algebraic Structure of Cyclic Codes First, we consider the codeword of a linear cyclic code $\boldsymbol{C}$ over $GF(q)$ as $\mathbf{v} = (v_{n-1}\ v_{n-2}\ \ldots\ v_2\ v_1\ v_0)$, called a code vector, where $v_j \in GF(q)$, $j = 0, 1, \ldots, n-1$. In a cyclic code this n-tuple vector is expressed by polynomial over $GF(q)$. That is,

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

This is called a *code polynomial* of **v**. There exists a one-to-one correspondence between the code vector **v** and the code polynomial $\mathbf{v}(x)$. Here we have another vector $\mathbf{v}^{(i)}$ that is cyclically shifted i places to the left in the previous n-tuple vector **v**, called a cyclic i-shift of **v**:

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

The code polynomial that corresponds to the code vector $\mathbf{v}^{(i)}$ is

$$\begin{aligned}\mathbf{v}^{(i)}(x) = {} & v_{n-i-1}x^{n-1} + v_{n-i-2}x^{n-2} + \cdots + v_2 x^{i+2} + v_1 x^{i+1} + v_0 x^i \\ & + v_{n-1}x^{i-1} + v_{n-2}x^{i-2} + \cdots + v_{n-i+1}x + v_{n-i}.\end{aligned} \tag{2.9}$$

Definition 2.26 An (n, k) linear code $\boldsymbol{C}$ is called a *cyclic code* if every cyclic shift of a code vector in $\boldsymbol{C}$ is also a code vector in $\boldsymbol{C}$. □

The code polynomial corresponding to $\mathbf{v}^{(i)}$ is expressed as

$$[x^i \mathbf{v}(x)] \mod x^n - 1.$$

That is,

$$[x^i \mathbf{v}(x)] \mod x^n - 1 = \mathbf{v}^{(i)}(x). \tag{2.10}$$

Since cyclic code is a linear code, the linear combination of the code polynomials is also a code polynomial. That is, for an arbitrary positive integer r and $w_i \in GF(q)$,

$$\sum_{i=1}^{r} w_i [x^i \mathbf{v}(x)] \bmod x^n - 1 = [\mathbf{w}(x)\mathbf{v}(x)] \bmod x^n - 1.$$

Here $\sum_{i=1}^{r} w_i x^i = \mathbf{w}(x)$, which is an arbitrary polynomial with degree r over $GF(q)$.

Next we consider the code polynomial $\mathbf{g}(x)$ with minimum degree r. Let $\mathbf{b}(x)$ be a residue of $\mathbf{v}(x)$ divided by $\mathbf{g}(x)$. Then the following relation holds:

$$\mathbf{v}(x) = \mathbf{a}(x)\mathbf{g}(x) + \mathbf{b}(x),$$

where $\mathbf{a}(x)$ is a quotient polynomial and the degree of $\mathbf{b}(x)$ is less than that of $\mathbf{g}(x)$. The degree of $\mathbf{a}(x)\mathbf{g}(x)$ is equal to that of $\mathbf{v}(x)$, less than or equal to $n-1$. From the previous discussion, since $\mathbf{g}(x)$ is a code polynomial, then $\mathbf{a}(x)\mathbf{g}(x)$ is also a code polynomial. The polynomial $\mathbf{b}(x)$ can be expressed by

$$\mathbf{b}(x) = \mathbf{v}(x) - \mathbf{a}(x)\mathbf{g}(x).$$

This says that $\mathbf{b}(x)$ is equal to the difference of two code polynomials, which is also a code polynomial. That is, $\mathbf{b}(x)$ is a code polynomial. This contradicts that the degree of $\mathbf{b}(x)$ is less than that of $\mathbf{g}(x)$. Therefore $\mathbf{b}(x) = 0$. So the following theorem characterizes an important property of a cyclic code.

Theorem 2.4 *Let $\mathbf{g}(x)$ be the nonzero code polynomial with minimum degree in an (n,k) cyclic code $\boldsymbol{C}$. A binary polynomial with degree $n-1$ or less is a code polynomial if and only if it is a multiple of $\mathbf{g}(x)$.*

The polynomial $\mathbf{g}(x)$ is called a *generator polynomial* with degree r of the (n,k) cyclic code $\boldsymbol{C}$. The polynomial $\mathbf{g}(x)$ has the following properties:

1. $\mathbf{g}(x)$ is a factor of $x^n - 1$.
2. $\mathbf{g}(x) = g_r x^r + g_{r-1}x^{r-1} + \cdots + g_1 x + g_0$ for $\{g_r, g_{r-1}, \ldots, g_1, g_0\} \in GF(q)$ is a monic polynomial with $g_r = 1$ uniquely determined, and $g_0 = 1$.
3. Since the number of polynomials with degree $n-1$ or less is a multiple of $\mathbf{g}(x)$ is q^{n-r} and there are q^k code polynomials (or codewords) in $\boldsymbol{C}$, then we have $r = n - k$.

Hence the nonzero code polynomial of minimum degree in an (n,k) cyclic code has the following form:

$$\mathbf{g}(x) = x^{n-k} + g_{n-k-1}x^{n-k-1} + \cdots + g_2 x^2 + g_1 x + 1, \tag{2.11}$$

where $\{g_{n-k-1}, \ldots, g_2, g_1\} \in GF(q)$.

Next the generator matrix and the parity-check matrix of the (n, k) cyclic code generated by $\mathbf{g}(x)$ can be easily formed. Dividing x^{n-k+i} by $\mathbf{g}(x)$ for $i = 0, 1, \ldots, k-1$, we obtain

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

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

$$\mathbf{b}_i(x) = b_{i,n-k-1}x^{n-k-1} + \cdots + b_{i,1}x + b_{i,0}.$$

Since $x^{n-k+i} + \mathbf{b}_i(x)$ for $i = 0, 1, \ldots, k-1$, are multiple of $\mathbf{g}(x)$, they are code polynomials. Arranging these k code polynomials as rows of a $k \times n$ matrix, we obtain the generator matrix of $\boldsymbol{C}$ in systematic form

$$\mathbf{G} = \begin{bmatrix} 0 & \cdots & 0 & 0 & 1 & b_{0,n-k-1} & \cdots & b_{0,1} & b_{0,0} \\ 0 & \cdots & 0 & 1 & 0 & b_{1,n-k-1} & \cdots & b_{1,1} & b_{1,0} \\ 0 & \cdots & 1 & 0 & 0 & b_{2,n-k-1} & \cdots & b_{2,1} & b_{2,0} \\ \vdots & \iddots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\ 1 & \cdots & 0 & 0 & 0 & b_{k-1,n-k-1} & \cdots & b_{k-1,1} & b_{k-1,0} \end{bmatrix} \begin{matrix} \uparrow \\ \\ \\ \\ \downarrow \end{matrix} k \,.$$
$$\underbrace{\qquad\qquad}_{k} \qquad \underbrace{\qquad\qquad\qquad\qquad}_{n-k}$$

The corresponding $(n-k) \times n$ parity-check matrix of $\boldsymbol{C}$ can be written as follows:

$$\mathbf{H} = \begin{bmatrix} b_{k-1,0} & \cdots & b_{2,0} & b_{1,0} & b_{0,0} & 0 & \cdots & 0 & 0 & 1 \\ b_{k-1,1} & \cdots & b_{2,1} & b_{1,1} & b_{0,1} & 0 & \cdots & 0 & 1 & 0 \\ b_{k-1,2} & \cdots & b_{2,2} & b_{1,2} & b_{0,2} & 0 & \cdots & 1 & 0 & 0 \\ \vdots & & \vdots & \vdots & \vdots & \vdots & \iddots & \vdots & \vdots & \vdots \\ b_{k-1,n-k-1} & \cdots & b_{2,n-k-1} & b_{1,n-k-1} & b_{0,n-k-1} & 1 & \cdots & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \uparrow \\ \\ \\ \\ \downarrow \end{matrix} n-k \,.$$
$$\underbrace{\qquad\qquad\qquad\qquad}_{k} \qquad \underbrace{\qquad\qquad}_{n-k}$$

Let $\mathbf{h}(x)$ be a *parity-check polynomial* satisfying the equation

$$\mathbf{h}(x) = \frac{x^n - 1}{\mathbf{g}(x)}.$$

Since arbitrary code polynomial is expressed by $\mathbf{v}(x) = \mathbf{a}(x)\mathbf{g}(x)$, we have the following relation:

$$\mathbf{h}(x)\mathbf{v}(x) = \mathbf{a}(x)(x^n - 1).$$

That is, $[\mathbf{h}(x)\mathbf{v}(x)] \bmod (x^n - 1) = 0$. Therefore we have $n-k$ parity equations and obtain the preceding parity-check matrix $\mathbf{H}$.

Error Detection by Cyclic Codes Encoding / decoding of the cyclic codes is easily and efficiently implemented by a linear feedback shift register, abbreviated LFSR. This type of

error detection is described in many coding theory books such as [PETE72, BRAH84, BERL84, LIN04]. Error detection by the cyclic codes is performed whether or not the received polynomial, in which the received word is expressed in polynomial, is divided by the generator polynomial. If it is not divided, then the received word is not a codeword because the received polynomial is not the code polynomial. This type of error detection is called a *cyclic check*, and it is used in many digital systems. The following shows a widely used cyclic code recommended by CCITT. The generator polynomial is written as

$$\mathbf{g}(x) = x^{16} + x^{12} + x^5 + 1. \tag{2.12}$$

Burst errors are detected by using the cyclic codes. The burst error with L-bit length that starts at the i-th bit position of the word is expressed by the *error polynomial* $\mathbf{e}(x) = x^i\mathbf{B}(x)$ where

$$\mathbf{B}(x) = x^{L-1} + b_{L-2}x^{L-2} + \cdots + b_1x + 1, \quad \{b_{L-2}, \cdots, b_1\} \in GF(2).$$

That is, the binary burst error pattern with a length of L bits is $(1, b_{L-2}, \ldots, b_1, 1)$, as expressed in polynomial form as $\mathbf{B}(x)$ above. In this case, if the degree of $\mathbf{g}(x)$ is larger than $L-1$, then $\mathbf{B}(x)$ is not divided by $\mathbf{g}(x)$. Therefore these L-bit burst errors can be detected. In other words, any burst errors of lengths smaller than or equal to the degree of the generator polynomial can be detected. Note that the burst error length detected by the cyclic code is determined only by the degree of its generator polynomial. If the degree of $\mathbf{B}(x)$, which is $L-1$, is equal to the degree of $\mathbf{g}(x)$, say m, and $\mathbf{B}(x) = \mathbf{g}(x)$, then the error polynomial is divided by the generator polynomial. So the error cannot be detected. If $L-1 > m$ and $\mathbf{B}(x)$ is a multiple of $\mathbf{g}(x)$, which is $\mathbf{B}(x) = \mathbf{A}(x)\mathbf{g}(x)$, then this also cannot be detected.

Theorem 2.5 *For cyclic codes defined by the generator polynomial* $\mathbf{g}(x)$ *of degree m, the burst errors with length L bits are detected as follows:*

1. *If* $L-1 < m$, *any burst errors can be detected.*
2. *If* $L-1 = m$, *the burst error detection capability is* $1 - 2^{-(m-1)}$.
3. *If* $L-1 > m$, *the burst error detection capability is* $1 - 2^{-m}$.

Proof of this theorem is left to the reader. Note that the burst error detection of the cyclic codes is very high even for burst errors with lengths greater than m. For the CCITT code whose generator polynomial is expressed by Eq. (2.12), the error detection capability of burst errors of lengths greater than 18 bits is $1 - 2^{-16} = 0.999985$.

Cyclic Parity-Check Codes and Cyclic Hamming Codes Simple parity-check codes can be expressed by the generator polynomial $\mathbf{g}(x) = x + 1$. This is because any code polynomials of a cyclic parity-check code can be obtained by multiplication of $\mathbf{g}(x) = x + 1$. For example, a cyclic code with $n = 4$ is expressed as the set of codewords

$$0000,\ 1100,\ 0110,\ 0011,\ 1001,\ 0101,\ 1010,\ 1111.$$

This set is expressed by the following code polynomials:

$$0, x^3 + x^2, x^2 + x, x + 1, x^3 + 1, x^2 + 1, x^3 + x, x^3 + x^2 + x + 1.$$

Note that all these polynomials are a factor of $x+1$. That is,

$$0 = 0 \cdot (x+1), x^3 + x^2 = x^2(x+1), x^2 + x = x(x+1), x+1 = 1 \cdot (x+1),$$
$$x^3 + 1 = (x^2 + x + 1)(x+1), x^2 + 1 = (x+1)(x+1), x^3 + x = (x^2 + x)(x+1),$$
$$x^3 + x^2 + x + 1 = (x^2 + 1)(x+1).$$

The following matrix shows how the parity-check matrix $\mathbf{H}$ is, in general, constructed by the generator polynomial $\mathbf{g}(x)$. Let β be a root of $\mathbf{g}(x)$. That is, $\mathbf{g}(\beta) = 0$. Then the $\mathbf{H}$ matrix of the code defined by $\mathbf{g}(x)$ can be expressed as

$$\mathbf{H} = \begin{bmatrix} | & & | & | & | \\ \beta^{n-1} & \cdots & \beta^2 & \beta^1 & \beta^0 \\ | & & | & | & | \end{bmatrix}, \tag{2.13}$$

where $\begin{vmatrix} \\ \beta^i \\ | \end{vmatrix}$ is a coefficient vector of x^i mod $\mathbf{g}(x)$ for $i = 0, 1, \ldots, n-1$, and n is the code length determined by exponent or period of $\mathbf{g}(x)$.

In the simple parity-check code, $\mathbf{g}(x)$ has the root $\beta = 1$. Hence the parity-check matrix organized by successive $k+1$ 1's is

$$\mathbf{H} = [\underbrace{1 \quad 1 \quad 1 \quad \cdots \quad 1 \quad 1}_{k+1}],$$

as was shown in Eq. (2.6) in Subsection 2.3.1.

The cyclic Hamming SEC code is generated by an irreducible polynomial $\mathbf{g}(x)$. The $\mathbf{H}$ matrix can be expressed as shown in Eq. (2.13). That is, the successive powers of β are all distinct from 0 through $n-1$. Then the columns of $\mathbf{H}$ are pairwise linearly independent, and therefore the code has a distance of at least three. If $\mathbf{g}(x)$ is a binary primitive polynomial with degree r, then the code length $n = 2^r - 1$.

The distance-4 Hamming SEC-DED code is described by $\mathbf{g}(x) = (x+1) \cdot \mathbf{p}(x)$, where $\mathbf{p}(x)$ is a binary primitive polynomial with degree r and has a root of α, meaning $\mathbf{p}(\alpha) = 0$:

$$\mathbf{H} = \begin{bmatrix} 1 & \cdots & 1 & 1 & 1 \\ | & & | & | & | \\ \alpha^{n-1} & \cdots & \alpha^2 & \alpha^1 & \alpha^0 \\ | & & | & | & | \end{bmatrix}, \tag{2.14}$$

where $\begin{vmatrix} \\ \alpha^i \\ | \end{vmatrix}$ is a coefficient vector of x^i mod $\mathbf{p}(x)$ and $n = 2^r - 1$. In this $\mathbf{H}$ matrix one row vector of successive n 1's indicates a simple parity check generated by the polynomial $x+1$ of $\mathbf{g}(x)$. The total number of check bits of this code is $r+1$. The matrix is related to the one shown in Eq. (2.8) where the all-1 row is located at the bottom of the matrix and one weight-1 column vector is added to the $\mathbf{H}$ matrix as shown in Eq. (2.14).

Example 2.7
A $(7, 3)$ SEC-DED code is generated by $\mathbf{g}(x) = (x + 1) \cdot \mathbf{p}(x)$, where $\mathbf{p}(x) = x^3 + x + 1$. The $\mathbf{H}$ matrix of the code is written as

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

After applying elementary row operations, as presented in Subsection 2.2.3, to this $\mathbf{H}$ matrix, we have a systematic form of the parity-check matrix. However, the systematic form of $\mathbf{H}$ can be obtained directly by expanding the generator polynomial as $\mathbf{g}(x) = (x + 1)(x^3 + x + 1) = x^4 + x^3 + x^2 + 1$. Let β be a root of $\mathbf{g}(x)$; that is, $\mathbf{g}(\beta) = \beta^4 + \beta^3 + \beta^2 + 1 = 0$. Then we have

$$\mathbf{H} = \begin{bmatrix} | & | & | & | & | & | & | \\ \beta^6 & \beta^5 & \beta^4 & \beta^3 & \beta^2 & \beta^1 & \beta^0 \\ | & | & | & | & | & | & | \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Note that all these columns are of odd weight.

2.3.5 Binary BCH Codes

BCH codes are a class of multiple random error correcting codes [BOSE60, HOCQ59]. In this subsection we cover *binary BCH codes* that are used to correct small numbers of random errors. For a detailed description of BCH codes, the reader is referred to the texts on coding theory. Among the nonbinary BCH codes, the next subsection covers the most important class of Reed-Solomon (RS) codes.

Definition 2.27 BCH codes are cyclic codes generated by $\mathbf{g}(x)$ involving multiple factors. For a t-error correcting *binary BCH code*, the generator polynomial is given by

$$\mathbf{g}(x) = \mathbf{m}_1(x)\mathbf{m}_3(x) \cdots \mathbf{m}_{2t-1}(x),$$

where $\mathbf{m}_i(x)$ is the minimal polynomial of $\alpha^i, i = 1, 3, 5, \ldots, 2t - 1$, and α is an element of $GF(2^m)$ of order n. In this case, if $\mathbf{m}_1(x)$ is a primitive polynomial of degree m over $GF(2)$, the code is called a *primitive binary BCH code*. Then α is a primitive element and its order $n = 2^m - 1$. □

In the primitive binary BCH codes, the roots of the factors of $\mathbf{g}(x)$ are as follows:

$$\begin{aligned} \mathbf{m}_1(x): &\quad \alpha, \alpha^2, \alpha^4, \cdots, \\ \mathbf{m}_3(x): &\quad \alpha^3, \alpha^6, \cdots, \\ \mathbf{m}_5(x): &\quad \alpha^5, \alpha^{10}, \cdots, \\ &\quad \cdots \\ \mathbf{m}_{2t-1}(x): &\quad \alpha^{2t-1}, \alpha^{4t-2}, \cdots. \end{aligned}$$

The $2t$ successive powers of α (i.e., $\alpha, \alpha^2, \alpha^3, \cdots, \alpha^{2t-1}, \alpha^{2t}$) are roots of $\mathbf{g}(x)$. A polynomial $\mathbf{v}(x)$ is a code polynomial if and only if $\mathbf{v}(\alpha^j) = 0$ for $j = 1, 2, \cdots, 2t$. That is, this means $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$, where $\mathbf{H}$ is a parity-check matrix of the BCH code that corrects the random t errors as follows:

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & \cdots & (\alpha^2)^{n-1} \\ 1 & \alpha^3 & (\alpha^3)^2 & \cdots & (\alpha^3)^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & \cdots & (\alpha^{2t})^{n-1} \end{bmatrix}. \tag{2.15}$$

In Eq. (2.15) the transmitted codeword is expressed as $\mathbf{V} = (v_0, v_1, v_2, \ldots, v_{n-1})$. In order to prove that the code has a distance at least $2t + 1$, we need to show that every $2t$ column vectors of $\mathbf{H}$ are linearly independent. By choosing any $2t$ columns from the matrix, we have a $2t \times 2t$ square matrix. After normalizing every column by the first row element, we have the Vandermonde matrix (shown in Subsection 2.2.3) with a nonzero determinant. This means that every $2t$ columns of $\mathbf{H}$ are linearly independent.

If α is a root of $\mathbf{g}(x)$ over $GF(2)$, then its *conjugate* α^{2^i} is also a root, which is shown in Subsection 2.1.4. For this reason even rows can be omitted from matrix (2.15). As a result the $\mathbf{H}$ matrix can be reduced to the following:

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ 1 & \alpha^3 & (\alpha^3)^2 & \cdots & (\alpha^3)^{n-1} \\ 1 & \alpha^5 & (\alpha^5)^2 & \cdots & (\alpha^5)^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha^{2t-1} & (\alpha^{2t-1})^2 & \cdots & (\alpha^{2t-1})^{n-1} \end{bmatrix}. \tag{2.16}$$

Note here that elements of $\mathbf{H}$ are in $GF(2^m)$, so each element can be represented by an m-tuple over $GF(2)$. That is, the total number of check bits is tm.

The binary BCH code has the following code parameters:

Maximum code length in bits: $n = 2^m - 1$,
Check-bit length: $n - k \leq tm$,
Minimum distance: $d_{\min} \geq 2t - 1$.

For example, using Table 2.3, we can design the triple-bit error correcting BCH code by choosing three minimal polynomials corresponding to α, α^3, and α^5 as their roots over $GF(2^4)$. That is, the generator polynomial is expressed as

$$\mathbf{g}(x) = (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1).$$

The code has 10 check bits, and these are at maximum 15 bits in the code length.

The primitive binary BCH code is defined by the generator polynomial $\mathbf{g}(x) = \mathbf{m}_1(x)$, where $\mathbf{m}_1(x)$ is a binary primitive polynomial with degree m, and it expresses a single-error correcting BCH code with code length $n = 2^m - 1$. This is same as the Hamming SEC code described in the previous subsection on cyclic codes.

TABLE 2.4 BCH Codes Generated by Primitive Elements of Order Less Than 2^9

n	k	t	n	k	t	n	k	t
7	4	1	255	199	7	511	358	18
15	11	1	255	191	8	511	349	19
15	7	2	255	187	9	511	340	20
15	5	3	255	179	10	511	331	21
31	26	1	255	171	11	511	322	22
31	21	2	255	163	12	511	313	23
31	16	3	255	155	13	511	304	25
31	11	5	255	147	14	511	295	26
31	6	7	255	139	15	511	286	27
63	57	1	255	131	18	511	277	28
63	51	2	255	123	19	511	268	29
63	45	3	255	115	21	511	259	30
63	39	4	255	107	22	511	250	31
63	36	5	255	99	23	511	241	36
63	30	6	255	91	25	511	238	37
63	24	7	255	87	26	511	229	38
63	18	10	255	79	27	511	220	39
63	16	11	255	71	29	511	211	41
63	10	13	255	63	30	511	202	42
63	7	15	255	55	31	511	193	43
127	120	1	255	47	42	511	184	45
127	113	2	255	45	43	511	175	46
127	106	3	255	37	45	511	166	47
127	99	4	255	29	47	511	157	51
127	92	5	255	21	55	511	148	53
127	85	6	255	13	59	511	139	54
127	78	7	255	9	63	511	130	55
127	71	9	511	502	1	511	121	58
127	64	10	511	493	2	511	112	59
127	57	11	511	484	3	511	103	61
127	50	13	511	475	4	511	94	62
127	43	14	511	466	5	511	85	63
127	36	15	511	457	6	511	76	85
127	29	21	511	448	7	511	67	87
127	22	23	511	439	8	511	58	91
127	15	27	511	430	9	511	49	93
127	8	31	511	421	10	511	40	95
255	247	1	511	412	11	511	31	109
255	239	2	511	403	12	511	28	111
255	231	3	511	394	13	511	19	119
255	223	4	511	385	14	511	10	121
255	215	5	511	376	15			
255	207	6	511	367	16			

Source: *[LIN04]. © 2004 pp 195–196, Adapted by permission of Pearson Education, Inc., Upper Saddle River, NJ.*

Table 2.4 shows the parameters for binary BCH codes of length $2^m - 1$ with $m \leq 9$. The generator polynomials of all primitive binary BCH codes of length $2^m - 1$ with $m \leq 10$ are given in Appendix C of [LIN04]. The more typical primitive polynomials with degrees less than or equal to 32 are presented in Table 2.1. For a comprehensive table of primitive and irreducible polynomials of degrees up to 34 over $GF(2)$, see appendix C of [PETE72]. The tables provided here are, however, very useful for constructing BCH codes and other practical codes for any given code parameters.

Decoding BCH Codes The decoding algorithm of the BCH codes goes as follows:

Let a codeword be expressed by the polynomial $\mathbf{v}(x) = v_{n-1}x^{n-1} + \cdots + v_2x^2 + v_1x + v_0$. Also let a received word be expressed by $\mathbf{r}(x) = r_{n-1}x^{n-1} + \cdots + r_2x^2 + r_1x + r_0$, which may include an error $\mathbf{e}(x)$ where $\mathbf{e}(x) = e_{n-1}x^{n-1} + \cdots + e_2x^2 + e_1x + e_0$. Then

$$\mathbf{r}(x) = \mathbf{v}(x) + \mathbf{e}(x). \tag{2.17}$$

Vectors $\mathbf{v}, \mathbf{r}$, and $\mathbf{e}$ are expressed as

$$\begin{aligned}\mathbf{v} &= (v_0\ v_1\ v_2\ \ldots\ v_{n-1}),\\ \mathbf{r} &= (r_0\ r_1\ r_2\ \ldots\ r_{n-1}),\\ \mathbf{e} &= (e_0\ e_1\ e_2\ \ldots\ e_{n-1}).\end{aligned}$$

The decoding is performed next in three steps. For decoding nonbinary BCH codes such as RS codes, an additional step (a fourth step) is needed for determining the nonbinary error values.

Step 1. ***Syndrome Generation*** For binary t-error correcting primitive BCH code, the syndrome is a $2t$-tuple,

$$\mathbf{S} = (S_1\ S_2\ \ldots\ S_{2t}) = \mathbf{r} \cdot \mathbf{H}^T,$$

where $\mathbf{H}$ is given by Eq. (2.15). So, the i-th component of the syndrome is

$$\begin{aligned}S_i &= \mathbf{r}(\alpha^i)\\ &= r_{n-1}\alpha^{(n-1)i} + \cdots + r_2\alpha^{2i} + r_1\alpha^i + r_0\end{aligned}$$

for $1 \le i \le 2t$. Note that syndrome components are elements in the field $GF(2^m)$. Divide $\mathbf{r}(x)$ by the minimal polynomial $\mathbf{m}_i(x)$ of α^i to get

$$\mathbf{r}(x) = \mathbf{a}_i(x)\mathbf{m}_i(x) + \mathbf{b}_i(x),$$

where $\mathbf{b}_i(x)$ is the remainder with degree less than that of $\mathbf{m}_i(x)$. Since $\mathbf{m}_i(\alpha^i) = 0$, write

$$S_i = \mathbf{r}(\alpha^i) = \mathbf{b}_i(\alpha^i).$$

From this the syndrome components S_i can be obtained by $\mathbf{b}_i(\alpha^i)$.

Step 2. ***Determination of Error Location Polynomial*** The syndrome is also determined by the error pattern $\mathbf{e}(x)$ from Eq. (2.17). Note that in Eq. (2.17) the relation between the syndrome components and the error pattern is

$$S_i = \mathbf{e}(\alpha^i) \qquad \text{for} \quad i = 1, 2, \ldots, 2t.$$

Assume that the error pattern $\mathbf{e}(x)$ has δ errors at location $x^{j_\delta}, \ldots, x^{j_2}, x^{j_1}$, that is,

$$\mathbf{e}(x) = x^{j_\delta} + \cdots + x^{j_2} + x^{j_1},$$

where $n > j_\delta > \cdots > j_2 > j_1 \geq 0$. Therefore write the following set of equations:

$$\begin{aligned} S_1 &= \alpha^{j_\delta} + \cdots + \alpha^{j_2} + \alpha^{j_1}, \\ S_2 &= (\alpha^{j_\delta})^2 + \cdots + (\alpha^{j_2})^2 + (\alpha^{j_1})^2, \\ &\cdots \\ &\cdots \\ S_{2t} &= (\alpha^{j_\delta})^{2t} + \cdots + (\alpha^{j_2})^{2t} + (\alpha^{j_1})^{2t}, \end{aligned}$$

where $\alpha^{j_\delta}, \ldots, \alpha^{j_2}, \alpha^{j_1}$, are unknown. Find these in the expression above and obtain the error locations $j_\delta, \ldots, j_2, j_1$ in $\mathbf{e}(x)$.

Now we need an effective procedure to use in determining α^{j_i} for $i = \delta, \ldots, 2, 1$, from the syndrome components S_i's. Let

$$\beta_i = \alpha^{j_i} \qquad \text{for} \quad 1 \leq i \leq \delta. \tag{2.18}$$

These β_i's are called *error location numbers*. The equations above are rewritten in the following form:

$$\begin{aligned} S_1 &= \beta_\delta + \cdots + \beta_2 + \beta_1, \\ S_2 &= \beta_\delta{}^2 + \cdots + \beta_2{}^2 + \beta_1{}^2, \\ &\cdots \\ &\cdots \\ S_{2t} &= \beta_\delta{}^{2t} + \cdots + \beta_2{}^{2t} + \beta_1{}^{2t}. \end{aligned}$$

These $2t$ equations are symmetric functions in $\beta_1, \beta_2, \ldots, \beta_\delta$. So we can define the polynomial as

$$\begin{aligned} \sigma(x) &= (\beta_\delta x + 1)(\beta_{\delta-1} x + 1) \cdots (\beta_2 x + 1)(\beta_1 x + 1) \\ &= \sigma_\delta x^\delta + \sigma_{\delta-1} x^{\delta-1} + \cdots + \sigma_2 x^2 + \sigma_1 x + \sigma_0. \end{aligned} \tag{2.19}$$

The roots of $\sigma(x)$ are $\beta_\delta{}^{-1}, \ldots, \beta_2{}^{-1}, \beta_1{}^{-1}$, which are the inverse of the error location numbers, so the function $\sigma(x)$ is called an *error location polynomial*. The coefficients of $\sigma(x)$ and the error location numbers are related by the equations

$$\begin{aligned} \sigma_0 &= 1, \\ \sigma_1 &= \beta_1 + \beta_2 + \cdots + \beta_\delta, \\ \sigma_2 &= \beta_1\beta_2 + \beta_2\beta_3 + \cdots + \beta_{\delta-1}\beta_\delta, \\ &\cdots \\ &\cdots \\ \sigma_\delta &= \beta_1\beta_2 \cdots \beta_\delta. \end{aligned}$$

We need to express the coefficients σ_i as functions of the syndrome components. This is accomplished in $2t + 1$ steps by using the following *Berlekamp-Massay algorithm*

TABLE 2.5 Evaluation of Error Locator Polynomial

μ	$\sigma^{(\mu)}(x)$	d_μ	δ_μ	$\mu - \delta_\mu$	ρ
-1	1	1	0	-1	—
0	1	S_1	0	0	—
1					
2					
⋮	⋮	⋮	⋮	⋮	⋮
$2t$	$\sigma(x)$				

[BERL65, MASS69]. The first two steps for the iteration index $\mu = -1, 0$, are given as the first two rows of Table 2.5. The rules behind the iterative steps follow as we proceed to fill out the table:

$$\begin{aligned}\sigma^{(\mu+1)}(x) &= \sigma^{(\mu)}(x) \qquad \text{for} \quad d_\mu = 0 \\ &= \sigma^{(\mu)}(x) + d_\mu d_\rho^{-1} x^{\mu-\rho}\sigma^{(\rho)}(x) \qquad \text{for} \quad d_\mu \neq 0.\end{aligned}$$

Here, d_μ is the μ-th *discrepancy* and is given by

$$d_\mu = S_{\mu+1} + \sigma_1^{(\mu)} S_\mu + \sigma_2^{(\mu)} S_{\mu-1} + \cdots + \sigma_{\delta_\mu}^{(\mu)} S_{\mu+1-\delta_\mu}.$$

The *step number* prior to μ is ρ such that $d_\rho \neq 0$ and $\rho - \delta_\rho$ has the largest value. Also δ_ρ is the degree of $\sigma^{(\rho)}(x)$ and $\sigma_i^{(\mu)}$ is the coefficient of the x^i term of $\sigma^{(\mu)}(x)$. This procedure terminates at step $2t$ with $\sigma(x) = \sigma^{(2t)}(x)$. If the errors are δ in $\mathbf{e}(x)$, then $\sigma(x)$ has degree δ.

For binary BCH codes, it is not necessary to fill out all rows of Table 2.5 in finding $\sigma(x)$. It can be obtained by filling out $t+2$ rows, meaning rows with $\mu = -1, 0, 1, 2, \ldots, t$. The computation required is almost one-half of that required in this above general algorithm.

Step 3. ***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 $1, \alpha, \alpha^2, \ldots, \alpha^{n-1}$, where $n = 2^m - 1$, into $\sigma(x)$. Since $\alpha^n = 1$, then $\alpha^{-j} = \alpha^{n-j}$. Therefore, if α^j is a root of $\sigma(x)$, α^{n-j} is an error location number and the received digit r_{n-j} is an erroneous digit. Therefore, if the error location numbers are α^{n-j_1} to α^{n-j_t} where $j_1 > j_2 \cdots > j_t$, the error polynomial is determined as

$$\mathbf{e}(x) = x^{n-j_t} + \cdots + x^{n-j_2} + x^{n-j_1},$$

which gives the assumed error pattern. The decoding is completed by adding $\mathbf{e}(x)$ to the received vector $\mathbf{r}(x)$.

Another procedure that can be used to carry out the substitution and error correction is known as *Chien's search* [CHIE64]. The received vector is decoded bit by bit. The high-order bits are first decoded. In order to decode r_{n-1}, the decoder tests whether or not α^{n-1} is an error location number. This is equivalent to testing whether or not an inverse alpha is a root of $\sigma(x)$. If α is a root, then α^{n-1} is an error location number and r_{n-1} is an erroneous

digit; otherwise, r_{n-1} is the correct digit. This process continues until the lowest digit r_0 is determined.

Example 2.8

A binary $(15, 5)$ triple-error correcting BCH code is given by the generator polynomial $\mathbf{g}(x) = \mathbf{m}_1(x)\mathbf{m}_3(x)\mathbf{m}_5(x) = (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)$. In this case the minimal polynomials for α, α^2, and α^4 are identical and are expressed as $\mathbf{m}_1(x) = x^4 + x + 1$. Likewise for α^3 and α^6 the minimal polynomial is $\mathbf{m}_3(x) = x^4 + x^3 + x^2 + x + 1$, and the minimal polynomial for α^5 is $\mathbf{m}_5(x) = x^2 + x + 1$. Here α is a primitive element of $GF(2^4)$ such that $\alpha^4 + \alpha + 1 = 0$.

Assume that the code vector of all zeros

$$\begin{aligned}\mathbf{v} &= (v_0\, v_1\, v_2\, v_3\, v_4\, v_5\, v_6\, v_7\, v_8\, v_9\, v_{10}\, v_{11}\, v_{12}\, v_{13}\, v_{14}) \\ &= (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)\end{aligned}$$

is transmitted and the vector

$$\begin{aligned}\mathbf{r} &= (r_0\, r_1\, r_2\, r_3\, r_4\, r_5\, r_6\, r_7\, r_8\, r_9\, r_{10}\, r_{11}\, r_{12}\, r_{13}\, r_{14}) \\ &= (0\ 1\ 0\ 0\ 0\ 0\ 0\, 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0)\end{aligned}$$

is received. Then $\mathbf{r}(x) = x^{11} + x^7 + x$. Dividing $\mathbf{r}(x)$ by $\mathbf{m}_1(x), \mathbf{m}_3(x)$ and $\mathbf{m}_5(x)$, respectively, we obtain the remainders

$$\begin{aligned}\mathbf{b}_1(x) &= x^2 + x + 1, \\ \mathbf{b}_3(x) &= x^2, \\ \mathbf{b}_5(x) &= x + 1.\end{aligned}$$

Substituting α, α^2, and α^4 into $\mathbf{b}_1(x)$, we obtain the syndrome components of $S_1 = \alpha^2 + \alpha + 1 = \alpha^{10}, S_2 = \alpha^4 + \alpha^2 + 1 = \alpha^5$, and $S_4 = \alpha^8 + \alpha^4 + 1 = \alpha^{10}$, respectively. Substituting α^3 and α^6 into $\mathbf{b}_3(x)$, we have $S_3 = \alpha^6$, and $S_6 = \alpha^{12}$, respectively. Also substituting α^5 into $\mathbf{b}_5(x)$, we have $S_5 = \alpha^5 + 1 = \alpha^{10}$.

By the iterative procedure described previously, we obtain the values shown in Table 2.6. Thus the error location polynomial is

$$\sigma(x) = \sigma^{(6)}(x) = \alpha^4 x^3 + \alpha x^2 + \alpha^{10} x + 1.$$

TABLE 2.6 Evaluation of Error Locator Polynomial

μ	$\sigma^{(\mu)}(x)$	d_μ	δ_μ	$\mu - \delta_\mu$	ρ
-1	1	1	0	-1	—
0	1	α^{10}	0	0	—
1	$\alpha^{10}x + 1$	0	1	0	-1
2	$\alpha^{10}x + 1$	α^{13}	1	1	—
3	$\alpha^3 x^2 + \alpha^{10}x + 1$	0	2	1	0
4	$\alpha^3 x^2 + \alpha^{10}x + 1$	α^7	2	2	—
5	$\alpha^4 x^3 + \alpha x^2 + \alpha^{10}x + 1$	0	3	2	2
6	$\alpha^4 x^3 + \alpha x^2 + \alpha^{10}x + 1$	—	—	—	—

We can easily check that α^4, α^8, and α^{14} are the roots of $\sigma(x)$. Their inverses are α^{11}, α^7, and α, respectively, which give the error location numbers. Therefore the error polynomial is

$$\mathbf{e}(x) = x^{11} + x^7 + x,$$

and it gives the error vector

$$\begin{aligned}\mathbf{e} &= (e_0\, e_1\, e_2\, e_3\, e_4\, e_5\, e_6\, e_7\, e_8\, e_9\, e_{10}\, e_{11}\, e_{12}\, e_{13}\, e_{14}) \\ &= (0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ ).\end{aligned}$$

After adding $\mathbf{e}(x)$ to the received polynomial $\mathbf{r}(x)$, or adding the error vector to the received vector, we obtain the code polynomial $\mathbf{v}(x) = 0$, which is the all-zero code vector.

2.3.6 Reed-Solomon Codes as Nonbinary BCH Codes

The *Reed-Solomon codes* [REED60], or *RS codes*, are a subclass of the nonbinary BCH codes. Codes over $GF(2^m)$ are especially important for binary digital systems, and these codes have found in many applications, including high-speed semiconductor memories, magnetic / optical disk memories, tape memories, and communication systems, as will be described in later chapters. A nonbinary element in $GF(2^m)$, called a *symbol* or a *byte*, is sometimes expressed in binary form as a cluster of m bits[a] (as we saw in Subsection 2.1.3).

Definition 2.28 Let α be a primitive element in $GF(2^m)$. The parity-check matrix of the *RS codes* with minimum Hamming distance d over $GF(2^m)$ is written as

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \cdots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{d-2} & \alpha^{2(d-2)} & \alpha^{3(d-2)} & \cdots & \alpha^{(d-2)(n-1)} \end{bmatrix}, \tag{2.20}$$

where $n = 2^m - 1$. □

The matrix shown in Eq. (2.20) can be obtained from the one shown in Eq. (2.15) by normalizing every column by its top element and $2t = d - 1$. The generator polynomial of this code, with distance d and code length $2^m - 1$, is expressed as

$$\begin{aligned}\mathbf{g}(x) &= (x + \alpha)(x + \alpha^2) \cdots (x + \alpha^{d-1}) \\ &= x^{d-1} + g_{d-2}x^{d-2} + \cdots + g_2x^2 + g_1x + g_0,\end{aligned}$$

where $\alpha, \alpha^2, \cdots, \alpha^{d-1}$, are all roots of $\mathbf{g}(x)$ and the coefficients $g_{d-2}, \cdots, g_2, g_1, g_0$, are from $GF(2^m)$.

[a] In later chapters a parameter of symbol size or byte size m will be designated as b.

Note that this code satisfies the *maximum distance separable* (*MDS*) characteristic because the check length r depends only on d, meaning $r = d - 1$. Further its maximum code length is determined only by m.

Lengthened Codes

Definition 2.29 The code can be lengthened by adding two columns to the **H** defined in Eq. (2.20) without reducing its minimum distance. The *lengthened RS code*, or *extended RS code*, has code length $n + 2 = 2^m + 1$ and has the same check length as the original one. That is, the **H** matrix of the lengthened code is expressed as

$$\mathbf{H}_L = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 & 0 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} & 0 & 0 \\ 1 & \alpha^2 & \alpha^4 & \cdots & \alpha^{2(n-1)} & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 1 & \alpha^{d-3} & \alpha^{2(d-3)} & \cdots & \alpha^{(d-3)(n-1)} & 0 & 0 \\ 1 & \alpha^{d-2} & \alpha^{2(d-2)} & \cdots & \alpha^{(d-2)(n-1)} & 0 & 1 \end{bmatrix}.$$

□

In particular, for $d = 4$, three columns can be added. After the 3×3 identity matrix is added [WOLF69], the code length becomes $n + 3 = 2^m + 2$. This case will be discussed in Chapter 5.

Decoding RS Codes

Decoding RS codes requires an additional step, a fourth step added to the previous decoding BCH code in order to determine the nonbinary error values.

Determination of Error Values From the first step of syndrome determination provided in Subsection 2.3.5, we have the following relation:

$$S_j = \mathbf{r}(\alpha_j) = \mathbf{e}(\alpha_j) \qquad \text{for} \quad j = 1, 2, \cdots, 2t.$$

We also obtain the syndrome polynomial as

$$\mathbf{S}(x) = S_{2t}x^{2t} + S_{2t-1}x^{2t-1} + \cdots + S_1x + 1.$$

Another polynomial $\mathbf{z}(x)$ with degree δ, called an *evaluator polynomial*, is defined as

$$\mathbf{z}(x) = (S_\delta + \sigma_1 S_{\delta-1} + \sigma_2 S_{\delta-2} + \cdots + \sigma_\delta)x^\delta + \cdots + (S_2 + \sigma_1 S_1 + \sigma_2)x^2 + (S_1 + \sigma_1)x + 1 \tag{2.21}$$

This polynomial consists of all components of degree $\leq \delta$ of the product $\sigma(x)\mathbf{S}(x)$. We compute the error value at location $\beta_i = \alpha^{j_i}$ as

$$e_{j_i} = \frac{\mathbf{z}(\beta_i{}^{-1})}{\prod_{v \neq i}(1 + \beta_v \beta_i{}^{-1})} \qquad \text{for} \quad i = 1, 2, \ldots, \delta. \tag{2.22}$$

Example 2.9
Consider a triple-error correcting RS code with symbols from $GF(2^4)$. The generator polynomial of this code is

$$\begin{aligned}\mathbf{g}(x) &= (x+\alpha)(x+\alpha^2)(x+\alpha^3)(x+\alpha^4)(x+\alpha^5)(x+\alpha^6)\\ &= x^6 + \alpha^{10}x^5 + \alpha^{14}x^4 + \alpha^4 x^3 + \alpha^6 x^2 + \alpha^9 x + \alpha^6,\end{aligned}$$

where α is a primitive element in $GF(2^4)$. Let the all-zero vector be the transmitted code vector, and let

$$\begin{aligned}\mathbf{r} &= (r_0\, r_1\, r_2\, r_3\, r_4\, r_5\, r_6\, r_7\ \ r_8\ r_9\, r_{10}\, r_{11}\, r_{12}\, r_{13}\, r_{14})\\ &= (0\ \ 0\ \alpha^5\ 0\ 0\ \ 0\ 0\ 0\ \ \alpha^{13}\, 0\ \ 0\ \ \alpha^2\ \ 0\ \ 0\ \ 0\ )\end{aligned}$$

be the received vector. Thus, $\mathbf{r}(x) = \alpha^2 x^{11} + \alpha^{13}x^8 + \alpha^5 x^2$.

Step 1. The syndrome elements are computed as

$$\begin{aligned}S_1 &= \mathbf{r}(\alpha) = \alpha^{13} + \alpha^6 + \alpha^7 = \alpha^9,\\ S_2 &= \mathbf{r}(\alpha^2) = \alpha^9 + \alpha^{14} + \alpha^9 = \alpha^{14},\\ S_3 &= \mathbf{r}(\alpha^3) = \alpha^5 + \alpha^7 + \alpha^{11} = \alpha^4,\\ S_4 &= \mathbf{r}(\alpha^4) = \alpha + 1 + \alpha^{13} = \alpha^{11},\\ S_5 &= \mathbf{r}(\alpha^5) = \alpha^{12} + \alpha^8 + 1 = \alpha^7,\\ S_6 &= \mathbf{r}(\alpha^6) = \alpha^8 + \alpha + \alpha^2 = \alpha^4.\end{aligned}$$

Step 2. To find the error location polynomial $\sigma(x)$, fill out the evaluation table, as shown in Table 2.7

Step 3. Substituting $1, \alpha, \alpha^2, \ldots, \alpha^{14}$ into $\sigma(x) = \alpha^6 x^3 + \alpha^{14}x^2 + \alpha^{12}x + 1$, find that α^4, α^7, and α^{13} are roots of $\sigma(x)$. The reciprocals of these roots are α^{11}, α^8, and α^2, respectively, which are the error location numbers of the error polynomial $\mathbf{e}(x)$. These errors occur at positions x^{11}, x^8, and x^2.

Step 4. From Eq. (2.21), find $\mathbf{z}(x) = \mathbf{S}(x)\sigma(x)$ with degree ≤ 3, $\mathbf{S}(x) = \alpha^4 x^6 + \alpha^7 x^5 + \alpha^{11}x^4 + \alpha^4 x^3 + \alpha^{14}x^2 + \alpha^9 x + 1$ and $\sigma(x) = \alpha^6 x^3 + \alpha^{14}x^2 + \alpha^{12}x + 1$, such that

$$\mathbf{z}(x) = \alpha^2 x^3 + \alpha^6 x^2 + \alpha^8 x + 1.$$

TABLE 2.7 Evaluation of Error Location Polynomial

μ	$\sigma^{(\mu)}(x)$	d_μ	δ_μ	$\mu - \delta_\mu$	ρ
−1	1	1	0	−1	—
0	1	α^9	0	0	—
1	$\alpha^9 x + 1$	1	1	0	−1
2	$\alpha^5 x + 1$	0	1	1	0
3	$\alpha^5 x + 1$	α^2	1	2	—
4	$\alpha^{11}x^3 + \alpha^2 x^2 + \alpha^5 x + 1$	α	3	1	1
5	$\alpha^{11}x^3 + \alpha^{10}x^2 + \alpha^{12}x + 1$	α^{13}	3	2	3
6	$\alpha^6 x^3 + \alpha^{14}x^2 + \alpha^{12}x + 1$	—	—	—	3

Using Eq. (2.22), obtain the error values at locations x^{11}, x^8, and x^2:

$$e_{11} = \frac{\alpha^2(\alpha^{-11})^3 + \alpha^6(\alpha^{-11})^2 + \alpha^8\alpha^{-11} + 1}{(1+\alpha^8\alpha^{-11})(1+\alpha^2\alpha^{-11})} = \frac{\alpha^{11}}{\alpha^{11}\alpha^{13}} = \alpha^2,$$
$$e_8 = \frac{\alpha^2(\alpha^{-8})^3 + \alpha^6(\alpha^{-8})^2 + \alpha^8\alpha^{-8} + 1}{(1+\alpha^{11}\alpha^{-8})(1+\alpha^2\alpha^{-8})} = \frac{\alpha^4}{\alpha^{14}\alpha^7} = \alpha^{13},$$
$$e_2 = \frac{\alpha^2(\alpha^{-2})^3 + \alpha^6(\alpha^{-2})^2 + \alpha^8\alpha^{-2} + 1}{(1+\alpha^{11}\alpha^{-2})(1+\alpha^8\alpha^{-2})} = \frac{\alpha^{10}}{\alpha^7\alpha^{13}} = \alpha^5.$$

Thus the error polynomial is

$$\mathbf{e}(x) = \alpha^2 x^{11} + \alpha^{13} x^8 + \alpha^5 x^2,$$

which is exactly the difference between the received polynomial and the code polynomial. The decoding is completed by taking $\mathbf{r}(x) - \mathbf{e}(x) = 0$, that is, the all-zero codeword.

2.3.7 Burst Error Correcting Fire Codes

The communication systems and disk or tape memory can sometimes cause clusters of errors, namely *burst errors*. As we saw in Subsection 2.3.4, cyclic codes are effective for burst error detection. *Fire codes* are a large class of burst error correcting cyclic codes that have been popularly applied to disk memory (as will be discussed again in Subsection 11.2.1). Many other effective burst error correcting cyclic codes, besides the Fire codes, have been constructed both analytically and with the aid of a computer search.

Burst Errors 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, $\mathbf{e} = (0\,0\,1\,0\,0\,1\,0\,1\,0\,0)$ is a burst of length 6. A linear code 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*.

In order to correct single l-burst errors, any $2l$ columns in the parity-check matrix of an (n,k) code should be linearly independent. Therefore the following theorem holds.

Theorem 2.6 *The number of check digits of an l-burst error correcting code must have at least 2l, that is*,

$$n - k \geq 2l.$$

In accordance with Theorem 2.6, the burst error length is expressed as

$$l \leq \left\lfloor \frac{n-k}{2} \right\rfloor, \tag{2.23}$$

where $\lfloor x \rfloor$ refers to the largest integer smaller than or equal to x. The upper bound on the l-burst error correcting capability of an (n,k) code is called the *Reiger bound* [REIG60].

As for detecting single l-burst errors, the number of check digit of the (n, k) code must have at least l, that is, $n - k \geq l$.

Single-Burst Error Correcting Fire Codes Fire codes [FIRE59] are a class of binary cyclic codes that correct single l-burst errors.

Definition 2.30 Let $\mathbf{p}(x)$ be a binary irreducible polynomial of degree m, and also let e be the smallest integer such that $\mathbf{p}(x)$ divides $x^e + 1$. The integer e is called the *period* of $\mathbf{p}(x)$. Also let l be a positive integer such that $l \leq m$ and $c = 2l - 1$ is not divisible by e. A *single l-burst error correcting Fire code* is generated by the following polynomial:

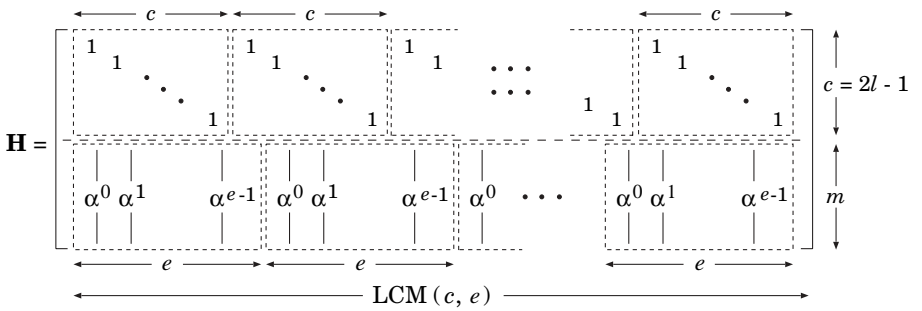
$$\mathbf{g}(x) = (x^c + 1)\mathbf{p}(x). \tag{2.24}$$

The code length n is the least common multiple of c and the period e of $\mathbf{p}(x)$, that is,

$$n = \text{LCM}(c, e). \tag{2.25}$$

The number of check digits of this code is $r = c + m = 2l - 1 + m$. Note that the polynomials $x^c + 1$ and $\mathbf{p}(x)$ are relatively prime. □

The parity-check matrix of the Fire code can be expressed as follows:



(2.26)

This **H** matrix consists of two parts. The upper part consists of a series of $c \times c$ identity matrices, and the lower part consists of a repeated series of coefficient column vectors of x^i mod $\mathbf{p}(x)$, where $0 \leq i \leq e - 1$. Each upper submatrix has the degree of c and each lower submatrix has the period of e. Therefore the code length n is determined as the first coincidence with these c and e, that is, the least common multiple of c and e. The upper $c \times c$ matrix corresponds to $x^c + 1$ and the lower corresponds to $\mathbf{p}(x)$ of $\mathbf{g}(x)$.

The decoding of the Fire code will be presented in Chapter 11 where the serial decoding is implemented by LFSRs. It is also discussed in Chapter 8 as parallel decoding implemented by combinational logic circuits.

Other Single-Burst Error Correcting Codes Besides the Fire codes, some efficient cyclic codes and shortened cyclic codes for correcting single short bursts have been explored either analytically or with the aid of computers [STON61, ELSP62, KASA63, KASA64]. These efficient codes with their generator polynomials are given in

TABLE 2.8 Efficient Burst Error Correcting Cyclic Codes

$n-k-2l$	Code (n,k)	Length of burst error correction l	Generator polynomial $\mathbf{g}(x)^a$
0	(7,3)	2	35
0	(15,9)	3	171
0	(15,7)	4	721
0	(15,5)	5	2467
0	(19,11)	4	1151
0	(21,9)	6	14515
0	(21,7)	7	47343
0	(21,5)	8	214537
0	(21,3)	9	1647235
0	(27,17)	5	2671
0	(34,22)	6	15173
0	(38,24)	7	114361
0	(50,34)	8	224531
0	(56,38)	9	1505773
0	(59,39)	10	4003351
1	(15,10)	2	65
1	(21,14)	3	171
1	(21,12)	4	11663
1	(21,10)	5	7707
1	(23,12)	5	5343
1	(27,20)	3	311
1	(31,20)	5	4673
1	(38,29)	4	1151
1	(48,37)	5	4501
1	(63,50)	6	22377
1	(63,48)	7	105437
1	(63,46)	8	730535
1	(63,44)	9	2002353
1	(67,54)	6	36365
1	(96,79)	7	114361
1	(103,88)	8	501001

Source: *[LIN04]. © 2004 pp 1114, Adapted by permission of Pearson Education, Inc., Upper Saddle River, NJ.*

[a] The generator polynomial is given as an octal representation. Each digit represents three binary digits according to the following code:

$$0 \leftrightarrow 000 \quad 2 \leftrightarrow 010 \quad 4 \leftrightarrow 100 \quad 6 \leftrightarrow 110$$
$$1 \leftrightarrow 001 \quad 3 \leftrightarrow 011 \quad 5 \leftrightarrow 101 \quad 7 \leftrightarrow 111$$

The binary digits are 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 $\mathbf{g}(x) = x^6 + x^5 + x^4 + x^3 + 1$.

Table 2.8 for $n - k - 2l = 0$ and 1. The codes with $n - k - 2l > 1$ are given in Table 20.3 of [LIN04]. These are the most efficient single-burst error correcting codes known.

A class of *phased burst error correcting cyclic codes*, called *Burton codes* [BURT71], are defined as single l-bit burst error correcting codes as follows:

$$\mathbf{g}(x) = (x^l + 1)\mathbf{p}(x),$$

where $\mathbf{p}(x)$ is an irreducible polynomial of degree l and period e. The code length is given as $\text{LCM}(l, e)$ and the check length as $2l$. The $\mathbf{H}$ matrix of this code is similar to the matrix shown in Eq. (2.26) where the upper submatrices are $l \times l$ identity matrices and the lower submatrices are $l \times e$ matrices. This type of codes is belonged to a *single-byte error correcting code*, as will be mentioned in Chapter 5.

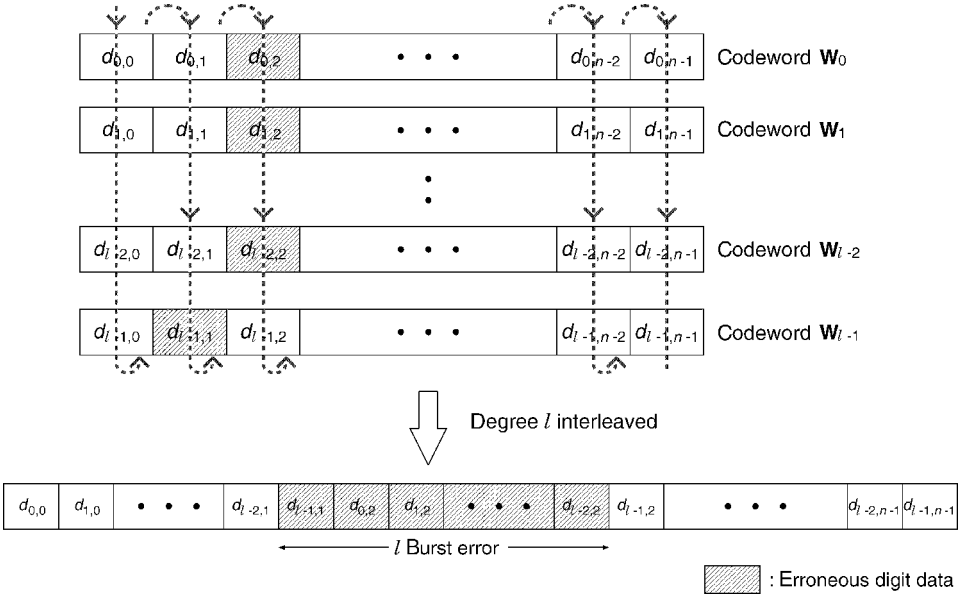


Figure 2.6 *Interleaving.*

Interleaving Given an original (n, k) code, it is possible to construct an (ln, lk) code by interleaving. This can be realized by using the l original codes, that is, by arranging the l codewords, $\mathbf{w}_0$ to $\mathbf{w}_{l-1}$, of the original code into l rows of a rectangular array, shown in Figure 2.6. Digit data in every codeword is transmitted (for communication systems) or recorded (for disk / tape memory systems) sequentially column by column from this array, as shown in the lower part of Figure 2.6. The resulting code is referred to as an *interleaved code*. The parameter l is called the *interleaving degree*.

In this code, no matter where the error starts, a burst error with length l affects no more than one digit in each row. Therefore burst errors can be corrected in the array if and only if the error in each row is a correctable error pattern of the original code. If the original code corrects single errors, the interleaved code corrects single bursts with length l or less. If the original code corrects any single burst with length p or less, the interleaved code corrects any single burst with length lp or less. It should be clear from this description that the interleaving method gives us a simple design of long burst error correcting codes. This is why the interleaved code is popularly applied to digital systems, and so is discussed in many places in this book. Note, however, that this effective coding technique requires a large number of check bits.

EXERCISES

2.1 Construct the group under modulo-6 addition.

2.2 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.

2.3 Find an isomorphism between the set of integers under addition $\boldsymbol{Z}_4 = \{0, 1, 2, 3\}$ and the multiplicative group $\boldsymbol{G} = \{1, 2, 3, 4\}$.

2.4 Find all primitive elements in $GF(7)$ and $GF(11)$.

2.5 Construct a table for $GF(2^3)$ defined by the primitive polynomial $\mathbf{p}(x) = x^3 + x + 1$. Express each element in $GF(2^3)$ by power, polynomial, vector, and matrix representations.

2.6 Show that the polynomial $x^5 + x^2 + 1$ is irreducible and primitive over $GF(2)$.

2.7 Prove that reciprocal polynomial $\mathbf{p}(x)^*$ is irreducible if and only if $\mathbf{p}(x)$ is irreducible over $GF(2)$. Also prove that $\mathbf{p}(x)^*$ is primitive if and only if $\mathbf{p}(x)$ is primitive.

2.8 Find all irreducible polynomials of degree 2 over $GF(3)$ and determine which of these are primitive.

2.9 Let the primitive polynomial with degree 5 be $\mathbf{p}(x) = x^5 + x^2 + 1$, and also α be a root of $\mathbf{p}(x)$, meaning a primitive element of $GF(2^5)$. Find the minimal polynomials of α^3, α^5, and α^7.

2.10 Let α be a root of primitive polynomial $x^2 + x + 2$ over $GF(3)$. Find the minimal polynomials of all elements in $GF(3^2)$.

2.11 Transform the matrices below to systematic forms.

$$\mathbf{H}_1 = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix} \quad \text{over} \quad GF(2),$$

$$\mathbf{H}_2 = \begin{bmatrix} a & 1 & b & 0 & 1 & b \\ b & 0 & 1 & a & b & a \\ 1 & b & a & 1 & 0 & a \end{bmatrix} \quad \text{over} \quad GF(4).$$

2.12 Prove that the companion matrix is nonsingular.

2.13 Show that rows or columns of nonsingular matrix are linearly independent. And show that any nonsingular matrix can be transformed into identity matrix by elementary row operations.

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

2.15 Given the following $\mathbf{H}$ matrix of a linear code $\boldsymbol{C}$, answer the questions below:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & a & b \\ 0 & 1 & b & a \end{bmatrix} \quad \text{over} \quad GF(4).$$

(a) Transform the above matrix to a systematic form.

(b) Find all codewords of $\boldsymbol{C}$.

(c) Encode a given message $(a\,b)$.

(d) Decode a received word $(1\,b\,b\,0)$.

2.16 Prove that the code is capable of correcting any combinations of random t errors and e erasures if the minimum distance of the code $d_{\min}$ is at least $2t + e + 1$.

2.17 Let a linear code be expressed by the parity-check matrix $\mathbf{H}$,

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ 1 & \alpha & \alpha^2 & 0 & 1 \end{bmatrix} \quad \text{over} \quad GF(4),$$

where α is a root of the binary polynomial $x^2 + x + 1$.

(a) Show that the set $\{0, 1, \alpha, \alpha^2\}$ is $GF(4)$.
(b) Show that the code is a single-error correcting code.
(c) For the given input information $(1\,0\,\alpha^2)$, find the transmitted word.
(d) Decode the received word $(1\,\alpha\,0\,\alpha^2\,1)$.
(e) For the single-error correcting code over $GF(4)$ expressed by a parity-check matrix with r rows, express the maximum code length by using r. Also mention how the answer is deduced.

2.18 Let $\boldsymbol{C}$ be a binary code with code length n and information length k whose parity-check matrix $\mathbf{H}$ is organized by distinct odd-weight-column vectors, each with length $r\,(= n - k)$. Here "weight of a vector" means the number of 1's in a vector.

(a) Construct the parity-check matrix $\mathbf{H}$ of code $\boldsymbol{C}$ for $n = 16$ and $k = 11$.
(b) Express the maximum code length of code $\boldsymbol{C}$ by using r.
(c) Prove that sum of arbitrary two column vectors in $\mathbf{H}$ is an even-weight column vector.
(d) Prove that the code $\boldsymbol{C}$ with maximum code length has the minimum Hamming distance 4.
(e) Prove that every codeword of $\boldsymbol{C}$ has even weight.

2.19 If an error pattern $\mathbf{e}(x)$ is detectable for a cyclic code, show that its cyclic shifted pattern of $\mathbf{e}(x)$ is also detectable.

2.20 Let $\mathbf{g}(x)$ be a generator polynomial with degree $n - k$ of the (n, k) cyclic code $\boldsymbol{C}$. Prove that $\mathbf{g}(x)$ has the following properties:

1. $\mathbf{g}(x)$ is a monic polynomial with constant term equal to 1, and is uniquely determined.
2. $\mathbf{g}(x)$ is a factor of $x^n + 1$, meaning $x^n + 1 = \mathbf{g}(x)\mathbf{h}(x)$, where $\mathbf{h}(x)$, called a *parity-check polynomial*, has degree k.
3. The reciprocal of $\mathbf{h}(x)$, defined as $x^k\mathbf{h}(1/x)$, is also a factor of $x^n + 1$, and generates a cyclic code, that is, a *dual code* of $\boldsymbol{C}$.

2.21 For the $(7, 4)$ cyclic code $\boldsymbol{C}$ defined by the generator polynomial $\mathbf{g}(x) = x^3 + x + 1$, show that the dual code of $\boldsymbol{C}$ has minimum Hamming distance 4.

2.22 Given a $(7, 4)$ cyclic SEC code $\boldsymbol{C}$ with a generator polynomial $\mathbf{g}(x) = x^3 + x + 1$, obtain the dual code of $\boldsymbol{C}$ which has an SEC-DED capability.

2.23 Prove Theorem 2.5.

2.24 Consider the cyclic Hamming code $\boldsymbol{C}$ with length $2^m - 1$ generated by $\mathbf{g}(x) = (x+1)\mathbf{p}(x)$, where $\mathbf{p}(x)$ is a primitive polynomial with degree m. An error pattern of the form

$$\mathbf{e}(x) = x^i + x^{i+1}$$

is called a double adjacent-error pattern, which is a burst error of length two. Prove that the code $\boldsymbol{C}$ is capable of correcting all double adjacent-error patterns as well as all single-error patterns by using the parity-check matrix of the code $\boldsymbol{C}$.

2.25 Using the $GF(2^5)$ generated by $\mathbf{p}(x) = x^5 + x^2 + 1$, find the generator polynomial of the primitive BCH codes of length 31. For the double-error correcting BCH code with code length 31 obtained by this generator polynomial, decode the received polynomial $\mathbf{r}(x) = x^8 + x^7 + 1$. (Answer: $\mathbf{e}(x) = x^{13} + x^2$, and hence $\mathbf{v}(x) = x^{13} + x^8 + x^7 + x^2 + 1$.)

2.26 Given the triple-error correcting RS code with symbols from $GF(2^4)$, answer the following questions:

(a) Find the generator polynomial.

(b) Let the all-zero vector be transmitted, and then let $\mathbf{r} = (r_0\, r_1\, r_2\, r_3\, r_4\, r_5\, r_6\, r_7\, r_8\, r_9\, r_{10}\, r_{11}\, r_{12}\, r_{13}\, r_{14}) = (0\,0\,\alpha^4\,0\,0\,0\,0\,0\,\alpha^3\,0\,0\,\alpha^7\,0\,0\,0)$ be the received vector, where α is a primitive element in $GF(2^4)$. Find the syndrome with 6 elements S_0, S_1, S_2, S_3, S_4, and S_5.

(c) Fill out the evaluation table and find the error location polynomial $\boldsymbol{\sigma}(x)$.

(d) Find the roots of $\sigma(x)$.

(e) Find the polynomial $\mathbf{z}(x)$.

(f) Find the error values by using Eq. (2.22).

2.27 Find the generator polynomial of the double-error correcting RS code with code length $2^4 - 1$ by using the symbols from $GF(2^4)$. Let α be a primitive element generated by $\mathbf{g}(x) = x^4 + x + 1$. Then decode the received polynomial $\mathbf{r}(x) = \alpha^2 x^3 + \alpha^5 x + \alpha^3$. Express the binary parity-check matrix of this code. (Answer: $\mathbf{g}(x) = x^4 + \alpha^{13}x^3 + \alpha^6 x^2 + \alpha^3 x + \alpha^{10}$, $\mathbf{e}(x) = \alpha^7 x^7 + \alpha^5 x^2$, $\mathbf{v}(x) = \alpha^7 x^7 + \alpha^2 x^3 + \alpha^5 x^2 + \alpha^5 x + \alpha^3$.)

2.28 Count the number of l-bit burst errors in a word of length L-bit. (Answer: $2^l + (L-l)\cdot 2^{l-1} - 1$.)

2.29 Prove that the following matrix $\mathbf{H}$ is the parity-check matrix of the single b-bit byte error detecting code with a code length of nb bits. Also, prove that this is the parity-check matrix of the single b-bit burst error detecting code.

$$\mathbf{H} = [\overbrace{\mathbf{I}\,\mathbf{I}\,\mathbf{I}\,\cdots\,\mathbf{I}\,\mathbf{I}}^{n}], \quad \mathbf{I}:\ b \times b \text{ identity matrix.}$$

2.30 Find the generator polynomial of the Fire code capable of correcting a single error burst of a length of 4 bits or less. Also find the code length and express this code by using the parity-check matrix.

2.31 Show that the code defined by the generator polynomial $\mathbf{g}(x) = (x^9 + 1)(x^5 + x^2 + 1)$ over $GF(2)$ can correct single burst errors with lengths of 5 bits. Find the maximum code length of this code.

REFERENCES

[BERL65] E. R. Berlekamp, ''On Decoding Binary Bose-Chaudhuri-Hocquenghem Codes,'' *IEEE Trans. Info. Theory*, IT-11 (October 1965): 577–579.

[BERL84] E. R. Berlekamp, *Algebraic Coding Theory*, rev. 1984 ed., Aegean Park Press (1984).

[BOSE60] R. C. Bose and D. K. Ray-Chaudhuri, ''On a Class of Error Correcting Binary Group Codes,'' *Info. Contr.*, 3 (March 1960): 68–79.

[BRAH84] R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison Wesley (1984).

[BURT71] H. O. Burton, ''Some Asymptotically Optimal Burst-Correcting Codes and Their Relation to Single-Error-Correcting Reed-Solomon Codes,'' *IEEE Trans. Info. Theory*, IT-17 (January 1971): 92–95.

[CHIE64] R. T. Cien, ''Cyclic Decoding Procedure for the Bose-Chaudhuri-Hocquenghem Codes,'' *IEEE Trans. Info. Theory*, IT-10 (October 1964): 357–363.

[ELSP62] B. E. Elspas and R. A. Short, ''A Note on Optimum Burst-Error-Correction Codes,'' *IRE Trans. Info. Theory*, IT-8 (January 1962): 39–42.

[FIRE59] P. Fire, ''A Class of Multiple-Error-Correcting Codes and Decoders for Non-independent Binary Errors,'' *Sylvania Report RSL-E-2*, Sylvania Electronic Defense Laboratory (March 1959).

[HAMM50] R. W. Hamming, ''Error Detecting and Error Correcting Codes,'' *Bell Syst. Tech. J.*, 29 (April 1950): 147–160.

[HOCQ59] A. Hocquenghem, ''Codes Correcteurs d'Erreurs,'' *Chifres*, 2 (1959): 147–156.

[HSIA70] M. Y. Hsiao, ''A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes,'' *IBM J. Res. Dev.*, 14 (July 1970): 395–401.

[KASA63] T. Kasami, ''Optimum Shortened Cyclic Codes for Burst-Error-Correction,'' *IEEE Trans. Info. Theory*, IT-9 (April 1963): 105–109.

[KASA64] T. Kasami and S. Matoba, ''Some Efficient Shortened Cyclic Codes for Burst-Error-Correction,'' *IEEE Trans. Info. Theory*, IT-10 (July 1964): 252–253.

[LIN04] S. Lin and D. J. Costello Jr., *Error Control Coding*, 2d ed., Pearson Prentice Hall (2004).

[MACW77] F. J. MacWilliams and N. J. Sloane, *The Theory of Error-Correcting Codes*, North-Holland (1977).

[MASS69] J. L. Massey, ''Shift-Register Synthesis and BCH Decoding,'' *IEEE Trans. Info. Theory*, IT-15 (January 1969): 122–127.

[PETE72] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*, 2d ed., MIT Press (1972).

[PLES98] V. Pless, *Introduction to the Theory of Error-Correcting Codes*, 3d ed., Wiley Inter-Science (1998).

[REED60] I. S. Reed and G. Solomon, ''Polynomial Codes over Certain Finite Fields,'' *SIAM J. Appl. Math.*, 8 (June 1960): 300–304.

[REIG60] S. H, Reiger, ''Codes for the Correction of 'Clustered Errors','' *IRE Trans. Info. Theory*, IT-6 (March 1960): 16–21.

[STON61] J. J. Stone, ''Multiple Burst Error Correction,'' *Info. Cont.*, 4 (December 1961): 324–331.

[WOLF69] J. K. Wolf, ''Adding Two Information Symbols to Certain Nonbinary BCH Codes and Some Applications,'' *Bell Syst. Techn. J.*, 48 (September 1969): 2405–2424.

CONTENTS

3

Code Design Techniques for Matrix Codes

The high-speed digital systems that depend on error control codes require encoding / decoding to be performed in a parallel high-speed manner. In these systems the encoder / decoder is usually implemented by combinational logic circuits, not by linear feedback shift registers (LFSRs). In this case parity-check matrices are used to directly express the code functions. Matrix codes offer us the freedom to modify the organization of the matrices within the range of preserving the code functions. For example, we can select low-density column vectors to organize the matrix, or we can move or exchange the column vectors in the matrix. Such flexibilities do not exist in the polynomial codes.

The codes for high-speed systems are usually expressed by matrices, so many designs are possible. Most codes for high-speed memories are *shortened codes*, whose lengths are less than the theoretical bound under a given check-bit length. This is because the information-bit length of high-speed memories can be less than 300, for example, and this is short compared to the codes for mass memories and communication systems. There are various ways of shortening a code. A code designer may construct a shortened code to meet certain objectives or to satisfy some conditions suitable for a particular application, as was mentioned in Chapter 1. The objectives or conditions involve the optimization of other factors: encoder / decoder circuit amount, decoding circuit delay, probability of detecting multiple errors, or modularlized organization of an encoding / decoding circuit suitable for LSI implementation.

This chapter presents some practical matrix code designs suitable for the efficient high-speed parallel encoder / decoder. The techniques of this chapter can be applied to any matrix code design.

Code Design for Dependable Systems: Theory and Practical Applications, by Eiji Fujiwara

3.1 MINIMUM-WEIGHT & EQUAL-WEIGHT-ROW CODES

3.1.1 Code Concept

It should be easy to understand that fewer 1's in the **H** matrix means fewer modulo-2 additions that bring faster encoding / decoding. Fewer gates also means lower cost and more reliable hardware. Therefore it is preferable to design the **H** matrix for a given code satisfying as closely as possible the following constraints [HSIA70]:

1. *The total number of nonzero elements in **H** should be minimal, that is, take the form of a minimum-weight code, or a lowest density code.*
2. *The number of 1's in each row of **H** should be made equal, or as close as possible, to the average number of 1's (the total number of 1's in **H** divided by the number of rows), that is, take the form of an equal-weight-row code.*

Definition 3.1 A *minimum-weight & equal-weight-row code* is defined as a code whose **H** matrix has the minimum number of nonzero elements and each row of **H** expressed in binary form has equal number of 1's, or as close as possible, to the average number of 1's. □

3.1.2 Lowest Density MDS Codes

In the code expressed by the lowest density parity-check matrix, there exists the smallest number of nonzero elements. The code with a *low-density parity-check matrix* leads to an excellent high-speed decoder requiring a small amount of hardware and a high-speed error recovery. Minimizing the density is good for high-speed semiconductor memory systems as well as for disk array systems (e.g., for RAID systems as mentioned in Chapter 14).

Using this knowledge, Blaum and Roth [BLAU99] presented lower bounds on a number of nonzero elements in the parity-check matrix of a linear maximum distance separable (MDS) code over $GF(q^b)$, besides the upper bounds on the MDS code length that attains those lower bounds. In the following discussion we consider these bounds on the weight of a parity-check matrix, and code length, without proof. For the proof details, the reader should refer to the original paper [BLAU99].

However, codes designed by this method are not necessarily efficient in code length. This is because of the design constraints placed on the lowest density parity-check matrices.

Definition 3.2 (MDS Codes) Let $\boldsymbol{C}$ be a code of length n over $GF(q^b)$, and let $\boldsymbol{C}$ have the minimum Hamming distance d, where the distance is measured with respect to symbols of $GF(q^b)$. By the *Singleton bound* for codes over $GF(q^b)$ the codes that attain the following bound are called *maximum distance separable* (*MDS*) *codes*:

$$d \leq n + 1 - \log_{q^b}|C| = r + 1,$$

where $k = \log_{q^b}|C|$ is an information symbol length and $r = n - k$ is a check symbol length. □

The Reed-Solomon code (RS code) is a typical MDS code. The following discussion relates to the companion matrix and to the nonsingular matrix given in Subsection 2.2.3.

In order to construct low-density MDS codes over $GF(q^b)$, we need to use the largest possible set of $b \times b$ sparse matrices, that is, a set of low-density $b \times b$ matrices over $GF(q)$ that satisfies the following properties:

(**P1**) Each matrix in the set is nonsingular.
(**P2**) Every two distinct matrices in the set have a difference that is also nonsingular.
(**P3**) Each matrix contains b nonzero elements.
(**P3′**) Each matrix contains at most $b + 1$ nonzero elements.

Set of Matrices Satisfying (P1), (P2), and (P3) For a positive integer b and an element $\alpha \in GF(q)$, we define the $b \times b$ matrix $\mathbf{T}_\alpha$ over $GF(q)$ by

$$\mathbf{T}_\alpha = \begin{bmatrix} 0 & 0 & \cdots & 0 & \alpha \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{bmatrix}_{b \times b}.$$

The matrix $\mathbf{T}_\alpha$ is the companion matrix defined by the polynomial $\mathbf{g}(x) = x^b - \alpha$ over $GF(q)$. When $\alpha \in GF(q) - \{0\}$, the matrix $\mathbf{T}_\alpha$ is nonsingular and contains exactly b nonzero elements. The same holds for $\mathbf{T}_\alpha^i$, a power of $\mathbf{T}_\alpha$. Hence the set

$$U_\alpha = \left\{ \alpha \cdot \mathbf{T}_\alpha^i \,|\, \alpha \in GF(q) - \{0\},\ 0 \le i < b \right\}$$

satisfies (**P1**) and (**P3**), and the size of U_α is $b \cdot (q - 1)$.

Example 3.1 (3×3 Companion Matrices over $GF(3)$)

The 3×3 companion matrix over $GF(3)$ is defined by the polynomial $\mathbf{g}(x) = x^3 - 2$, and the set is given by $U_2 = \left\{ \mathbf{T}_2^0,\ \mathbf{T}_2^1,\ \mathbf{T}_2^2,\ \mathbf{T}_2^3 = 2 \cdot \mathbf{T}_2^0,\ \mathbf{T}_2^4 = 2 \cdot \mathbf{T}_2^1, \mathbf{T}_2^5 = 2 \cdot \mathbf{T}_2^2 \right\}$:

$$\mathbf{T}_2^0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{T}_2^1 = \begin{bmatrix} 0 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{T}_2^2 = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 0 & 2 \\ 1 & 0 & 0 \end{bmatrix},$$

$$\mathbf{T}_2^3 = 2 \cdot \mathbf{T}_2^0 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \quad \mathbf{T}_2^4 = 2 \cdot \mathbf{T}_2^1 = \begin{bmatrix} 0 & 0 & 1 \\ 2 & 0 & 0 \\ 0 & 2 & 0 \end{bmatrix},$$

$$\mathbf{T}_2^5 = 2 \cdot \mathbf{T}_2^2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 2 & 0 & 0 \end{bmatrix}.$$

The following theorem provides a necessary and sufficient condition for U_α to satisfy the property (**P2**).

Theorem 3.1 *Let α be an element of $GF(q) - \{0\}$ with exponent δ. The difference between every two distinct matrices in U_α is nonsingular if and only if every prime divisor of b divides δ but not $(q-1)/\delta$.*

When $q \not\equiv 3 \pmod 4$, the conditions of b in the theorem are necessary and sufficient for $x^b - \alpha$ to be an irreducible polynomial over $GF(q)$. When $q \equiv 3 \pmod 4$, the conditions for irreducibility require, in addition, that b is not divisible by 4. When $x^b - \alpha$ is irreducible, every nontrivial polynomial of degree less than b over $GF(q)$ is relatively prime to $x^b - \alpha$. Hence every nontrivial linear combination over $GF(q)$ of the matrices $\mathbf{I}$, $\mathbf{T}_\alpha$, $\mathbf{T}_\alpha^2$, ..., $\mathbf{T}_\alpha^{b-1}$, is nonsingular; in particular, irreducibility implies that the difference between every two distinct elements in U_α is nonsingular.

To obtain the widest range of the value b that satisfies the conditions of Theorem 3.1, we will choose α to be the primitive in $GF(q)$. In this case the conditions of the theorem require that the prime divisor of b also divides $q-1$. For example, we can take $b = 2^m$ when $q =$ odd, or $b = 3^m$ when $q = 2^{2h}$, where m and h are integers larger than or equal to 1.

Set of Matrices Satisfying (P1), (P2), and (P3′):

Definition 3.3 Let p be an odd prime, and let α be an element of $GF(q)$. For $0 \le i < p$, define the $(p-1) \times (p-1)$ matrix $\mathbf{Q}_\alpha^{(i)} = \left(\vartheta_{l,m}\right)_{l,m=1}^{p-1}$ over $GF(q)$ by

$$\vartheta_{l,m} = \begin{cases} 1 & \text{if} \quad l \neq p-i \text{ and } \langle m-l \rangle = i, \\ -1 & \text{if} \quad l = p-i \text{ and } m = i, \\ -\alpha & \text{if} \quad l = p-i \text{ and } m = \langle i/2 \rangle, \\ 0 & \text{otherwise}, \end{cases}$$

where $\langle a/b \rangle$ denotes the unique integer σ, $0 \le \sigma < p$, such that $a \equiv b\sigma \pmod p$.

The matrix $\mathbf{Q}_\alpha^{(0)}$ is the identity matrix $\mathbf{I}_{p-1}$, and $\mathbf{Q}_\alpha^{(1)}$ is the transposed companion matrix of the polynomial $x^{p-1} + \alpha x^{(p-1)/2} + 1$ over $GF(q)$. □

Example 3.2 [BLAU99]

For $p = 5$ and $\alpha \in GF(q)$, we have $\langle 1/2 \rangle = 3$, $\langle 2/2 \rangle = 1$, $\langle 3/2 \rangle = 4$. The matrices $\mathbf{Q}_\alpha^{(i)}$ are given by

$$\mathbf{Q}_\alpha^{(0)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{Q}_\alpha^{(1)} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & -\alpha & 0 \end{bmatrix}, \quad \mathbf{Q}_\alpha^{(2)} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\alpha & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{Q}_\alpha^{(3)} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & -\alpha \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{Q}_\alpha^{(4)} = \begin{bmatrix} 0 & -\alpha & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

For example, the element in the crosspoint of the fourth row and the third column in $\mathbf{Q}_{\alpha}^{(1)}$ (i.e., $\vartheta_{4,3}$) is determined as $\vartheta_{l,m} = -\alpha$ for $l = 5 - 1 = 4$ and $m = \langle 1/2 \rangle = 3$.

The matrix $\mathbf{Q}_{\alpha}^{(0)}$ contains $p - 1$ nonzero elements and each of the remaining matrices in the set contains at most p nonzero elements. Hence the set

$$\Omega_{\alpha} = \left\{ a \cdot \mathbf{Q}_{\alpha}^{(i)} \mid a \in GF(q) - \{0\},\ 0 \le i < p \right\}$$

satisfies the property (**P3**′). Note that the size of Ω_{α} is

$$p(q-1) = (b+1)(q-1).$$

For $q = 3$ in Example 3.2, the size of Ω_{α} is 10, and additional five matrices of $2 \cdot \mathbf{Q}_2^{(0)}$, $2 \cdot \mathbf{Q}_2^{(1)}$, $2 \cdot \mathbf{Q}_2^{(2)}$, $2 \cdot \mathbf{Q}_2^{(3)}$, $2 \cdot \mathbf{Q}_2^{(4)}$ are added to the existing five matrices of $\mathbf{Q}_2^{(i)}$, $0 \le i \le 4$. The following theorem provides sufficient conditions on p and α, showing that Ω_{α} satisfies the property (**P2**).

Theorem 3.2 *Let p be a prime such that $p - 1$ is divisible by $2(q-1)$, and let α be an element in $GF(q) - \{0\}$ such that the polynomial $x^2 + \alpha x + 1$ is irreducible over $GF(q)$. Then the difference of any two distinct matrices in Ω_{α} is nonsingular.*

Lowest Density Bounds on MDS Codes We present here lower bounds on the number of nonzero elements in the parity-check matrices of the linear MDS codes over $GF(q^b)$, as well as upper bounds on dimension and redundancy of the codes that attain those bounds.

Theorem 3.3 *Let $\boldsymbol{C}$ be a linear $(n,\ k = n - r)$ MDS code over $GF(q^b)$, and suppose that $\boldsymbol{C}$ has an $rb \times nb$ systematic parity-check matrix $\boldsymbol{H}$ having at least $k + 1$ nonzero elements in each row. If $k > 1$ and $r > 1$, then $k \le b(q-1)$ and $r \le b(q-1)$.*

For $q = 2$, we can improve these bounds.

Theorem 3.4 *Let $\boldsymbol{C}$ be a linear $(n,\ k = n - r)$ MDS code over $GF(2^b)$, and assume that $k \ge r \ge 2$. The average number of 1's in each row of the systematic parity-check matrix of $\boldsymbol{C}$* is at least $k + 1 + (k-1)/2b$, which is attained in the case $k \le b + 1$.

Theorem 3.5 *Let $\boldsymbol{C}$ be a linear $(n,\ k = n - r)$ MDS code over $GF(2^b)$, and assume that $k \ge r \ge 3$. The average number of 1's in each row of the systematic parity-check matrix of $\boldsymbol{C}$ is at least $k + 1 + (2k-3)/3b$.*

Example 3.3 [BLAU99]

Let $k = r = 3$ and $b > 1$, and let $\mathbf{A}$ be an $rb \times kb$ matrix over $GF(2)$:

$$\mathbf{A} = \begin{bmatrix} \mathbf{D} & \mathbf{I}_b & \mathbf{I}_b \\ \mathbf{I}_b & \mathbf{D} & \mathbf{I}_b \\ \mathbf{I}_b & \mathbf{I}_b & \mathbf{D} \end{bmatrix},$$

where $\mathbf{I}_b$ is a $b \times b$ binary identity matrix and $\mathbf{D} = (d_{l,m})$ is a $b \times b$ matrix over $GF(2)$ of the form

$$d_{l,m} = \begin{cases} 1 & \text{if } l < b \text{ and } m = l + 1, \\ 1 & \text{if } l = b \text{ and } m \in \{1,\ t\} \text{ for some } 2 \le t \le b,\ 1 \le l, m \le b, \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to check that both $\mathbf{D}$ and $\mathbf{D} + \mathbf{I}_b$ are nonsingular. This implies that every square submatrix of $\mathbf{A}$ that consists of full $b \times b$ submatrices $\mathbf{D}$ or $\mathbf{I}_b$, is nonsingular. Hence the following $\mathbf{H} = [\,\mathbf{A} \vdots \mathbf{I}_{3b}\,]$ is a systematic parity-check matrix of a linear $(6, 3)$ MDS code over $GF(2^b)$, and the average number of 1's in each row is $4 + 1/b$, thus attaining lower bounds.

$$\mathbf{H} = \left[\ \mathbf{A} \vdots \mathbf{I}_{3b}\ \right] = \left[\begin{array}{ccc:ccc} \mathbf{D} & \mathbf{I}_b & \mathbf{I}_b & \mathbf{I}_b & \mathbf{O} & \mathbf{O} \\ \mathbf{I}_b & \mathbf{D} & \mathbf{I}_b & \mathbf{O} & \mathbf{I}_b & \mathbf{O} \\ \mathbf{I}_b & \mathbf{I}_b & \mathbf{D} & \mathbf{O} & \mathbf{O} & \mathbf{I}_b \end{array}\right] :\ (6,\ 3)\ \text{MDS code.}$$

For $b = 4$, submatrix $\mathbf{D}$ can be expressed as

$$\begin{array}{ccc} t = 2 & t = 3 & t = 4 \\ \left[\begin{array}{cccc} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{array}\right] & \left[\begin{array}{cccc} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{array}\right] & \left[\begin{array}{cccc} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{array}\right] \end{array}.$$

The following shows the lowest density systematic parity-check matrix of the (6, 3) MDS codes over $GF(2^4)$:

$$\mathbf{H} = \left[\begin{array}{ccc:ccc} 0100 & 1000 & 1000 & 1000 & 0000 & 0000 \\ 0010 & 0100 & 0100 & 0100 & 0000 & 0000 \\ 0001 & 0010 & 0010 & 0010 & 0000 & 0000 \\ 1100 & 0001 & 0001 & 0001 & 0000 & 0000 \\ 1000 & 0100 & 1000 & 0000 & 1000 & 0000 \\ 0100 & 0010 & 0100 & 0000 & 0100 & 0000 \\ 0010 & 0001 & 0010 & 0000 & 0010 & 0000 \\ 0001 & 1100 & 0001 & 0000 & 0001 & 0000 \\ 1000 & 1000 & 0100 & 0000 & 0000 & 1000 \\ 0100 & 0100 & 0010 & 0000 & 0000 & 0100 \\ 0010 & 0010 & 0001 & 0000 & 0000 & 0010 \\ 0001 & 0001 & 1100 & 0000 & 0000 & 0001 \end{array}\right].$$

The average number of 1's in each row equals $4 + 1/4$, which is equal to $k + 1 + (2k - 3)/3b$ for $k = 3$ and $b = 4$.

3.2 ODD-WEIGHT-COLUMN CODES

Let $\mathbf{H}$ be a parity-check matrix of (n, k) code over $GF(2^b)$ having $r = n - k$ rows and n columns.

Definition 3.4 If every column in the **H** matrix of code $\boldsymbol{C}$ over $GF(2^b)$ satisfies the following condition (3.1), then $\boldsymbol{C}$ is called an *odd-weight-column code*:

$$\sum_{i=0}^{r-1} h_{i,j} = I \quad \text{for columns } j \;=\; 0, 1, \ldots, n-1, \tag{3.1}$$

where

$h_{i,j}$: i-th element in the j-th column vector $\in GF(2^b)$,
I: identity element in $GF(2^b)$,
$\sum$: summation in $GF(2^b)$. □

To convert the (n, k) code over $GF(2^b)$ to binary form, we replace each element of $GF(2^b)$ in **H** to a corresponding $b \times b$ matrix over $GF(2)$. Then the **H** matrix is an $R \times N$ binary matrix, where $R = b \cdot r$ and $N = b \cdot n$.

Theorem 3.6 *[FUJI78, FUJI81] If the **H** matrix of a code over GF(2^b) satisfies Eq. (3.1), then its corresponding binary converted form of matrix is an odd-weight-column matrix over GF(2). In other words, if **H** is an $r \times n$ matrix over $GF(2^b)$ that satisfies Eq. (3.1), then the binary conversion of **H** will yield an $R \times N$ binary odd-weight-column matrix.*

Proof Let $h_{i,j}$ be the i-th row and the j-th column element of the $r \times n$ matrix **H** over $GF(2^b)$, where $0 \le i \le r-1$, $0 \le j \le n-1$. Every element of $GF(2^b)$ can be expressed by a $b \times b$ binary matrix, called its *companion matrix*, as further explained in Chapter 5. In particular, the identity element **I** of $GF(2^b)$ is equivalent to the $b \times b$ identity matrix. Now write $\mathbf{h}_{i,j}$ as the following binary $b \times b$ matrix:

$$\mathbf{h}_{i,j} = \begin{bmatrix} & & \overset{l}{\downarrow} & & \\ (a_{0,0})_i & \cdots & (a_{0,l})_i & \cdots & (a_{0,b-1})_i \\ & & (a_{1,l})_i & & \\ \vdots & & \vdots & & \vdots \\ & \cdots & (a_{l,l})_i & \cdots & \\ \vdots & & \vdots & & \vdots \\ (a_{b-1,0})_i & \cdots & (a_{b-1,l})_i & \cdots & (a_{b-1,b-1})_i \end{bmatrix} \leftarrow l,$$

$$(a_{s,l})_i \in GF(2),$$
$$0 \le s,\ l \le b-1,\ 0 \le i \le r-1.$$

Equation (3.1) leads to the following relation for $0 \le l \le b-1$:

$$\sum_{i=0}^{r-1}{}^{\oplus}(a_{l,l})_i = 1, \qquad \sum^{\oplus} : \textit{mod-2 sum}.$$

This shows that the modulo-2 sum of the corresponding diagonal elements in the binary square matrices $\mathbf{h}_{i,j}$'s is equal to 1 of the corresponding element in the identity matrix $\mathbf{I}$. Thus the binary represented set $\{(a_{l,l})_0, (a_{l,l})_1, \ldots, (a_{l,l})_{r-1}\}$ has an odd number of l's, meaning it is odd weight.

On the other hand, the following relation holds for $0 \leq m \neq l \leq b-1$:

$$\sum_{i=0}^{r-1}{}^{\oplus}(a_{m,l})_i = 0.$$

This shows that the modulo-2 sum of the corresponding nondiagonal elements in the binary square matrices $\mathbf{h}_{i,j}$'s is equal to 0 of the corresponding element in the identity matrix $\mathbf{I}$. Thus the binary represented set $\{(a_{m,l})_0, (a_{m,l})_1, \ldots, (a_{m,l})_{r-1}\}$, where $m \neq l$, has an even number of l's, meaning it is even weight. The foregoing equations result in the relation

$$\sum_{i=0}^{r-1}{}^{\oplus}\sum_{m=0}^{b-1}{}^{\oplus}(a_{m,l})_i = 1.$$

This shows that the l-th column vector of the binary $\mathbf{H}$ matrix is of odd weight. These relations hold for any integer l and j. Hence every column vector of the binary $\mathbf{H}$ matrix that satisfies Eq. (3.1) is of odd weight. Q.E.D.

The odd-weight-column code gives a good discrimination of even number and odd number of errors. Therefore it has better multiple error detection capability than the non–odd-weight-column code, as will be shown in later chapters.

3.3 EVEN-WEIGHT-ROW CODES

Definition 3.5 Let the binary $\mathbf{H}$ matrix of code $\mathbf{C}$ be expressed by r row vectors as

$$\mathbf{H} = \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_{r-1} \end{bmatrix}.$$

If every nonzero row vector $P_i = (h_{i,0} \ \ldots \ h_{i,n-1})$ of $\mathbf{H}$ satisfies Eq. (3.2), then $\mathbf{C}$ is called an *even-weight-row code*:

$$\sum_{j=0}^{n-1} h_{i,j} = 0 \qquad \text{for rows } i = 0, 1, \ldots, r-1, \tag{3.2}$$

Where

$h_{i,j}$: j-th element in the i-th row vector $\in GF(2^b)$,

0: zero element in $GF(2^b)$,

$\sum$: summation in $GF(2^b)$. □

It can be easily proved that the code satisfying Eq. (3.2) has all even-weight rows in the binary form of **H**. This code has an important characteristic given below.

Definition 3.6 For every binary codeword **W** of code **C**, if its bitwise complement $\overline{W}$ is also in **C**, then **C** is called a *self-complementing* code. □

Theorem 3.7 *Even-weight-row code **C** is a self-complementing code.*

Proof Let the codeword **W** of code **C** be expressed as

$$\mathbf{W} = [D \mid P] = [d_0 \ d_1 \ \ldots \ d_{k-1} \ p_0 \ p_1 \ \ldots \ p_{r-1}],$$

where $D = (d_0 \ d_1 \ \ldots \ d_{k-1})$ is an information part and $P = (p_0 \ p_1 \ \ldots \ p_{r-1})$ is a check part. In addition let the **H** matrix be expressed as $\mathbf{H} = [\mathbf{H}', \ \mathbf{I}]$, where $\mathbf{H}'$ is an $r \times k$ binary matrix for information part of **H**, and **I** is an $r \times r$ binary identity matrix for the check part of **H**. Clearly, $\mathbf{H}'$ has r odd-weight-row vectors, $h'_0, h'_1, \ldots, h'_{r-1}$. Therefore each check bit can be obtained by the following relation:

$$p_i = (d_0 \ d_1 \ \ldots \ d_{k-1}) \cdot (h'_i)^T, \qquad 0 \le i \le r-1.$$

Let the bitwise complement of W be $\overline{W} = [\overline{d_0} \ \overline{d_1} \ \ldots \ \overline{d_{k-1}} \ \overline{c_0} \ \ldots \ \overline{c_{r-1}}] = [\overline{D} \mid \overline{P}]$. For $\overline{D} = (\overline{d_0} \ \overline{d_1} \ \ldots \ \overline{d_{k-1}})$, the following relation always holds because every row vector h'_i is of odd weight:

$$(\overline{d_0} \ \overline{d_1} \ \ldots \ \overline{d_{k-1}}) \cdot (h_i'^T) = \overline{p_i}, \qquad 0 \le i \le r-1.$$

This means that $\overline{D} \cdot H'^T = \overline{P}$, and hence the code is self-complementing. Q.E.D.

From the characteristic of the code given above, an even-weight-row SEC-DED code can be used for *mask error correction* of double errors in memory words [SUND78, SUND79, WALK79, AICH84] (see Example 4.3). This code can also be applied to memory testing because it allows a small number of memory cell accesses to test the semiconductor memory modules. Another application is mask error correction in logic circuits, especially in self-complementary logic circuits, which is called *alternate data retry* (ADR) [YAMA70, SHED78], as we saw in Subsection 1.3.2.

Several well-known codes, such as m-out-of-$2m$ codes, complemented duplication codes, residue codes with checkbase $2^a - 1$ (called *low-cost residue codes*), and self-complementing $AN + B$ codes (e.g., $3N + 2$ code [BROW60, RAO74]) also satisfy the condition that the bitwise complement of the codeword is a codeword as well.

Corollary 3.1 *If the **H** matrix of a code **C** satisfies both odd-weight-column and even-weight-row conditions, then the code length of **C*** is even.

The reader is encouraged to prove this.

Example 3.4

A simple example of the even-weight-row code is the code with its **H** matrix:

$$\mathbf{H} = \left[\begin{array}{cccc|cccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right].$$

All row vectors have even-weight 4. The codeword $W = [D \mid P]$ is shown below. There exist 16 codewords.

$$\mathrm{W} = \left[\begin{array}{cc} D & P \end{array}\right]$$

$$= \left[\begin{array}{cc} 0\,0\,0\,0 & 0\,0\,0\,0 \\ 1\,0\,0\,0 & 1\,1\,1\,0 \\ 0\,1\,0\,0 & 0\,1\,1\,1 \\ 1\,1\,0\,0 & 1\,0\,0\,1 \\ 0\,0\,1\,0 & 1\,0\,1\,1 \\ 1\,0\,1\,0 & 0\,1\,0\,1 \\ 0\,1\,1\,0 & 1\,1\,0\,0 \\ 1\,1\,1\,0 & 0\,0\,1\,0 \\ \hline 0\,0\,0\,1 & 1\,1\,0\,1 \\ 1\,0\,0\,1 & 0\,0\,1\,1 \\ 0\,1\,0\,1 & 1\,0\,1\,0 \\ 1\,1\,0\,1 & 0\,1\,0\,0 \\ 0\,0\,1\,1 & 0\,1\,1\,0 \\ 1\,0\,1\,1 & 1\,0\,0\,0 \\ 0\,1\,1\,1 & 0\,0\,0\,1 \\ 1\,1\,1\,1 & 1\,1\,1\,1 \end{array}\right] \begin{array}{c} \uparrow \\ \cdots \\ \downarrow \end{array}$$

These codewords are self-complementary, as is obvious.

3.4 ODD-WEIGHT-ROW CODES

Next we consider the codes whose **H** matrix rows are of odd weight. These codes satisfy the following definition.

Definition 3.7 If every binary row vector $(h_{i,0} \ \ldots \ h_{i,n-1})$ of the **H** matrix of code **C** satisfies Eq. (3.3), then **C** is called an *odd-weight-row code*.

$$\sum_{j=0}^{n-1} h_{i,j} = 1 \qquad \text{for rows } i = 0, 1, \ldots, r-1. \tag{3.3}$$

□

Theorem 3.8 *For a binary codeword $W = [D \mid P]$ of an odd-weight-row code **C**, where D is the information part and P is the check part, $W' = [\bar{D} \mid P]$ is also a codeword of **C**.*

The proof is similar to the previous one and hence is omitted.

An odd-weight-column and odd-weight-row SEC-DED code, called a *Maintenance code (M code)*, can be shown to be useful for mask error correction in memory units [CART76].

Example 3.5

A simple example of the odd-weight-row code is the code with its **H** matrix:

$$\mathbf{H} = \left[\begin{array}{cc} 1\,0\,0\,1\,1\,1 & 1\,0\,0\,0\,0 \\ 1\,0\,1\,0\,1\,1 & 0\,1\,0\,0\,0 \\ 1\,1\,0\,1\,0\,1 & 0\,0\,1\,0\,0 \\ 0\,1\,1\,0\,1\,1 & 0\,0\,0\,1\,0 \\ 0\,1\,1\,1\,0\,1 & 0\,0\,0\,0\,1 \end{array}\right].$$

In these codewords, even if the information part is bitwise complemented, the corresponding check part remains same:

$$\begin{array}{cc} \text{Codewords} & \\ \left[\begin{array}{cc} 0\,0\,0\,0\,0\,0 & 0\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0\,0 & 1\,1\,1\,0\,0 \\ 0\,1\,0\,0\,0\,0 & 0\,0\,1\,1\,1 \\ 1\,1\,0\,0\,0\,0 & 1\,1\,0\,1\,1 \\ 0\,0\,1\,0\,0\,0 & 0\,1\,0\,1\,1 \\ 1\,0\,1\,0\,0\,0 & 1\,0\,1\,1\,1 \\ 0\,1\,1\,0\,0\,0 & 0\,1\,1\,0\,0 \\ 1\,1\,1\,0\,0\,0 & 1\,0\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0 & 1\,0\,1\,0\,1 \\ 1\,0\,0\,1\,0\,0 & 0\,1\,0\,0\,1 \\ \vdots & \\ D & P \end{array}\right] & \Longleftrightarrow \end{array} \quad \begin{array}{cc} \text{Codewords} & \\ \left[\begin{array}{cc} 1\,1\,1\,1\,1\,1 & 0\,0\,0\,0\,0 \\ 0\,1\,1\,1\,1\,1 & 1\,1\,1\,0\,0 \\ 1\,0\,1\,1\,1\,1 & 0\,0\,1\,1\,1 \\ 0\,0\,1\,1\,1\,1 & 1\,1\,0\,1\,1 \\ 1\,1\,0\,1\,1\,1 & 0\,1\,0\,1\,1 \\ 0\,1\,0\,1\,1\,1 & 1\,0\,1\,1\,1 \\ 1\,0\,0\,1\,1\,1 & 0\,1\,1\,0\,0 \\ 0\,0\,0\,1\,1\,1 & 1\,0\,0\,0\,0 \\ 1\,1\,1\,0\,1\,1 & 1\,0\,1\,0\,1 \\ 0\,1\,1\,0\,1\,1 & 0\,1\,0\,0\,1 \\ \vdots & \\ \overline{D} & P \end{array}\right] \end{array}.$$

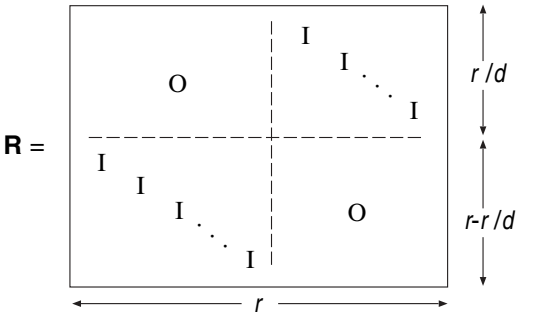
3.5 ROTATIONAL CODES

Rotational codes offer modularity of the encoding / decoding circuits with small additional check bits, and hence they are practical for LSI / VLSI implementation of the encoder / decoder [FUJI80].

3.5.1 Code Concept

Definition 3.8 A code whose **H** matrix has the following d submatrices, each having r rows and n/d columns is called a *rotational code*. In this case, d is a divisor of r and of n.

$$\begin{aligned} \mathbf{H} &= \left[\mathbf{H}_0|\mathbf{H}_1|\ldots|\mathbf{H}_j|\ldots|\mathbf{H}_{d-1}\right]_{r\times n}, \\ \mathbf{H}_j &= R^j \cdot \mathbf{H}_0, \quad j = 1, 2, \ldots, d-1, \end{aligned} \tag{3.4}$$



$\{O, I\} \in GF(2^b)$.

□

In this definition $\mathbf{H}_0$ and $\mathbf{R}$ are called the *generating submatrix* and the *rotational operating matrix*, respectively. The $\mathbf{H}$ matrix design of the rotational codes presents d cyclic shifts of the r/d rows in a vertical direction between adjacent submatrices.

The unique feature of the rotational code is that its encoding / decoding circuit can be implemented with d identical subcircuits specified by the generating submatrix $\mathbf{H}_0$, each by simply altering the input / output connections [CART73, BOSS74, FUJI77]. Therefore the rotational code gives a modularized encoding / decoding circuit suitable for LSI implementation [FUJI80].

In order to understand the concept of rotational code, a simple example is presented next.

Example 3.6

The code expressed by the following $\mathbf{H}$ matrix can be made rotational by permuting its column vectors as shown below. In this case $d = r = 4$. Figure 3.1 shows the modularized decoding circuit based on $\mathbf{H}'$.

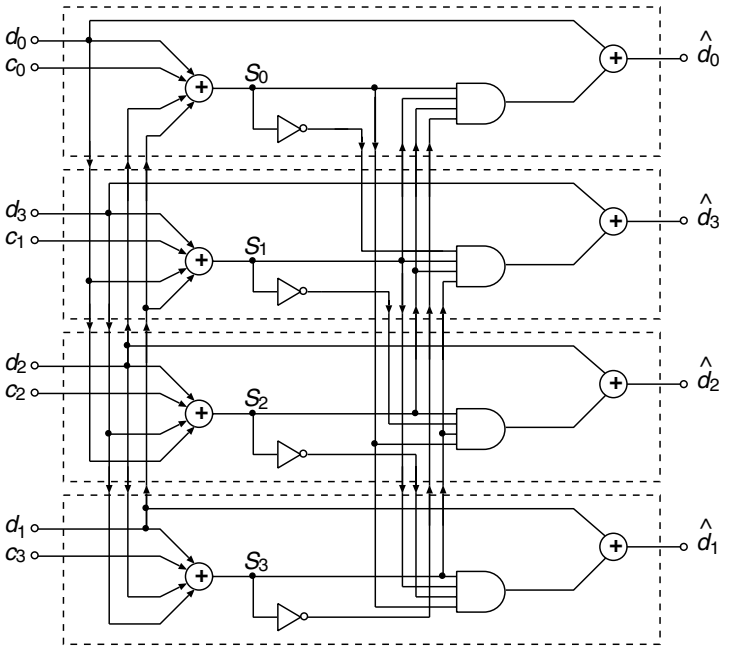


Figure 3.1 *Modularized decoding circuit for the rotational code* $\mathbf{H}'$.

$$\mathbf{H} = \begin{array}{c} \begin{matrix} d_0 & d_1 & d_2 & d_3 & c_0 & c_1 & c_2 & c_3 \end{matrix} \\ \left[\begin{array}{cccccccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \end{array}$$

$$\Downarrow$$

$$\mathbf{H}' = \begin{array}{c} \begin{matrix} d_0 & c_0 & d_3 & c_1 & d_2 & c_2 & d_1 & c_3 \end{matrix} \\ \left[\begin{array}{cc:cc:cc:cc} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{array}\right] \\ \mathbf{H}_0 \quad \mathbf{H}_1 \quad \mathbf{H}_2 \quad \mathbf{H}_3 \end{array}, \quad \mathbf{H}_0 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad ,$$

$$\{0,1\} \in GF(2),$$

$$\mathbf{H}_j = \mathbf{R}^j \cdot \mathbf{H}_0, \quad j = 1,2,3,$$

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

Figure 3.1 shows four identical subscircuits, each having seven inputs and four outputs including signals communicating with other subcircuits. Also shown are the four-input parity checker, an inverter, a four-input AND gate, and an exclusive-OR (XOR) gate to give the whole decoding circuit specified by the rotational (8, 4) SEC-DED code. Modularity is the main feature of this decoder. In Chapter 4 the original (8, 4) odd-weight-column SEC-DED code will be described in Example 4.1. Its decoding circuit shown in Figure 4.1 is transformed here to the rotational code $\mathbf{H}'$ and its modularized decoding circuit shown in Figure 3.1. The matrix $\mathbf{H}'$ is identical to the original matrix $\mathbf{H}$ because the column vectors in $\mathbf{H}'$ are just a permutation of those in $\mathbf{H}$.

3.5.2 Maximum Code Length of Rotational Codes

A simple example of the rotational code is a rotational single-bit error correcting (SEC) code. Other rotational memory codes, such as the rotational single-byte error correcting (S*b*EC) codes, rotational single-byte error correcting and double-byte error detecting (S*b*EC-D*b*ED) codes, will be demonstrated in later chapters.

Definition 3.9 Let the i-th cyclic shift of a vector $H = (h_0\ h_1\ \ldots\ h_{r-1})$ be denoted by

$$H^{(i)} = (h_i\ h_{i+1}\ \ldots\ h_{r-1}\ h_0\ \ldots\ h_{i-1}).$$

A set of all distinct vectors obtained by cyclic shifts of a vector is called a *cyclic equivalence class*. □

Example 3.7

The cyclic equivalence classes of binary 4-tuples sequences are as follows:

$$\begin{aligned} E_1 &= \{0000\}, \\ E_2 &= \{1111\}, \\ E_3 &= \{1010, 0101\}, \\ E_4 &= \{0001, 0010, 0100, 1000\}, \\ E_5 &= \{0011, 0110, 1100, 1001\}, \\ E_6 &= \{0111, 1110, 1101, 1011\}. \end{aligned}$$

Each cyclic equivalence class (of r-tuples) will have m vectors, where m is a divisor of r. If $m \neq r$ for any class, then it is called a *degenerate cyclic equivalence class*. In the example, E_1, E_2, and E_3 are degenerate cyclic equivalence classes, whereas E_4, E_5, and E_6 are *nondegenerate cyclic equivalence classes*.

In general, for r and q-ary sequence the number of nondegenerate cyclic equivalence classes, $N_q(r)$, is given as follows [GOLO58]:

$$N_q(r) = \frac{1}{r} \sum_{m|r} \mu(m) \cdot q^{r/m}. \tag{3.5}$$

Here $\sum_{m|r}$ expresses summation over all m that divides r, and $\mu(m)$ is *Möbius function* [PETE72] defined as

$$\begin{aligned} \mu(m) = 1 &= 1, \qquad m = 1, \\ &= 0, \qquad m \text{ has any square factor}, \\ &= (-1)^l, \qquad m = p_1 p_2 \ldots p_l, \text{where } p_i\text{'s are distinct primes.} \end{aligned}$$

Values of $\mu(m)$ for $m = 1$ to 15 are shown below:

m	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mu(m)$	1	−1	−1	0	−1	1	−1	0	0	1	−1	0	−1	1	1

The following scheme shows $N_2(r)$, which is the number of nondegenerate cyclic equivalence classes for binary r-tuples:

r	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$N_2(r)$	2	3	6	9	18	30	56	99	186	335	630	1,161	2,182	4,096

In the case of Example 3.7, $N_2(4) = 3$ by this table. The $\mathbf{H}$ matrix of a rotational SEC code can be constructed by using the nondegenerate cyclic equivalence classes as follows:

$$\mathbf{H} = \begin{array}{c} \begin{array}{cccccccccccc} c_0 & d_0 & d_1 & c_1 & d_2 & d_3 & c_2 & d_4 & d_5 & c_3 & d_6 & d_7 \end{array} \\ \left[\begin{array}{ccc:ccc:ccc:ccc} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \end{array}\right] \\ \mathbf{H}_0 \quad\quad \mathbf{H}_1 \quad\quad \mathbf{H}_2 \quad\quad \mathbf{H}_3 \end{array}.$$

Here we consider the code length of a rotational SEC code. Let the number of cyclic equivalence classes for a binary sequence having length m be $N_2(m)$, where m is a divisor of r.

In Example 3.7, we have $N_2(1) = 2$, $N_2(2) = 1$, and $N_2(4) = 3$. Therefore the following relation is satisfied for the cyclic equivalence classes of binary 4-tuples:

$$1 \cdot N_2(1) + 2 \cdot N_2(2) + 4 \cdot N_2(4) = 2^4$$

In general, for binary r-tuples the following relation holds:

$$2^r = \sum_{m|r} m \cdot N_2(m). \tag{3.6}$$

In order to obtain the code length of a rotational SEC code, we are required to find the number of nondegenerate cyclic equivalence class, $N_2(r)$.

For functions f and g over integers m and r, we have the following inversion formula.

Definition 3.10 If for an arbitrary positive integer r the relation

$$f(r) = \sum_{m|r} g(m)$$

holds, then $g(r)$ can be expressed by $f(r)$ as

$$g(r) = \sum_{m|r} \mu(m) \cdot f\left(\frac{r}{m}\right).$$

This inversion formula is called *Möbius inversion formula*. □

We can apply a Möbius inversion formula to Eq. (3.6) and then obtain the following equation:

$$N_2(r) = \frac{1}{r} \sum_{m|r} \mu(m) \cdot 2^{r/m}$$

TABLE 3.1 Code Length of Rotational SEC Codes

r	$n_{\text{rotational}}$	$n_{\text{nonrotational}}$
3	6	7
4	12	15
5	30	31
6	54	63
7	126	127
8	240	255
9	504	511
10	990	1,023
11	2,046	2,047
12	4,020	4,095

Note: $n_{\text{rotational}} = \sum_{m|r} \mu(m) \cdot 2^{r/m}$; $n_{\text{nonrotational}} = 2^r - 1$.

This is equal to Eq. (3.5) for $q = 2$. Therefore the code length n of a rotational SEC code can be expressed as

$$n = r \cdot N_2(r) = \sum_{m|r} \mu(m) \cdot 2^{r/m}. \tag{3.7}$$

Table 3.1 shows the code length in bits of this rotational code as well as that of the nonrotational Hamming SEC code. This table says that the code length of the rotational SEC codes is decreased by small number of bits, compared to that of the nonrotational codes.

EXERCISES

3.1 Use Theorem 3.1 for the following exercises. Let α be an element in $GF(q) - \{0\}$ with exponent δ.

(a) Find the exponent δ for every element α in $GF(7) - \{0\}$, that is, $\alpha = 1,\ 2,\ 3,\ 4,\ 5$, and 6.

(b) Verify that prime divisor of b divides δ but not $(q-1)/\delta$ for $q = 7$, $b = 4$, and $\alpha = 3$ and 5.

(c) Find the set U_α over $GF(7)$ for $b = 4$ and $\alpha = 5$.

(d) Verify that the difference between every two distinct matrices in U_α is nonsingular.

3.2 Verify Theorem 3.2 for $p = 3$ and $q = 2$.

3.3 Design the lowest density MDS code over $GF(2^4)$ with $r = 2$ that satisfies the conditions of Theorem 3.4.

3.4 Prove Corollary 3.1.

3.5 Design the **H** matrix of a rotational odd-weight-column (30, 24) SEC-DED code, and design its modularized decoding circuit.

3.6 Design the generating submatrix $\mathbf{H}_0$ of a rotational odd-weight-column (63, 56) SEC-DED code.

3.7 The length between 1's in a binary vector is called a *gap length* [BOSS74]. A gap length is defined as the number of 0's between the adjacent two 1's in the vector. The gap length notation of a vector with length r and Hamming weight w is a w-tuple $(l_1, l_2, \ldots, l_w)$ where l_i denotes the i-th gap length between the i-th 1 and the $(i+1)$-th 1 in the vector in a cyclic order. For example, vector (010110010) with $r = 9$ and $w = 4$ has the gap lengths $l_1 = 1$, $l_2 = 0$, $l_3 = 2$, and $l_4 = 2$. In particular, the gap length between the fourth 1 and the first 1 rounded through the end of the example vector gives $l_4 = 2$ because there are two 0's between the forth 1 and the first 1 in the vector. Hence the 4-tuple gap length vector of this example vector can be expressed as (1022).

Under the foregoing preparation, do the following:

(a) Show that the relation between the parameters of l_i, r, and w is expressed as

$$\sum_{i=1}^{w} l_i = r - w.$$

(b) The gap length notation of the vector completely characterizes the vector up to its r cyclic shifts. Prove that the vector included in a nondegenerate cyclic equivalence class (NCEC) has the gap length vector that is also included in NCEC, and vice versa. Since the example vector (010110010) above has the gap length vector (1022), which is included in NCEC, the original example vector is included in NCEC.

(c) The vector with parameters of $r = 7$ and $w = 2$, denoted as $(r,\ w) = (7,\ 2)$, has the number of 0's equal to $r - w = 7 - 2 = 5$. Consequently the gap length vector of (5, 0), which is included in NCEC, can be easily obtained. We can get another two gap length vectors of (4, 1) and (3, 2) also included in NCEC. Based on this process, find the 7 gap length vectors included in NCEC for $(r,\ w) = (8,\ 3)$, the 8 gap length vectors for $(r,\ w) = (8,\ 4)$, and the 12 gap length vectors for $(r,\ w) = (10,\ 3)$.

(d) Prove that if the vector with $(r,\ w)$ is included in NCEC, then the complemented vector with $(r,\ r - w)$ is also included in NCEC.

(e) Using the column vectors with $(r,\ w) = (8,\ 1)$, $(8,\ 3)$, $(8,\ 5)$, and $(8,\ 7)$, design the rotational odd-weight-column (128, 120) code; that is, design the 8×16 generating submatrix of the code.

REFERENCES

[AICH84] F. J. Aichelmann Jr., "Fault-Tolerant Design Techniques for Semiconductor Memory Applications," *IBM J. Res. Dev.*, 28 (March 1984): 177–183.

[BLAU99] M. Blaum and R. M. Roth, "On Lowest Density MDS Codes," *IEEE Trans. Info. Theory*, 45 (January 1999): 46–59.

[BOSS74] D. C. Bossen, S. J. Hong, M. Y. Hsiao, and A. M. Patel, ''Modular Distributed Error Detection and Correction Apparatus and Method,'' US Patent 3825893 (July 23, 1974).

[BROW60] D. T. Brown, ''Error Detecting and Correcting Binary Codes for Arithmetic Operations,'' *IEEE Trans. Electron. Comput*., EC-9 (September 1960): 333–337.

[CART73] W. C. Carter, K. A. Duke, and D. C. Jessep Jr., ''Lookaside Techniques for Minimum Circuit Memory Translators,'' *IEEE Trans. Comput*., C-22 (March 1973): 283–289.

[CART76] W. C. Carter and C. E. McCarthy, ''Implementation of an Experimental Fault-Tolerant Memory System,'' *IEEE Trans. Computer*, C-25 (June 1976): 557–568.

[FUJI77] E. Fujiwara, ''A Modularized *b*-Adjacent Error Correction Memory Unit,'' *Trans. IECE Japan*, E60 (February 1977): 69–76.

[FUJI78] E. Fujiwara, ''Odd-Weigh-Column *b*-Adjacent Error Correcting Codes,'' *Trans. IECE Japan*, E61 (October 1978): 781–787.

[FUJI80] E. Fujiwara and K. Haruta, '' Design of Main Storage Error Checking and Correcting Circuit for LSI Implementation'' (in Japanese), *Trans. IECE Japan*, 63-D (February 1980): 129–136.

[FUJI81] E. Fujiwara, ''Error Correcting Code and Its Application to Digital Systems'' (in Japanese), PhD dissertation, Tokyo Institute of Technology (April 1981).

[GOLO58] S. W. Golomb, B. Gordon, and L. R. Welch, ''Comma-Free Codes,'' *Canadian J. Math.*, 10 (1958): 202–204.

[HSIA70] M. Y. Hsiao, ''A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes,'' *IBM J. Res. Dev.*, 14 (July 1970): 395–401.

[PETE72] W. W. Peterson and E. J. Weldon Jr., *Error Correcting Codes*, 2d ed., MIT Press (1972).

[RAO74] R. N. Rao, *Error Coding for Arithmetic Processors*, Academic Press (1974).

[SHED78] J. J. Shedletsky, ''Error Correction by Alternate-Data Retry,'' *IEEE Trans. Comput*., C-27 (February 1978): 106–112.

[SUND78] C.-E. W. Sundberg, ''Erasure and Error Decoding for Semiconductor Memories,'' *IEEE Trans. Comput*., C-27 (August 1978): 696–705.

[SUND79] C.-E. W. Sundberg, ''Properties of Shortened Codes for Memories with Stuck-at Faults,'' *IEEE Trans. Comput*., C-28 (September 1979): 686–690.

[WALK79] W. K. S. Walker, C.-E. W. Sundberg, and C. J. Black, ''A Reliable Spaceborne Memory with a Single Error and Erasure Correction Scheme,'' *IEEE Trans. Comput*., C28 (July 1979): 493–500.

[YAMA70] H. Yamamoto, T. Watanabe, and Y. Urano, ''Alternating Logic and Fault Detection,'' *Proc. 1970 IEEE Int. Comput. Group Conf.* (June 1970): 220–228.

CONTENTS

4

Codes for High-Speed Memories I: Bit Error Control Codes

Error control codes (ECCs) have been successfully applied to computer systems, especially to memory systems. One can say that every memory designer has adopted some types of error detecting or error correcting codes in order to enhance system reliability [HSIA69, TANG69, FUJI82, CHEN84, BOSE86B, FUJI90]. In Chapters 4 through 7 we discuss error control codes for high-speed memories, namely for semiconductor memories such as cache memories, main memories, control memories, and disk cache memories. These memories all employ random access memory (RAM) semiconductor chips. Therefore we also call these applications *codes for semiconductor memories*. Chapter 4 covers bit error correcting code applications, and Chapters 5, 6, and 7 cover some types of byte error correction / detection for recent memory systems with byte organized high-density RAM chips.

One of the notable features of the codes developed for high-speed memories is that *parallel encoding and decoding* is required to maintain high rates of data throughput. Therefore encoding and decoding circuits have to be implemented by combinational logic [HSIA69, FUJI75].

In high-speed memories, single-bit error correcting and double-bit error detecting codes (SEC-DED codes) were commonly used. This is because the original first-generation semiconductor DRAM (dynamic RAM) chips are organized for one bit of data input / output at a time, and therefore any failure in one chip manifested itself as one bit in error. For the purpose of correcting *soft errors* induced by *α-particles*, external noises, and sometimes by neutrons and cosmic rays, some new techniques and some advanced error correcting codes are being required for large-capacity, high-speed memories [HSIA70b, IMAI77b, BOSS80]. This chapter deals with these considerations for codes such as the modified Hamming SEC-DED codes and double-bit error correcting codes (DEC codes). This chapter also presents on-chip error control codes, called *on-chip ECCs*, that are used to solve the problems of soft-errors and chip yield degradation.

Code Design for Dependable Systems: Theory and Practical Applications, by Eiji Fujiwara

4.1 MODIFIED HAMMING SEC-DED CODES

A distance-4 Hamming code [HAMM50] can correct single-bit errors and also detect double-bit errors (SEC-DED). This code can be formed by extending a distance-3 Hamming code with an overall parity check, that is, a check on all the symbols shown in Subsection 2.3.3.

This section shows that this Hamming SEC-DED code can be modified and optimized from the practical point of view. The resulting code is called a *modified Hamming SEC-DED code*. This code can also be used to solve multiple soft error problems.

4.1.1 Odd-Weight-Column Codes — Hsiao Codes —

The minimum distance of an SEC-DED code is 4. Since a nonzero n-tuple of weight 3 or less is not a codeword, any set of three columns of the **H** matrix should be linearly independent. Note that *the sum of two odd-weight r-tuples is an even-weight r-tuple* (*i.e.*, *odd + odd = even, even + odd = odd, even + even = even*). Because of this property, an SEC-DED code with r check bits can be constructed with its **H** matrix consisting of distinct nonzero r-tuples of column vectors having odd weight [HSIA70a]. This code is different from the original Hamming SEC-DED code whose **H** matrix has an all-1 row vector in addition to the SEC code **H** matrix. Therefore this code is called a modified Hamming code or, more specifically, an *odd-weight-column SEC-DED code*, because every **H** matrix column vector is odd weight. The following code design was first proposed by M. Y. Hsiao in 1970, and therefore the code is also called *Hsiao code*.

This code has a possibility to have minimum number of 1's in the **H** matrix, which makes the hardware and the speed of the encoding / decoding circuit optimal. That is, it satisfies the condition of the *minimum-weight & equal-weight-row code* shown in Section 3.1, and hence this code is called optimal from the practical point of view.

With these considerations, the **H** matrix of this code is constructed as follows:

Step 1. *Use all* $\binom{r}{1}$ *weight-1 columns for the r check-bit positions.*

Step 2. *Next, if* $\binom{r}{3} \geq k$*, where k is information-bit length, select k weight-3 columns out of all possible* $\binom{r}{3}$ *combinations. If* $\binom{r}{3} < k$*, all* $\binom{r}{3}$ *columns should be selected.*

Step 3. *For the case of* $\binom{r}{3} < k$*, select the leftover columns first from among all* $\binom{r}{5}$ *weight-5 columns. The process is continued until all k columns (corresponding to information positions) have been specified.*

If the code length $n = k + r$ is exactly equal to

$$\sum_{\substack{i=1 \\ i=odd}}^{r} \binom{r}{i} = 2^{r-1}, \tag{4.1}$$

then each row of the **H** matrix will have the following number of 1's:

$$\frac{1}{r} \sum_{\substack{i=1 \\ i=odd}}^{r} i \cdot \binom{r}{i}. \tag{4.2}$$

TABLE 4.1 Sample Examples on the Code Parameter Relations

			Structure of **H**					
n	k	r	$\binom{r}{1}$	$\binom{r}{3}$	$\binom{r}{5}$	Total number of 1's in **H**	Average number of 1's in each row in **H**	XOR gate levels I_s
8	4	4	$\binom{4}{1}$	$+\ \binom{4}{3}$		16	4	$\lceil \log_2 4 \rceil^b = 2$
22	16	6	$\binom{6}{1}$	$+\ 16/\binom{6}{3}^a$		54	9	$\lceil \log_2 9 \rceil = 4$
30	24	6	$\binom{6}{1}$	$+\ \binom{6}{3}$	$+\ 4/\binom{6}{5}$	86	14.3	$\lceil \log_2 15 \rceil = 4$
39	32	7	$\binom{7}{1}$	$+\ 32/\binom{7}{3}$		103	14.7	$\lceil \log_2 15 \rceil = 4$
55	48	7	$\binom{7}{1}$	$+\ \binom{7}{3}$	$+\ 13/\binom{7}{5}$	177	25.3	$\lceil \log_2 26 \rceil = 5$
72	64	8	$\binom{8}{1}$	$+\ \binom{8}{3}$	$+\ 8/\binom{8}{5}$	216	27	$\lceil \log_2 27 \rceil = 5$
137	128	9	$\binom{9}{1}$	$+\ \binom{9}{3}$	$+\ 44/\binom{9}{5}$	481	53.4	$\lceil \log_2 54 \rceil = 6$

Source: *[HSIA70a]. Copyright 1970 by International Business Machines Corporation; republished by permission.*

[a] The notation $j/\binom{r}{i}$ means that j out of all possible $\binom{r}{i}$ combinations is used.

[b] $\lceil x \rceil$ is the smallest integer greater than or equal to x, called the *ceiling* of x. $\lfloor x \rfloor$ is the largest integer less than or equal to x, called the *floor* of x.

If n does not exactly satisfy Eq. (4.1), then a proper selection of the $\binom{r}{i}$ cases should make the number of 1's in each row close to the average number, as shown in Table 4.1. In this table, I_s represents the number of *gate levels* required to generate syndrome S, when only 2-input modulo-2 adders (exclusive-OR gates, or XOR gates) are used. Here the gate level is defined as the time required for the signal to pass through one gate.

An algorithm for correcting single errors and detecting multiple errors by using the syndrome $S = (s_0, s_1, \ldots, s_{r-1})$ is as follows:

Step 1. *Test whether S is* $\mathbf{0}$*. If S is* $\mathbf{0}$*, the word can be assumed to be error free.*

Step 2. *If* $S \neq \mathbf{0}$*, try to find a perfect match between S and a column of the* $\mathbf{H}$ *matrix. The match can be implemented in r-input AND gates.*

Step 3. *If S is the same as the i-th column of* $\mathbf{H}$ *from step 2, the i-th bit of the word is in error and hence inverted.*

Step 4. *If* $S \neq \mathbf{0}$ *and overall parity (i.e., the exclusive-OR sum) of all syndrome bits is equal to zero, a double error (or even number of errors) is detected. This condition is expressed in logical form as*

$$\left(\bigcup_{i=0}^{r-1} s_i\right) \cdot \left(\overline{\sum_{i=0}^{r-1}{}^{\oplus}\, s_i}\right) = 1, \tag{4.3}$$

$\bigcup$: *OR operation*, $\sum^{\oplus}$: *mod 2 sum*, $\bar{x}$: *complement value of* x.

Step 5. *If a nonzero S is equal to none of the columns of* $\mathbf{H}$*, and the overall parity of all syndrome bits is equal to 1, then three or more odd number of errors are detected.*

A simple example of the odd-weight-column SEC-DED code and its decoding circuit is illustrated in Example 4.1.

Example 4.1

Memory readout word: $D = [d_0\ d_1\ d_2\ d_3\ c_0\ c_1\ c_2\ c_3]$.

$$\mathbf{H} = \begin{array}{c} \begin{array}{cccc} \underline{\text{Information bits}} & & & \end{array} \quad \begin{array}{cccc} \underline{\text{Check bits}} & & & \end{array} \\ \begin{array}{cccc|cccc} d_0 & d_1 & d_2 & d_3 & c_0 & c_1 & c_2 & c_3 \end{array} \\ \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \end{array}.$$

Syndrome: $S = D \cdot \mathbf{H}^T = [s_0\ s_1\ s_2\ s_3]$.

$$\begin{aligned} s_0 &= d_0 \oplus d_1 \oplus d_2 \oplus c_0 \\ s_1 &= d_0 \oplus d_1 \oplus d_3 \oplus c_1 \qquad \oplus : \textit{modulo} - 2\ \textit{addition}. \\ s_2 &= d_0 \oplus d_2 \oplus d_3 \oplus c_2 \\ s_3 &= d_1 \oplus d_2 \oplus d_3 \oplus c_3 \end{aligned}$$

Parallel decoding circuit is shown in Figure 4.1.

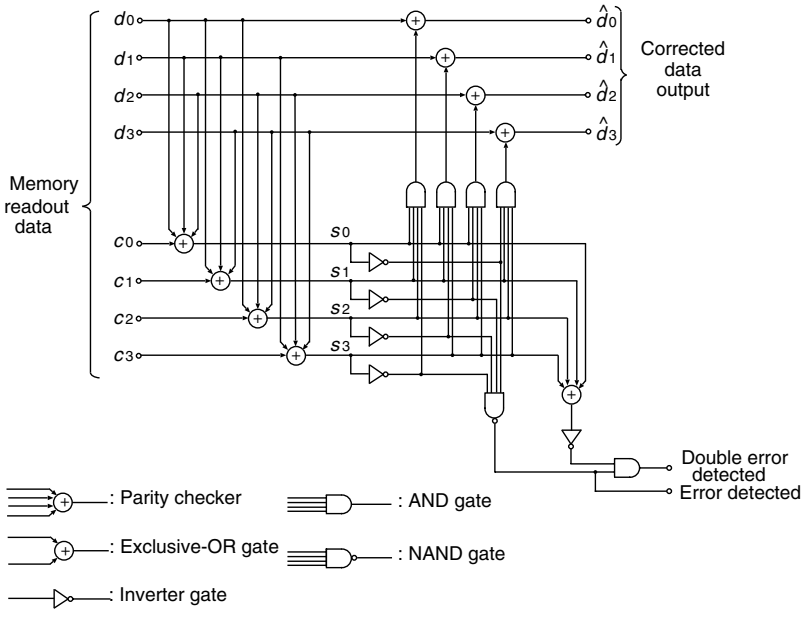


Figure 4.1 *Parallel decoding circuit.*

An example of the **H** matrix for the $(72, 64)$ code is shown in Figure 4.2. From the practical perspective, it is important that the SEC-DED code minimizes the probability of *miscorrection* when triple or more errors occur. A *miscorrection* in this context refers to an erroneous decoding that results in an error-free word or in an incorrect word containing miscorrected single bit. Now, again, we consider the (72, 64) codes. Since these codes are *shortened* SEC-DED *codes* triple errors have a possibility to generate syndrome patterns outside the column patterns of the **H** matrix. In this case the triple errors will be correctly decoded, that is, detected. If the syndrome pattern coincides with a column of **H**, the decoder will mistake it for a single error and performs a miscorrection. Table 4.2 shows the probability of detected triple errors and quadruple errors of the (72, 64) SEC-DED codes. From this table we see that the code shown in Figure 4.2 gives better protection.

In general, the undetectable error probabilities of triple errors and quadruple errors, denoted as P_{UD_3} and P_{UD_4}, respectively, for odd-weight-column SEC-DED codes depend on the number of codewords with minimum weight 4, A_4, as follows [HSIA70a]:

$$P_{\text{UD}_3} = \frac{4 \cdot A_4}{\binom{n}{3}}, \quad P_{\text{UD}_4} = \frac{A_4}{\binom{n}{4}} \tag{4.4}$$

A code design method with a reduced number of weight-4 codewords has been studied by [MATS87]. After the code condition of minimum weight of the **H** matrix is relaxed, a $(72,\ 64)$ SEC-DED code with improved error detection capabilities has been obtained [AZUM75].

As the preceding analyses, odd-weight-column SEC-DED codes have some practical advantages in decoding speed, amount of encoder / decoder hardware, and a lower probability of erroneous decoding. These codes were therefore widely used in the semiconductor memory systems of the 1970s and 1980s [CHEN84]. Parallel error detection and correction circuit ICs based on the Hsiao codes were designed and sold by some semiconductor industries.

4.1.2 Davydov-Tombak Codes

Davydov and Tombak [DAVY91] have designed an excellent SEC-DED code that appears to be more capable of detecting triple and quadruple errors than the conventional SEC-DED codes. This code has neither odd-weight-column vectors nor a minimum weight in **H**.

The **H** matrix of this code has the following form:

$$\mathbf{H} = \begin{bmatrix} \mathbf{B}_0 & \mathbf{B}_1 & \mathbf{B}_2 & \cdots & \mathbf{B}_{D-1} \\ \mathbf{G} & \mathbf{G} & \mathbf{G} & \cdots & \mathbf{G} \end{bmatrix}. \tag{4.5}$$

Here **G** consists of the following 4×5 matrix, and $\mathbf{B}_i = [b_i, b_i, \ldots, b_i]$, $i = 0, 1, \ \ldots, D-1$, where $D = 2^{r-4}$, is an $(r-4) \times 5$ matrix consisting of identical columns b_i, where b_i is

TABLE 4.2 Probability of Error Detection for (72,64) Codes

Codes	Probability of triple-error detection (%); P_3	Probability of quadruple-error detection (%); P_4
Odd-weight-column code shown in Figure 4.2	43.72	99.19
Hamming distance-4 code (non–odd-weight- column code)	24.0 $\sim$ 43.5	98.90 $\sim$ 99.18

Byte	0								1								2								3							
bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
S_0	1	1	1	1	1	1	1	1			1			1	1		1			1				1	1			1				1
S_1	1	1	1						1	1		1	1	1	1	1		1			1	1				1			1			1
S_2				1	1	1			1	1	1						1	1	1	1	1	1	1	1			1			1	1	
S_3				1				1	1			1	1	1			1	1	1						1	1	1	1	1	1	1	1
S_4				1								1			1	1				1	1	1			1	1	1					
S_5	1			1								1								1			1	1				1	1	1		
S_6		1				1		1	1			1								1								1			1	1
S_7			1				1	1		1				1			1	1		1								1				

Byte	4								5								6								7								8: Check Byte							
bit	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	C_0	C_1	C_2	C_3	C_4	C_5	C_6	C_7
S_0				1								1			1	1				1	1	1			1	1			1				1							
S_1	1			1								1								1			1	1				1	1	1		1							
S_2		1			1			1	1			1								1								1			1	1			1				
S_3					1		1			1			1			1	1			1								1							1				
S_4	1	1	1	1	1	1	1	1			1			1	1			1			1			1	1			1								1			
S_5	1	1	1						1	1	1	1	1	1	1	1			1			1	1			1			1			1					1		
S_6				1	1	1			1	1	1						1	1	1	1	1	1	1	1		1			1	1								1	
S_7				1			1	1				1	1	1			1	1	1						1	1	1	1	1	1	1	1								1

Figure 4.2 *(72, 64) Odd-weight-column SEC-DED code.* Source: *[HSIA70a]. © Copyright 1970 by International Business Machines Corporation; republished by permission.*

the binary representation of i:

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

The maximum code length (in bits) of the code is $n = 5 \cdot 2^{r-4}$ for $r \geq 5$.

Example 4.2 (40, 33) SEC-DED code [DAVY91]

$$\mathbf{H} = \left[\begin{array}{cccccccc} 00000 & 00000 & 00000 & 00000 & 11111 & 11111 & 11111 & 11111 \\ 00000 & 00000 & 11111 & 11111 & 00000 & 00000 & 11111 & 11111 \\ 00000 & 11111 & 00000 & 11111 & 00000 & 11111 & 00000 & 11111 \\ \hline 10001 & 10001 & 10001 & 10001 & 10001 & 10001 & 10001 & 10001 \\ 01001 & 01001 & 01001 & 01001 & 01001 & 01001 & 01001 & 01001 \\ 00101 & 00101 & 00101 & 00101 & 00101 & 00101 & 00101 & 00101 \\ 00011 & 00011 & 00011 & 00011 & 00011 & 00011 & 00011 & 00011 \end{array}\right].$$

The shortening algorithm of the code can obtain the excellent code with any code parameters.

Shortening Algorithm The matrix $\mathbf{H}_r$ is shortened by j columns, $j \leq 8$, where the columns of $\mathbf{H}_r$ are deleted in the following order:

$$\left[\frac{b_\gamma}{g_{15}}\right], \left[\frac{b_\gamma}{g_8}\right], \left[\frac{b_\gamma}{g_4}\right], \left[\frac{b_\gamma}{g_2}\right], \left[\frac{b_\gamma}{g_1}\right], \left[\frac{b_\delta}{g_{15}}\right], \left[\frac{b_\sigma}{g_8}\right], \left[\frac{b_\mathscr{H}}{g_4}\right],$$

where g_v is a column of matrix $\mathbf{G}$ corresponding to the binary representation of v, and columns b_γ, b_δ, b_σ, $b_\mathscr{H}$ are distinct.

Let $r = 7, j = 1$, and $\gamma = 7$. Then the parity-check matrix $\mathbf{H}$ of the (39, 32) code is the one with the last column omitted. Take $r = 8, j = 8, \gamma = 15, \delta = 14, \sigma = 13$, and $\mathscr{H} = 12$. The parity-check matrix of the (72, 64) code has the following form:

$$\left[\begin{array}{l} 00000\ 00000\ 00000\ 00000\ 00000\ 00000\ 00000\ 00000\ 11111\ 11111\ 11111\ 11111\ 11\ 11\ 111\ 1\ 1111 \\ 00000\ 00000\ 00000\ 00000\ 11111\ 11111\ 11111\ 11111\ 00000\ 00000\ 00000\ 00000\ 11\ 11\ 111\ 1\ 1111 \\ 00000\ 00000\ 11111\ 11111\ 00000\ 00000\ 11111\ 11111\ 00000\ 00000\ 11111\ 11111\ 00\ 00\ 000\ 0\ 1111 \\ 00000\ 11111\ 00000\ 11111\ 00000\ 11111\ 00000\ 11111\ 00000\ 11111\ 00000\ 11111\ 00\ 00\ 111\ 1\ 0000 \\ \hline 10001\ 10001\ 10001\ 10001\ 10001\ 10001\ 10001\ 10001\ 10001\ 10001\ 10001\ 10001\ 10\ 01\ 100\ 1\ 1000 \\ 01001\ 01001\ 01001\ 01001\ 01001\ 01001\ 01001\ 01001\ 01001\ 01001\ 01001\ 01001\ 01\ 01\ 010\ 1\ 0100 \\ 00101\ 00101\ 00101\ 00101\ 00101\ 00101\ 00101\ 00101\ 00101\ 00101\ 00101\ 00101\ 00\ 01\ 001\ 1\ 0010 \\ 00011\ 00011\ 00011\ 00011\ 00011\ 00011\ 00011\ 00011\ 00011\ 00011\ 00011\ 00011\ 00\ 11\ 000\ 1\ 0001 \end{array}\right]$$

$\uparrow \quad \uparrow \quad \uparrow\uparrow\uparrow\uparrow\uparrow$

$\binom{\mathscr{H}=12}{v=4}\binom{\sigma=13}{v=8}\binom{\delta=14}{v=15}\ \Big|\ \binom{\gamma=15}{v=15,8,4,2,1}$

Note : the shortened place (i.e., deleted columns) from the original matrix is indicated by the upward-pointing arrow (↑).

(4.6)

Although the (72, 64) SEC-DED code is not the minimum-weight & equal-weight-row code defined in Section 3.1, it is the best code so far obtained on error detection capabilities of triple and quadruple errors, $P_3 = 55.37\%$ and $P_4 = 99.35\%$, respectively.

4.1.3 Double-Bit Error Correction Using SEC-DED Codes

High-density memory chips create new reliability problems. Good examples are the soft errors caused by α-particles, and neutrons induced by cosmic rays in high-density RAM chips [NOOR80, SAIH82, OGOR96]. These *soft errors* may line up with existing hard errors, giving rise to multiple errors that are not correctable with SEC-DED codes.

To solve these problems, extended reliability techniques have been proposed for large-capacity memory systems with SEC-DED facilities [KANE84b, AICH84]. Some of them are the *read-retry technique* in which the soft errors disappear during the repeated read cycles, the *sparing technique*, which replaces a defective component with a spare without requiring manual intervention, and the *mask error correction technique*, which requires some additional operations for detection and correction of hard errors. The mask error correction by retry method is illustrated in the following example. This idea is based on the ADR (alternate data retry) mentioned in Subsection 1.3.2.

Example 4.3

The following sequence of operations allows for mask error correction of *hard-plus-soft errors* with using the self-complementing SEC-DED code where *self-complementing code* has been defined by Definition 3.6:

1	0	0	1	1	1	0	1	1	Correct data
		h		α					Faults (h : hard error position, α : soft error position)
		↓		↓					
1	0	**1**	1	**0**	1	0	1	1	Read from memory : Uncorrectable error(_)
0	1	0	0	1	0	1	0	0	Complement, written
0	1	1	0	1	0	1	0	0	Read from memory
1	0	0	1	[**0**]	1	0	1	1	Recomplement : Correctable error([ ])
1	0	0	1	1	1	0	1	1	Correct data rewritten
1	0	[**1**]	1	1	1	0	1	1	Read on refetch : Correctable error([ ])

From the foregoing Read-Invert-Write-Read-Invert procedure we can correct double errors by using only the SEC-DED code. The soft errors will be masked and disappear during the subsequent read operations. In recent one-transistor-type dynamic RAMs, however, the stored data are destroyed in every read operation, and therefore a rewrite operation is always performed. That is, the erroneous datum caused by hard-plus-soft errors is readout from the DRAMs and then this erroneous datum is rewritten, which means the errors are retained in memory even in the read operation. The read-retry operation cannot then recover the erroneous data caused by soft errors in recent DRAMs. Therefore the read-retry operation even in cooperation with the SEC-DED code cannot tolerate the above hard-plus-soft errors.

Another effective method for correcting multiple errors is to apply an error location technique to distance-4 codes (i.e., *erasure correction technique*). With the error

location capability both soft errors and hard errors can be corrected [BOSS80, DEZA82], and the system will concentrate on erasure correction. Error location enables the distance-4 code (SEC-DED code) to correct up to three errors, in accordance with the discussion in Subsection 2.2.4. The following algorithm shows way to correct the case of one hard error and one soft error occurring simultaneously, in effect, hard-plus-soft errors [BOSS80]. In the algorithm the hard error is statically located during the recovery process, and an algorithmic modification of the original syndrome allows the correction of the soft error. In this way, no extra auxiliary storage will be required.

Let h be the hard-error bit position and α be the soft-error bit position. Then the resultant syndrome S_u is the exclusive-OR of the syndromes due to erroneous bits α and h, meaning $S_u = S_\alpha \oplus S_h$. The decoding algorithm is as follows:

Step 1. *Detect the uncorrectable error, represented by S_u, a double-error syndrome. Save the codeword with errors as well as the syndrome.*

Step 2. *Using the exerciser diagnostic patterns, locate position h of the solid (hard) error. Knowing the index h, generate S_h.*

Step 3. *Determine $S_\alpha = S_u \oplus S_h$.*

Step 4. *Decode S_α to correct bit α in the data. Invert bit h (determined in step 2) to correct the hard error.*

This algorithm can be implemented as shown in Figure 4.3, where (72, 64) SEC-DED code is applied. In step 2 of the algorithm three diagnostic test patterns are generally required in order to locate the hard error position in this memory system. However, if an *even-weight-row SEC-DED code* is applied to the system, only two test patterns are satisfactory because a bit-wise complemented codeword of the even-weight-row code is also a codeword (see Section 3.3). Hence only all-0 data and all-1 data, both of which are codewords of this code, can test the memory system. In general, how we get the *error pointers* is important in this technique. In the foregoing method, the pointers are obtained by applying the test patterns to the memories. It is apparent that these error pointers are available only for hard errors.

4.2 MODIFIED DOUBLE-BIT ERROR CORRECTING BCH CODES

System reliability usually decreases as the capacity of a memory system increases. To maintain the same high level of reliability, or to improve reliability, more powerful error control codes such as double-bit error correcting (DEC) code, double-bit error correcting and triple-bit error detecting (DEC-TED) code (both based on the well-known BCH code), or multiple-bit error correcting *majority-logic decodable code* [PETE72] may be applied to large-scale high-reliability memory systems. However, these codes generally make the decoding slower and increase code redundancy.

Because of these problems some new extended codes that are based on the BCH code [PATE72a, IMAI77b, IMAI79] and the new majority-logic decodable codes [HSIA70b, CHEN73, HORI75, MATS77, MATS78] have been studied. Some modifications to the double-bit error correcting BCH codes have been proposed in [PATE72b, FUJI76, IMAI77a, HOWE77, YAMA80, GOLA83, OKAN87].

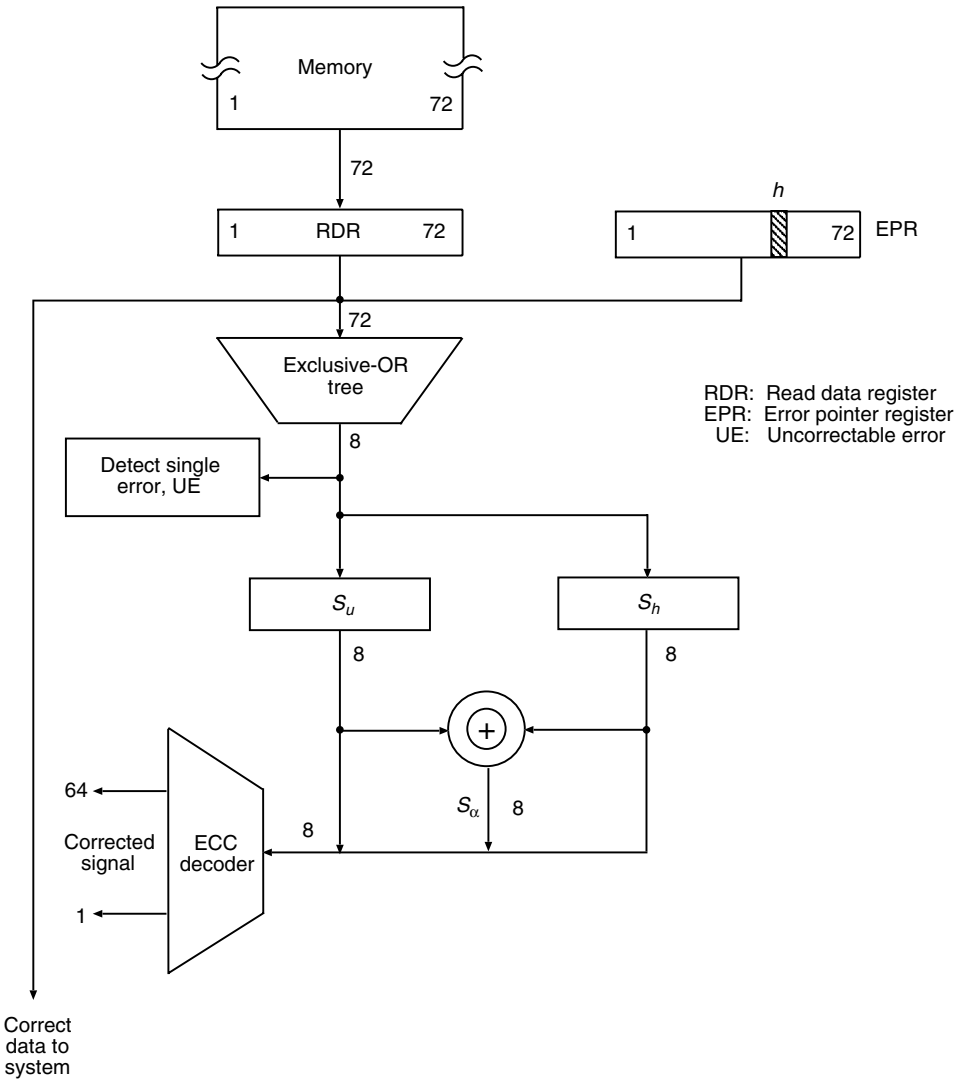


Figure 4.3 *Schematic implementation for correction of an α-particle-induced uncorrectable error. (Note: ⊕ indicates exclusive-OR, or XOR.)* Source: *[BOSS80]. © Copyright 1980 by International Business Machines Corporation; republished by permission.*

Table 4.3 shows the check-bit lengths of the existing codes with double-bit error correction capability. These two types of codes have the following general features:

1. BCH-based codes have a minimum or smaller number of check bits, but have complex decoding hardware and a longer decoding time.
2. Majority-logic decodable codes have a high decoding speed and simple decoding hardware, but have a large number of check bits (i.e., nearly one-half of the information-bit length).

TABLE 4.3 Check-Bit Lengths of DEC Codes

Codes	Information-bit length k (bits)		
	32	64	128
BCH code	12	14	16
Imai-Kamiyanagi code [IMAI77b]	14 ∼ 16	15 ∼ 19	19 ∼ 23
Horiguchi-Morita code[a] [HORI75]	16 ∼ 19	22 ∼ 26	31 ∼ 37
MA code[a] [MATS77]	23	31	40
Orthogonal Latin square code[a] [HSIA70b]	24	32	48

[a]Majority-logic decodable code.

In the next subsection we will study the extended double-bit error correcting code based on the BCH (DEC-BCH) code, and then study new parallel algebraic decoding method of the DEC-BCH code.

4.2.1 BCH-Based Codes

The basic structure of the double-bit error correcting (DEC) BCH code is as follows: Let α be a primitive element in GF(2^m) and α^i be a coefficient binary column vector corresponding to $x^i \bmod \mathbf{g}(x)$, where $\mathbf{g}(x)$ is a primitive polynomial having degree m. From these elements the following matrix can be obtained:

$$\mathbf{H}_1 = [\alpha^0 \ \alpha^1 \ \ldots \ \alpha^{n-1}],$$
$$\mathbf{H}_3 = [\alpha^0 \ \alpha^3 \ \ldots \ \alpha^{3(n-1)}].$$

In this case $n = 2^m - 1$. The DEC-BCH codes can be expressed as

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_3 \end{bmatrix}_{2m \times n}. \tag{4.7}$$

The binary BCH codes have been mentioned previously in Subsection 2.3.5.

Imai-Kamiyanagi Code Imai and Kamiyanagi [IMAI77b] give an extended structure of the BCH codes such that some all-0 row vectors and all-1 row vectors are added to the original structure of the BCH codes. The resultant code has the following characteristics:

1. The decoding hardware has nearly the same complexity as that of the majority-logic decodable code, but is much simpler to design than the original DEC-BCH code.
2. The decoding can be made faster than by the DEC-BCH code.
3. The check-bit length is only slightly larger than that of the DEC-BCH code (see Table 4.3).

The following notations are defined here:

1_n: all-1 n-tuple row vector,
0_n: all-0 n-tuple row vector,

$\mathbf{I}_n$: $n \times n$ identity matrix,
$\mathbf{0}_{k\times l}$: $k \times l$ zero matrix.

The following two matrices $\mathbf{A}$ and $\mathbf{B}$ have an important role in this code construction:

A: $r_0 \times n_0$ matrix over $GF(2)$. The rightmost r_0 column vectors are linearly independent. The i-th (for $i = 1, 2, \ldots, n_0$) column vector is denoted as $\mathbf{a}_i$.

B: $r_1 \times n_0$ matrix over $GF(2)$. This has r_1 linearly independent column vectors. The i-th column vector is denoted as $\mathbf{b}_i$.

From these matrices the following three matrices can be constructed:

$$\mathbf{H}'_1 = [\mathbf{A} \otimes \mathbf{H}_1 \quad \mathbf{0}_{mr_0\times r_1}],$$
$$\mathbf{H}'_2 = [\mathbf{B} \otimes 1_{n_1} \quad \mathbf{I}_{r_1}],$$
$$\mathbf{H}'_3 = [1_{n_0} \otimes \mathbf{H}_3 \quad \mathbf{0}_{m\times r_1}].$$

Here $\otimes$ shows the *Kronecker product* (see Example 4.4). Then

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}'_1 \\ \mathbf{H}'_2 \\ \mathbf{H}'_3 \end{bmatrix}$$

is the $\mathbf{H}$ matrix of this new code. The codeword in this code can be expressed as $(n_0 + 1)$ blocks (i.e., n_0 blocks each having length n_1 and one block with length r_1). The code parameter of this code is as follows:

Code length: $n = n_0 n_1 + r_1$,
Information-bit length: $k = n_0 n_1 - (r_0 + 1)m$,
Check-bit length: $n - k = (r_0 + 1)m + r_1$.

Example 4.4

Let

$$n_0 = 3,\ \mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{B} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

In this case $r_0 = 2$, and $r_1 = 2$. Hence

$$\mathbf{H}'_1 = \begin{bmatrix} \mathbf{H}_1 & \mathbf{H}_1 & \mathbf{0}_{m\times n_1} & \mathbf{0}_{m\times 2} \\ \mathbf{H}_1 & \mathbf{0}_{m\times n_1} & \mathbf{H}_1 & \mathbf{0}_{m\times 2} \end{bmatrix},$$
$$\mathbf{H}'_2 = \begin{bmatrix} 0_{n_1} & 1_{n_1} & 0_{n_1} & 10 \\ 0_{n_1} & 0_{n_1} & 1_{n_1} & 01 \end{bmatrix},$$

and $\mathbf{H}_3' = [\,\mathbf{H}_3 \quad \mathbf{H}_3 \quad \mathbf{H}_3 \quad \mathbf{0}_{m\times 2}\,]$. Then

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_1 & \mathbf{H}_1 & \mathbf{0}_{m\times n_1} & \mathbf{0}_{m\times 2} \\ \mathbf{H}_1 & \mathbf{0}_{m\times n_1} & \mathbf{H}_1 & \mathbf{0}_{m\times 2} \\ 0_{n_1} & 1_{n_1} & 0_{n_1} & 10 \\ 0_{n_1} & 0_{n_1} & 1_{n_1} & 01 \\ \mathbf{H}_3 & \mathbf{H}_3 & \mathbf{H}_3 & \mathbf{0}_{m\times 2} \end{bmatrix}.$$

If m = 5, we can determine $n_1 = 31$, and therefore the (95, 78) linear DEC code can be obtained.

The principle of correcting double-bit errors is such that a double-bit error in one block can be corrected by the matrix $\begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_3 \end{bmatrix}$ (i.e., DEC-BCH code shown in Eq. (4.7)) included in each block, and errors over two blocks can be corrected by the structures of both matrices **A** and **B**. We leave it to the reader to verify precisely that the code corrects random double-bit errors.

4.2.2 Algebraic Parallel Decoding DEC-BCH Codes

We turn now to an algebraic parallel decoding method for the DEC-BCH code [PATE72b, HOWE77, TAKE77, GOLA83]. From Eq. (4.7) the DEC-BCH code can be written in the form

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_1 \\ \mathbf{H}_3 \end{bmatrix} = \begin{bmatrix} h_{1,0} & h_{1,1} & \dots & h_{1,n-1} \\ h_{3,0} & h_{3,1} & \dots & h_{3,n-1} \end{bmatrix},$$

where $h_{1,i}$, and $h_{3,i}$, for $i \in \{0, 1, \dots, n-1\}$ are elements of $GF(2^m)$ and are expressed as column vectors (m-tuples) over $GF(2)$. The syndromes S_1 and S_3 are defined by

$$S_1 = \mathrm{D} \cdot \mathbf{H}_1^T, \qquad S_3 = \mathrm{D} \cdot \mathbf{H}_3^T.$$

Here D is a received word from the memory that is to be decoded. In the decoder the locator x of the single-bit and the double-bit errors can be found as a solution of the following equation (see Exercise 4.16):

$$\left(\frac{x}{S_1}\right)^2 \oplus \frac{x}{S_1} = 1 \oplus \frac{S_3}{S_1^3}. \tag{4.8}$$

There exist two types of circuits to implement the parallel decoder: the first type employs only combinational circuits such as square, invert, and multiply circuits [IMAI77a], and the second type employs ROMs, which store the table that can obtain the solution x from the input $(1 \oplus S_3/{S_1}^3)$ in Eq. (4.8) [HOWE77, YAMA80, OKAN87].

Some other decoding methods and their circuits [TAKE77, GOLA83] are interesting as well. One is based on finding all nonzero solutions of the key equation

$$f(S_1 \oplus X) = S_3 \oplus f(X), \tag{4.9}$$

where $X \in \{h_{1,0}, h_{1,1}, \ldots, h_{1,n-1}\}$ and the function f is defined by

$$f(y) = \begin{cases} h_{3,i} & \text{if } y = h_{1,i} \in \mathbf{H}_1, 0 \le i \le n-1, \\ 0 & \text{if } y = 0. \end{cases}$$

This guarantees that for a nonzero syndrome the column vector $X \in \{h_{1,i}\}$ can be a solution of the key equation (4.9) only if it is a locator of single-bit or double-bit errors. For the DEC-BCH code, it is clear that the function f is determined as

$$f(y) = y^3.$$

Although Eq. (4.9) is equivalent to Eq. (4.8) in the DEC-BCH code, the correction mechanism gives a different decoder construction (i.e., bit-sliced decoder). The correction algorithm is as follows:

Step 1. *Find syndromes S_1 and S_3.*

Step 2. *If $S_1 = S_3 = 0$, meaning no correction, then end.*

Step 3. *Add S_1 to all $h_{1,i}$'s and S_3 to all $h_{3,i}$'s to decompose the syndromes S_1 and S_3:*

$$\begin{aligned} S_{1,i} &= S_1 \oplus h_{1,i}, \\ S_{3,i} &= S_3 \oplus h_{3,i}, \end{aligned} \qquad 0 \le i \le n-1. \tag{4.10}$$

Step 4. *Find the values of $f(S_{1,i})$ for $i = 0, 1, \ldots, n-1$, by Eq. (4.10).*

Step 5. *Find if there are any i's for which the key equation in the form*

$$f(S_{1,i}) = S_{3,i} \tag{4.11}$$

is satisfied; if it does for any i, then invert the i-th bit of the word.

Step 6. *If there is no i satisfying Eq. (4.11), then an uncorrectable error is signaled. End.*

The decoder is bit-sliced as can be seen in Figure 4.4.

4.3 ON-CHIP ECCs

Several important problems have to be overcome in today's high-packing-density and large-capacity RAM or ROM chips. The soft errors induced by α-particles [MAY79, SAIH82] and yield degradation caused by increased defects in enlarged chips [WOOD86] concern system designers most. An on-chip error control code, called *on-chip ECC*, appears to be a good way to prevent such problems [CLIF74, CLIF80, NOOR80, MANO83, GHAF84, DAVI85, FUJA85, HAN87, YAMA88]. The on-chip ECC is expected to bring about highly reliable and cost-effective memory chips with ultra large capacity, but the problem is the large amount of circuitry due to the additional error correcting and detecting circuit and additional memory cells for the check bits. The circuitry increases call for larger chips that therefore yield more defects and α-particle

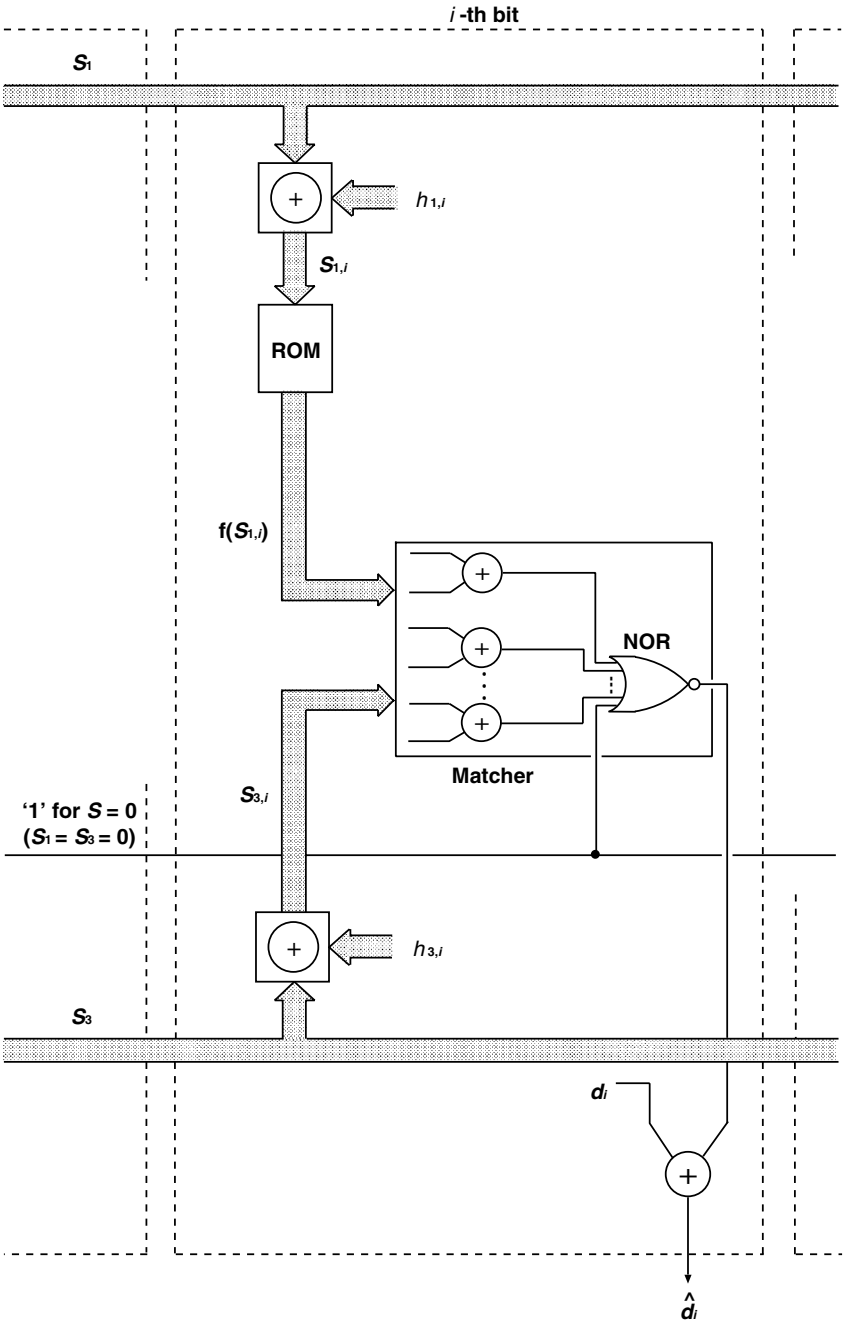


Figure 4.4 *Bit-sliced DEC-BCH decoder.*

bombardment. So, it is important to choose the most suitable type of code for the on-chip ECC.

For soft error problems not only DRAMs but also SRAMs (static RAMs) can use on-chip ECCs. High-speed cache memories implemented by SRAMs have been mounted

in recent microprocessor chips, and they can take such types of bit / byte error control codes as the SEC-DED codes and the SEC-DED-S*b*ED codes discussed in Subsection 6.1.3. *Microprocessor on-chip ECCs* will be discussed in Subsection 12.4.2.

4.3.1 Two-Dimensional Cross-Parity Codes for Soft Error Problems

The on-chip ECC has proved to be essential for the protection of data from soft errors in ultra–large-capacity RAM chips. A block diagram of a RAM chip with an ECC circuit is shown in Figure 4.5.

Note that each word line in the organization is connected to regular memory cells that contain the information bits and also to extra memory cells for check bits. Bit lines connected to the sense circuits transmit the information bits and the check bits to the error decoding circuit. If a bit error is detected, the corresponding output data of the multiplexer is corrected through an exclusive-OR circuit. The error detection and correction operation is performed during *read / write memory cycles*. To prevent bit-error accumulation, this operation should be carried out during the *refresh memory cycles*.

Soft errors can remain, even with on-chip ECCs, depending on the capability of the code used in the RAM chip. Among the different types of codes we focus here on the single-bit error correcting code because of its decoder simplicity and improved reliability.

The occurrence rate of soft errors, called the soft error rate, in a conventional DRAM chip without ECC is determined by the probability of a single α-particle hitting a single memory cell [SAIH82]. On the other hand, the error rate in a chip with single-bit error

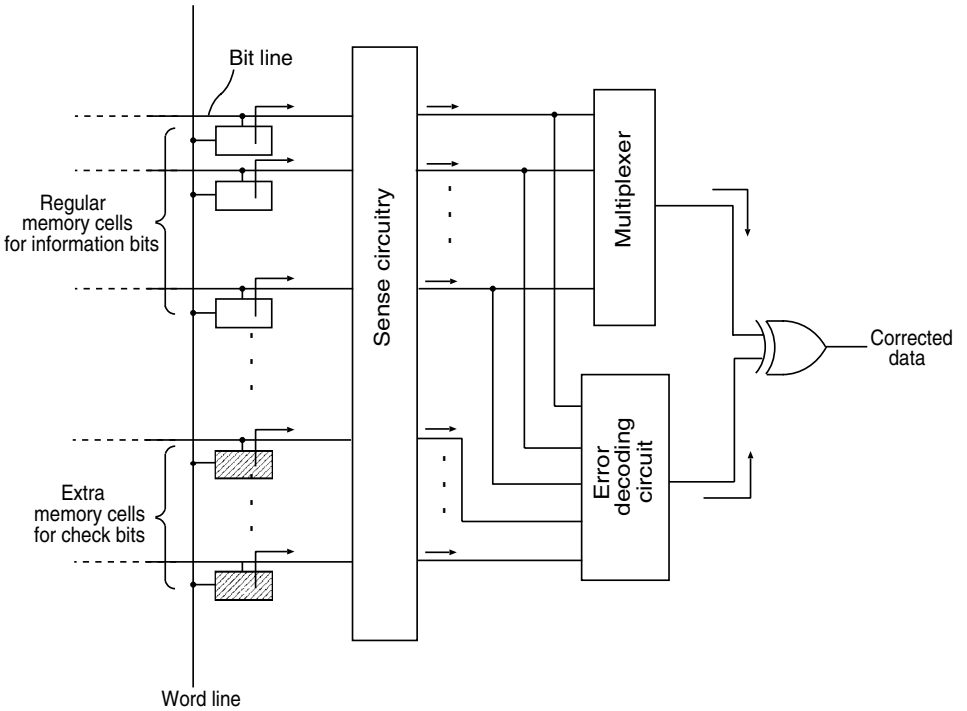


Figure 4.5 *Block diagram of the on-chip ECC RAM.* Source: *[MANO83]. © 1983 IEEE.*

correction capability depends on the probability of the data stored in two or more memory cells being damaged by the α-particle within an error correction period t_0.

Assume that the incidence of the α-particle's strike obeys a Poisson distribution and that the impact of single strike always generates a single-bit error. Then the soft error rate of a RAM chip with ECC is calculated as follows [MANO83]:

$$R_{ECC} = MN^2S_0 - (N/t_0)\{\ln(1 + MNS_0t_0)\},$$

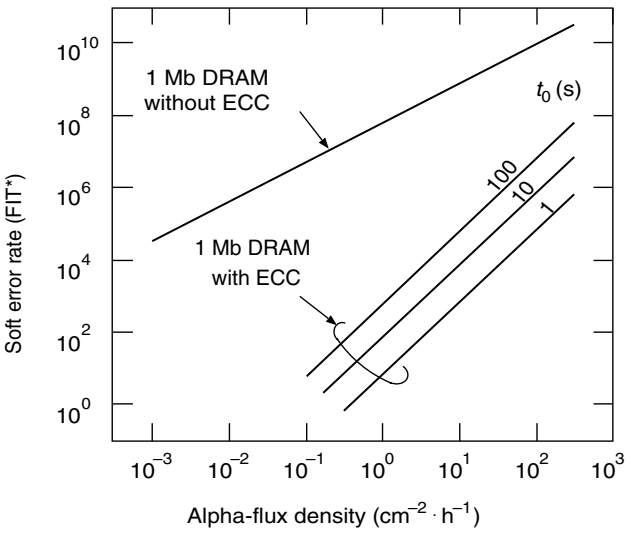
where M is the α-flux density, N^2 is the number of memory cells in a chip, and S_0 is the effective memory cell area. However, the error rate of a RAM chip without ECC is

$$R_0 = MN^2S_0.$$

The soft error rate of the 1 mega-bit (1 Mb) RAM chip with ECC is shown in Figure 4.6 and compared with the error rate without ECC. As the figure shows, the smaller the α-flux density, the greater is the improvement. The improvement factor is more than 10^6 at an α-flux density of $1\,\text{cm}^{-2}\,\text{h}^{-1}$. Clearly, high α-particle immunity is achieved by the on-chip single-bit error correcting code.

To implement an on-chip ECC circuit, it is necessary to reduce the area of the logic circuit and the decoding delay. As was mentioned before, the circuit for the on-chip ECC requires extra memory cells and decoding circuit.

Among the codes with a single-bit error correction capability, we consider here a two-dimensional distance-4 cross-parity code and a distance-3 Hamming code. In a two-dimensional cross-parity code, the horizontal and vertical parity bits are appended to the



*FIT = Number of faults occurring over 10^9 hours per DRAM chip
t_0 : Error correction period(s)

Figure 4.6 *Soft error rate characteristic of the DRAM chip with an on-chip single-bit error correcting code. (Effective storage area in a single transistor cell is assumed to be* $10\,\mu m^2$*.)* Source: *[MANO83].* © *IEEE.*

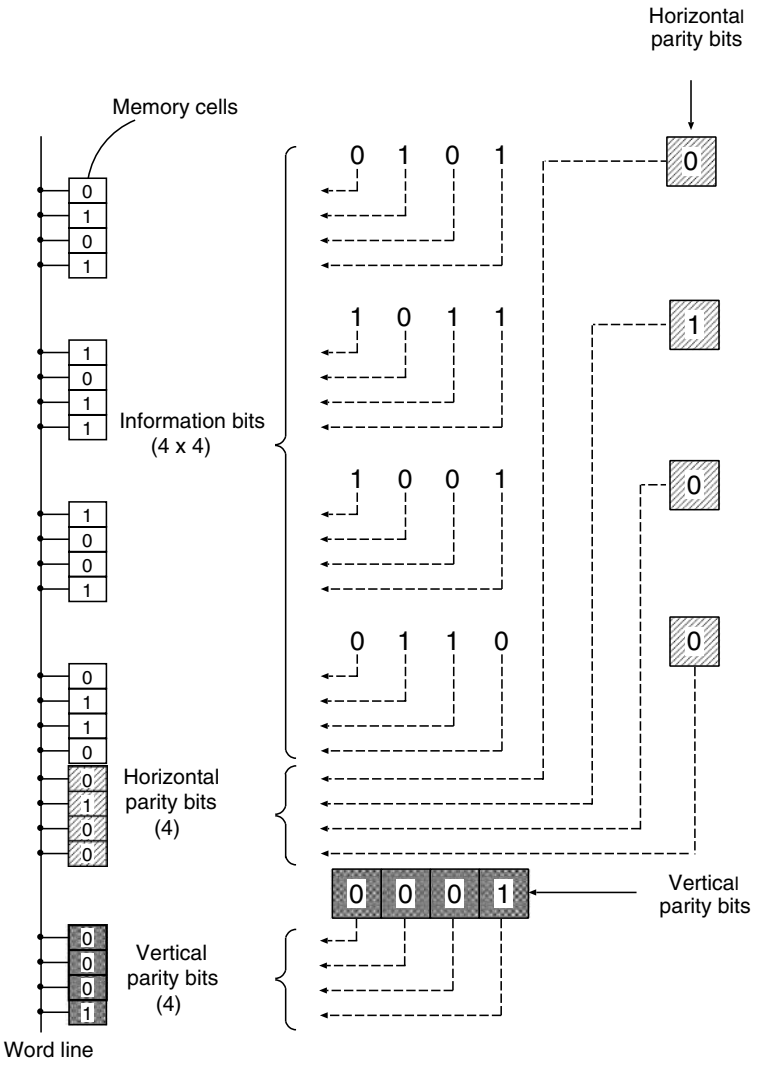


Figure 4.7 *Organization of a cross-parity codeword and its corresponding data in memory cells connected to a specified word line.* Source: *[MANO83]. © 1983 IEEE.*

information bits organized in a two-dimensional array. This is shown in Figure 4.7. In this case, if a single-bit error occurs in the information bits, the error can be simply corrected by horizontal and vertical parity checks. In Figure 4.7 the two-dimensional information bits and the horizontal and vertical parity bits correspond to the stored data in a set of memory cells connected to a specified word line. The readout word from the memory is entered to the decoding circuit, and then vertical and horizontal parity checks are performed. Each information bit is checked by twice independently, by both a vertical and a horizontal check, therefore called a *cross-parity check*. For example, the third information bit in the second row belongs to the second parity check in the horizontal group (**H** group) and also to the third parity check in the vertical group (**V** group). This

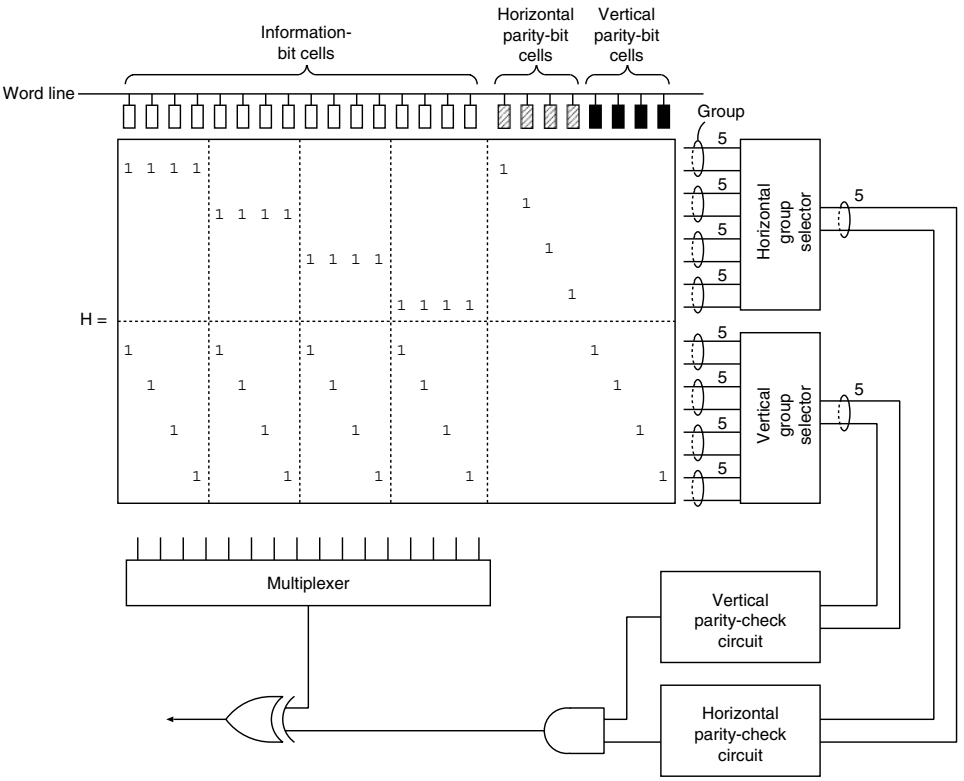


Figure 4.8 *On-chip error decoding circuit for a cross-parity code.* Source: *[YAMA84b]. © 1984 IEICE Japan.*

information bit is therefore corrected only when both syndromes of the cross-parity check are 1's. The decoding circuit of this code is very simple, as shown in Figure 4.8. Figure 4.9 shows a logic diagram of a RAM with a two-dimensional cross-parity code.

In addition to the $l \times m$ regular memory cells, the $l + m$ parity cells are connected to a word line. Each regular memory cell belongs to two groups, that is, the **V** group and the **H** group. During the decoding two kinds of parity checks are carried out by m-bit data in the **V**-group data, l-bit data in the **H**-group, and the two vertical and horizontal parity bits. Also in this decoding the vertical group selector and the horizontal group selector (i.e., multiplexers) are required for selecting the parity-check groups. Hence the decoding circuit shown in Figure 4.8 and in Figure 4.9 has a simple structure, and therefore provides high-speed decoding and small area overhead.

On the other hand, if we use a Hamming SEC code, the number of extra memory cells for check bits can be minimized, but the decoding circuit becomes more complex than the two-dimensional cross-parity code. An example of this decoding circuit is shown in Figure 4.10, where the code has the same information-bit length as the cross-parity code.

From the discussion above it should be clear that the on-chip ECC technique using a two-dimensional cross-parity code is a better way to reduce the soft error rate. Since single-bit errors in each word line can be corrected by this ECC technique, the soft error rate in a high-density RAM chip can be drastically reduced.

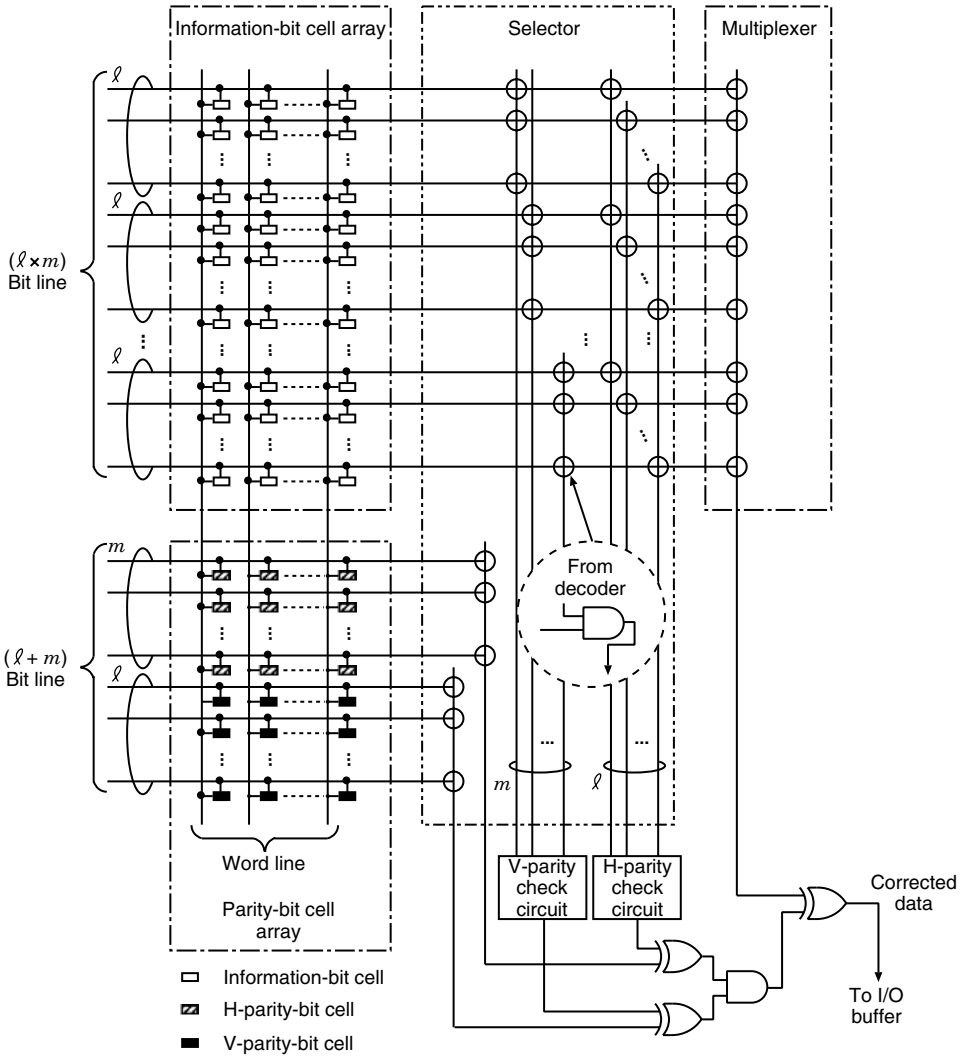


Figure 4.9 *Logic diagram of on-chip ECC RAM using cross-parity code.* Source: *[MANO83]. © 1983 IEEE.*

Figure 4.11 gives an example of an efficient cell alignment for the cross-parity codes [YAMA87b]. The adjacent cell failures on a word line can be detected in an efficient way.

The on-chip ECC technique using a two-dimensional cross-parity code was first practically applied to a 256K-bit DRAM design in 1983 [MANO83]. In addition to the 256K information-bit cells, the DRAM requires 24K parity-bit cells for two-dimensional parity codes. In this DRAM, 512 ($= 16 \times 32$) information-bit cells and 48 ($= 16 + 32$) parity-bit cells are connected to each word line.

The indicated on-chip ECC technique is extended to correct 4-bit byte errors by performing double-precision horizontal and vertical parity checks [YAMA84a], shown in Figure 4.12. This technique has been practically applied to a 1 M-bit DRAM chip in which the ECC circuit occupies about 12% of 52.5 mm^2 ($= 6.4\ mm \times 8.2\ mm$) entire chip area, and the error correction time is 20 ns of 140 ns chip access time. A further improved

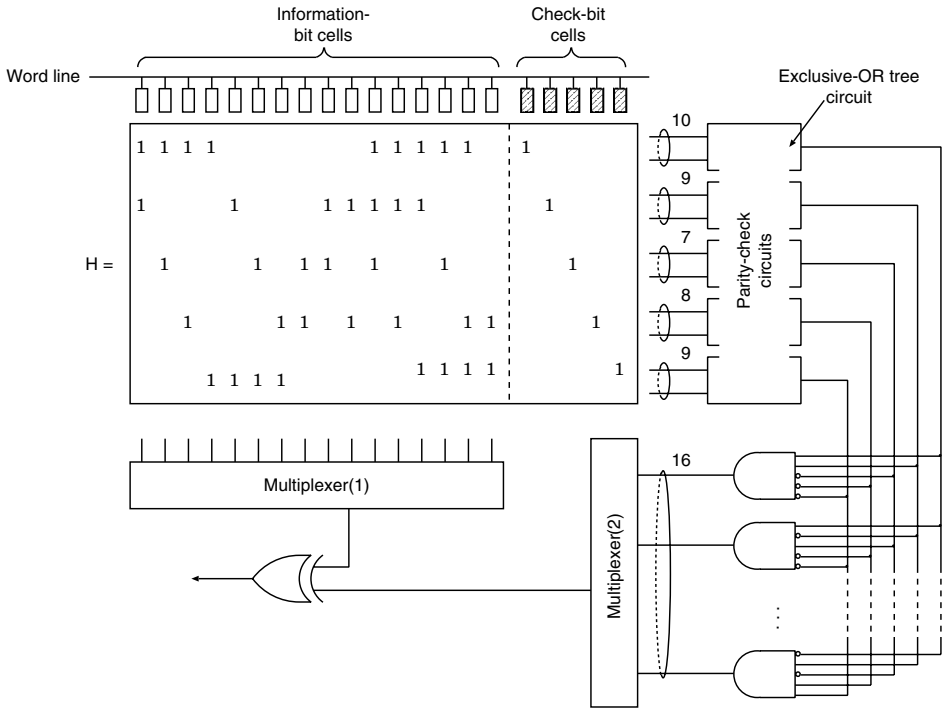


Figure 4.10 *On-chip error decoding circuit for the Hamming SEC code.* Source: *[YAMA84b]. © 1984 IEICE Japan.*

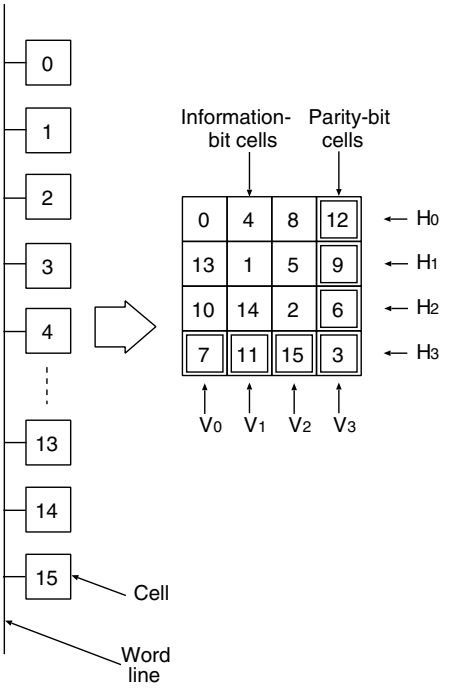


Figure 4.11 *Example of an efficient logical cell alignment for cross-parity codes.* Source: *[MANO87]. © 1987 IEEE.*

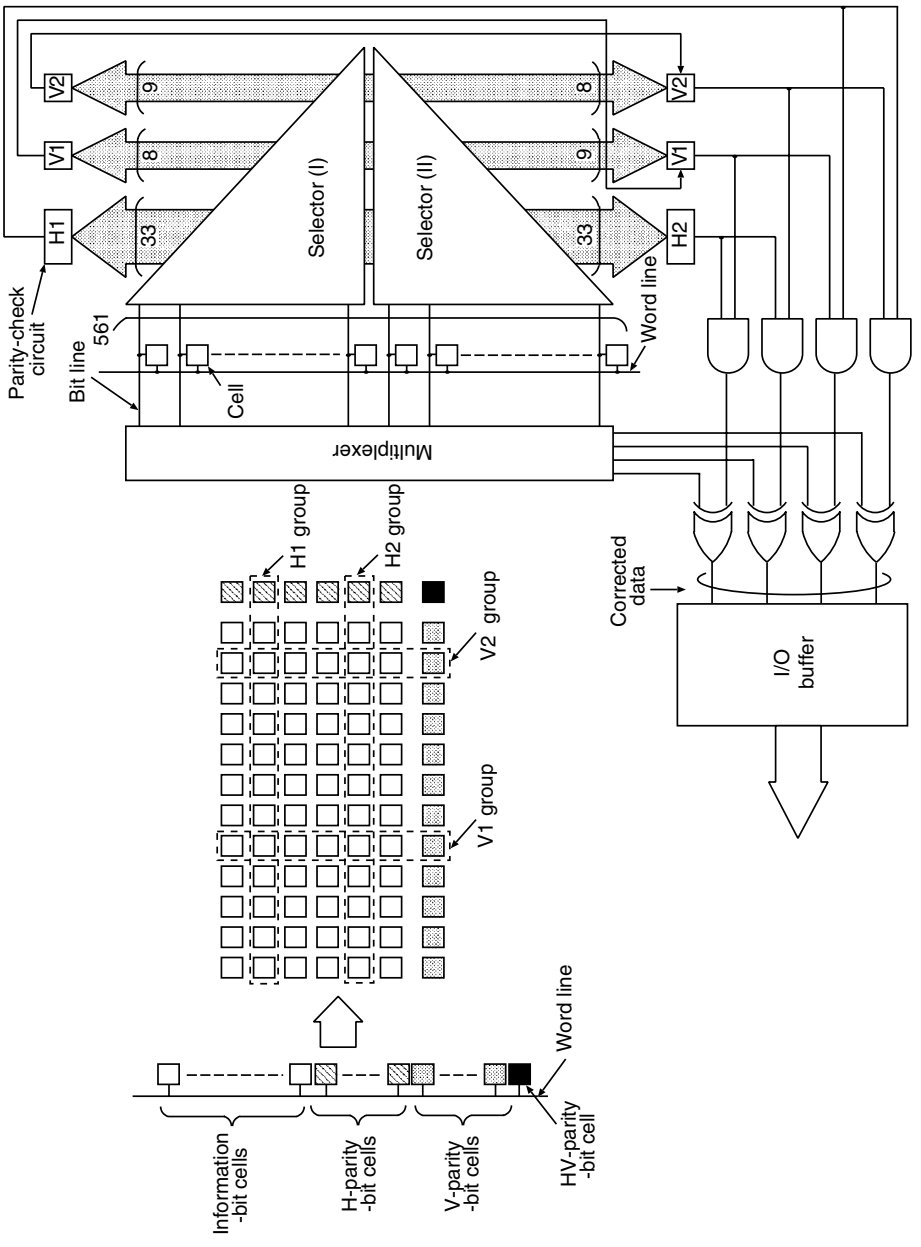


Figure 4.12 *Schematic logic diagram of the on-chip ECC circuit (4-bit byte error correction circuit).* Source: *[YAMA84a]. © 1984 IEEE.*

design method was employed to a 16 M-bit DRAM [MANO87, YAMA87b] and gave a below 5 ns penalty in an access time of 80 ns and occupied about 10% of the $147.7\,mm^2 (= 8.9\,mm \times 16.6\,mm)$ entire chip area.

4.3.2 On-Chip Hamming SEC Codes for Yield Improvement

As the VLSI chip has advanced to very high density, the chip size has become larger. *Wafer scale integration (WSI)* is said to be the final goal of VLSIs [KITA80, LEIG82, PELT83, MANG84a, MANG84b, CARL86, YAMA87a]. Such integration would encompass all possible processor functions. Nowadays *system-on-chip (SoC)* fabrication is being widely employed in microprocessor chips. This type of chip contains processors, some types of memories such as cache memories, register arrays, and address translation arrays (TLB), and peripheral control circuits, all mounted together on one large chip. So it is not surprising that the large LSI chip contains process-induced defects, such as pinholes in the oxide, or missing or extra patterns in the diffusion, polysilicon, or metal [STAP80, STAP83]. Some important *defect-tolerant techniques*, such as *N-modular redundancy* (*NMR*) [KITA80, UEOK84], *m-out-of-n modular redundancy* [SMIT81, MANO82, MOOR86], *address reallocation techniques* [BEAU73, EGAW80], and so forth, have been proposed and studied. Use of error correcting codes also promises to improve yield [CLIF74, MATS88].

Yield can be expressed as a function of circuit area, a (i.e., $y(a)$), assuming that the defects have occurred randomly, and hence the distribution obeys a Poisson distribution.*

$$y(a) = y_0^{a/a_0}.$$

Here y_0 is the yield for specific area a_0.

Figure 4.13 shows the memory model with the redundancy. In this model, memory is divided into $l \times n$ units, each having one-bit input / output data and equal memory capacity, meaning equal area a_m. We can assume that the error control circuit of the SEC code requires a circuit area a_t. Under the Poisson distribution of defects, the total yield of this memory module [OHNI81] is expressed as

$$Y = y(a_t)\{n(y(a_m))^{n-1}(1 - y(a_c)) + (P_0 + P_x + P_y + P_{xy})(y(a_c))^n\}^l,$$

where

a_c: area of the circuit, which is commonly used in a unit such as an address buffer and the input-output data control circuit,

a_m: area of the unit,

a_t: area of the ECC circuit,

P_0, P_x, P_y, and P_{xy}: probabilities of tolerable defects in every memory cell array for
the case where no defect exists in both X and Y decoders (P_0),
the case where defects exist in X decoder (P_x),
the case where defects exist in Y decoder (P_y), and
the case where defects exist in both X and Y decoders (P_{xy}).

*Negative binomial statistics are thought to be better suited for analyzing the defect distribution in large chips or wafers [STAP83, 84, 85, 86].

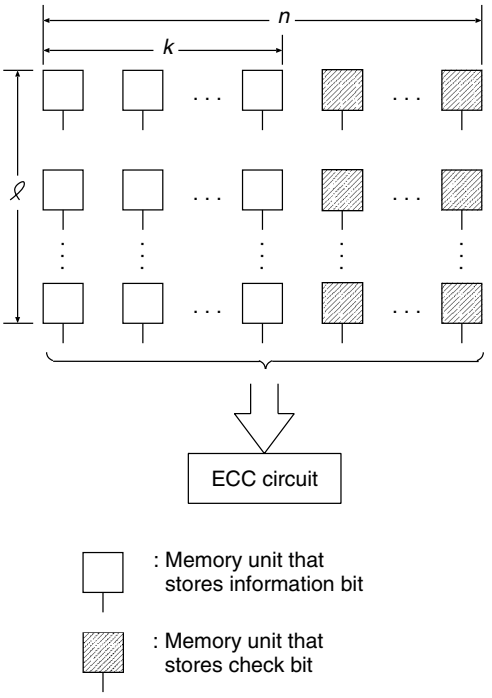


Figure 4.13 *Memory model using an* (n, k) *code.*

In the redundant memory model careful attention must be given to the percentage of the area increase due to the ECC circuit. If the ECC area of the circuit is proportionally large, the yield improvement will be negatively affected. Defects in this area then cannot be tolerated at all.

An example is the application of a Hamming SEC code to the 1 Mb ROM chip [SHIN83, DAVI85]. This memory chip has a 128K-word $\times$ 8-bit organization [SHIN83]. Figure 4.14 shows an ECC circuit of the on-chip (38, 32) Hamming SEC code. The total 38-bit word, consisting of 32 information bits and 6 check bits, is simultaneously readout

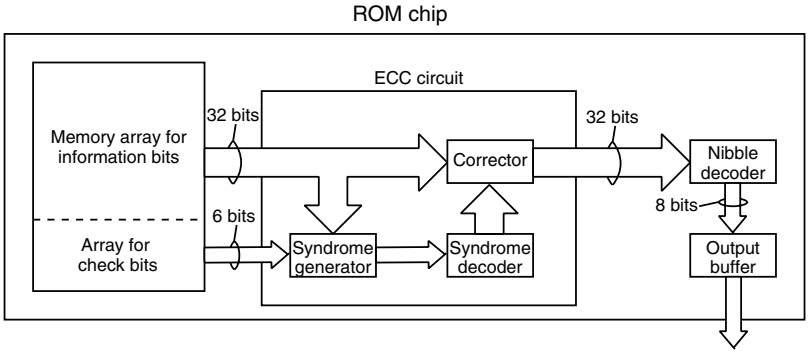


Figure 4.14 *On-chip ECC for the ROM chip.*

from the ROM and entered to the ECC circuit. The corrected 32 information bits are transferred to a nibble decoder and divided into four blocks. The output buffer drives 8 bits. According to [SHIN83], the ECC circuit consumes less than 20% of the total chip area. Even though it increases the chip area, the theoretical yield can be enhanced by a factor of 3 at about 2.5 particles/cm^2 defect density. The access time increase by the ECC circuit is less than 15%.

As another practical example, *semi-distance codes*, a class of asymmetric error masking codes [BLAU93], have been applied to ROM bus line circuits [MATS88, 90]. The faults caused by open- or short-circuit defects in the bus lines can be made asymmetric by controlling the bus drivers and the bus terminal gates. These asymmetric faults are tolerated by using new codes based on the concept of *semi-distance* [MATS90]. The technique has the unique feature that no error correction circuits are required, and therefore additional circuits are very small.

4.3.3 Further Discussion on Recent Memory On-Chip ECCs

Application of the Hamming $(136, 128)$ SEC codes to 16 M-bit DRAM chips may be an option for tolerating soft error problems. It can relax the problems of circuit area overhead and error correction operating time. The on-chip ECC circuit requires 12% to 20% chip overhead and around 10% access time penalty [FURU89, ARIM90].

For the experimental 1 M-bit cache DRAM [ASAK90], which consists of 1 M-bit DRAM operating as a main memory and 8 K-bit SRAM as a cache, $(40, 32)$ modified Hamming SEC-DED code has been applied to the DRAM part in the chip. On-chip ECC circuit requires 12 ns access penalty, amounting to a 15% access time overhead and around a 15% chip area overhead. The soft-error rate is improved by more than ten orders of magnitude.

Augmented Product Code (APC) A new class of product codes, called an *augmented product code (APC)* with double-bit error correction capability, has been applied to the DRAM chip whose memory cell capacitors have a *trench structure* [MAZU92]. It is known that if the α-particles are incident to the intervening space between two-adjoining vertically mounted trench capacitors, the resulting plasma discharge may delete the data in both capacitors. This is a simulation result of the charge-sharing mechanism due to α-particle-induced plasma shorts between adjoining capacitors [CHER86]. The trench capacitors produce double-bit upsets very frequently in a DRAM chip.

For example, take a rectangular subarray of size $m_1 \times m_2$, that is, a subarray of m_2 memory cells in each of its m_1 word-lines. To construct a product code, or a cross-parity code, for each word line, we organize the m_2 cells in the form of a logical rectangular array of size $(p + 1) \times (q + 1)$ where m information bits describe the inner array $p \times q = m$, and the $(p + 1)$-th (bottom-most) row and the $(q + 1)$-th (right-most) column consist of parity bits. In this case, if $m_1 m_2 = m^2$, the DRAM is called a nonredundant one, but a fault-tolerant DRAM requires $m_1 m_2 > m^2$. The augmented product code (APC) is constructed by adding a set of p-diagonal parity bits (if $p \geq q$) to the regular horizontal and vertical parity bits ($p + q + 1$ bits) of the product code. That is, the code requires $2p + q + 1$ check bits. This $(pq, pq - (2p + q + 1))$ APC is a distance-5 code, so it can correct all double-bit errors in the readout word, including these parity bits.

The on-chip APC is evaluated for 1 M-bit to 64 M-bit DRAMs such that chip area overhead is around 10% to 15% and timing overhead is 14% to 18%.

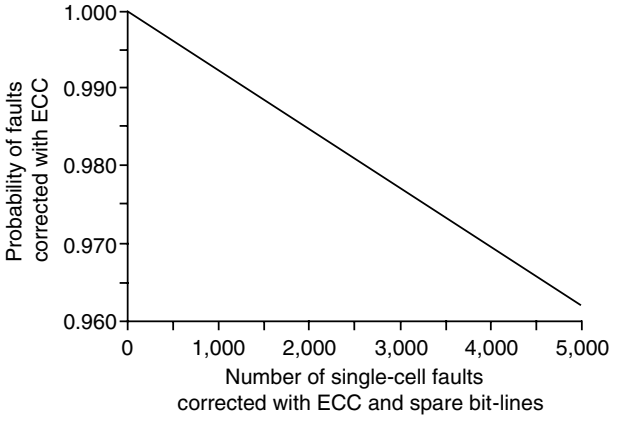


Figure 4.15 *Reliability improvements.* Source: *[KALT90]. © 1990 IEEE.*

Combination of ECC and Spare Bit-Lines / Word-Lines A combination of ECC and spare circuits has been shown to enhance both reliability and yield [KALT90, FIFI91]. For the 16 M-bit DRAM chip divided into four quadrants each having 4 M-bit capacity, the $(137, 128)$ odd-weight-column SEC-DED code, shown in Subsection 4.1.1, and the spare circuits of 32 redundant bit-lines and 24 redundant word-lines per quadrant were applied and proved to improve both reliability and yield dramatically. Improvement in fault tolerance to manufacturing defects was obtained with an increase of chip area by less than 11% and an addition of access time by 10%.

Figures 4.15, 4.16, and 4.17 show reliability improvements, yield improvements with on-chip ECC and spare bit-lines, and yield improvements with on-chip ECC and spare

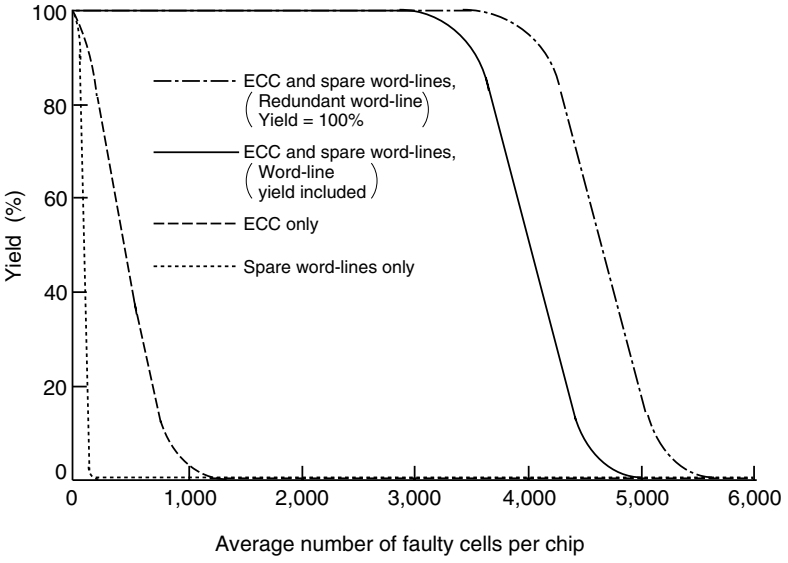


Figure 4.16 *Yield improvement with on-chip ECC and spare word-lines.* Source: *[KALT90]. © 1990 IEEE.*

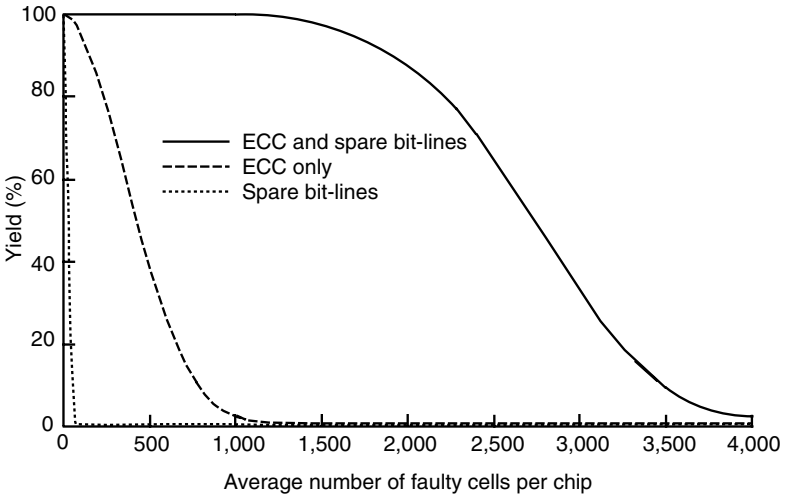


Figure 4.17 *Yield improvement with on-chip ECC and spare bit-lines.* Source: *[KALT90]. © IEEE.*

word-lines, respectively. Figure 4.15 shows the effectiveness of ECC circuits in correcting faults as a function of the number of hard random single-cell faults originally in the chip. The important result shown by the curves in Figure 4.16 is that without use of the ECC circuits (i.e., use of only spare word-lines), an average of 186 randomly failing single-cells per chip has an expected yield of 50% for this chip. Use of ECC circuits only and no word-line redundancy results in a 50% yield for an average of 428 random single-cell faults per chip. Combined use of the ECC and the spare word-lines produces a 50% yield at an average of 4,661 randomly failing single cells per chip. This effect is synergistic [FIFI91]. This is also true to combined use of the ECC and the spare bit-lines, as shown in Figure 4.17.

EXERCISES

4.1 For the following **H** matrix of the (15, 10) odd-weight-column SEC-DED code, answer the following questions:

$$\mathbf{H} = \begin{array}{c|ccccccccccccccc} & d_0 & d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & d_8 & d_9 & c_0 & c_1 & c_2 & c_3 & c_4 \\ \hline & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{array}.$$

(a) Encode the data $(d_0\ d_1\ \ldots\ d_9) = (101101101)$.

(b) The received data $(d_0\ d_1\ \ldots\ d_9\ c_0\ \ldots\ c_4) = (1\,0\,1\,0\,1\,0\,0\,1\,0\,1\,1\,0\,0\,0\,1)$ has a single-bit error. Which bit is in error?

(c) In the received data, d_1 and d_9 are in error. Find the syndrome pattern.

(d) In the code we can detect some three or larger odd number of bit errors. Find the syndrome pattern of this case.

(e) There are some cases where we cannot detect 4-bit errors. Find some undetectable 4-bit errors.

(f) Assume that we have already had three error pointers on the d_3, d_7, and c_2 bit positions, and now we get a syndrome of $S = (0\,0\,1\,1\,0)$. Explain how to correct the errors. In this case do not use the method of investigating all combination of syndromes due to double-bit errors among the d_3, d_7, and c_3 bits.

4.2 Design the odd-weight-column $(32, 26)$ SEC-DED code and its parallel decoding circuit.

4.3 Prove that the codeword of an odd-weight-column code is of even weight. Next prove that the codeword of a Hamming SEC-DED code is of even weight.

4.4 Discuss an error control method using coding techniques for both memory address information and memory read / write data.

4.5 Discuss an operation sequence of a partial store in the high-speed memory using error correcting codes. Partial store is a special write operation such that a specified part of the codeword (i.e., not all the codeword) is written by a new data, but the other part is unchanged. In this case discuss fast operation sequence of the partial store, and indicate the ECC circuit block diagram.

4.6 Show that the code defined by the following $\mathbf{H}$ matrix (*Melas code* [MELA60]) is a DEC code:

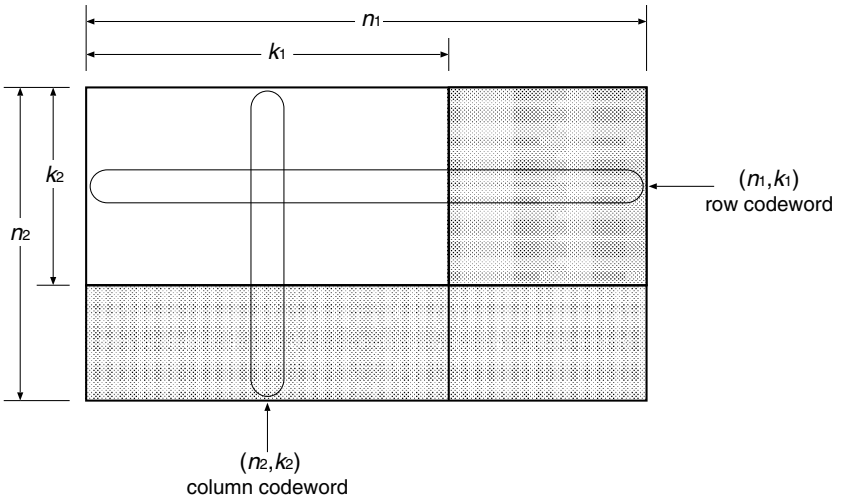
$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \ldots & \alpha^i & \ldots & \alpha^{n-1} \\ 1 & \alpha^{-1} & \alpha^{-2} & \ldots & \alpha^{-i} & \ldots & \alpha^{-(n-1)} \end{bmatrix},$$

$\alpha^i \in GF(2^m), 0 \le i \le n-1, n = 2^m - 1, r = 2m$ where m is an odd number. Next find the parallel decoding procedure of this code [HORI76].

4.7 Assume that a two-dimensional code that includes an (n_1, k_1) row code with a minimum Hamming distance d_1 and an (n_2, k_2) column code with minimum distance d_2 is expressed as an $(n_1 k_1, n_2 k_2)$ code with minimum Hamming distance d, as shown below. This has the constraints on the decoding such that the estimate of any symbol in the information array is a function of only the row and column to which that symbol belongs, rather than the whole codeword. Then the code has the error correcting capability of the sum of the minimum Hamming distances (i.e., $d \ge d_1 + d_2 - 1$) of the constituent codes rather than their product [FUJA85]. Prove this conclusion.

Next, indicate the error correcting capability and the decoding procedure of the SED / SED code (SED: Single-bit error detecting), the SEC / SEC code, and the SED / (SEC-DED) code, where the A / B code shows the A code for the row

code and the B code for the column code, and vice versa, in a two-dimensional code.



4.8 Suppose that the encoding-decoding procedure for an (n, k) SEC-DED code is as follows: (1) input k-bits data are encoded, (2) δ bits $(1 \leq \delta < n)$ in the n bits encoded data are inverted, (3) these n-bits data are stored, (4) the corresponding δ bits in the readout data are inverted, and (5) the resulting n bits are decoded. Find the conditions under which this encoding-decoding scheme can detect all-0 and all-1 readout data errors (i.e., unidirectional errors).

4.9 Prove that undetectable error probabilities of triple-bit errors and quadruple-bit errors, denoted as P_{UD_3} and P_{UD_4}, respectively, for odd-weight-column $(n,\ k)$ SEC-DED codes can be expressed by Eq. (4.4).

4.10 Find the probabilities of P_{UD_3} and P_{UD_4} for non–odd-weight-column $(n,\ k)$ SEC-DED codes.

4.11 Show that the code length n of a rotational odd-weight-column SEC-DED codes can be expressed as follows:

$$n = \sum_{\substack{m|r \\ m:\text{odd}}} \mu(m) 2^{r/m-1},$$

where $\mu(m)$ is a Möbius function defined in Subsection 3.5.2 and $\sum_{\substack{m|r \\ m:\text{odd}}}$ means summation over all odd integer that divides r.

4.12 Find the error detection capabilities of three or larger odd number of bit errors and of four or larger even number of bit errors for the (n, k) odd-weight-column SEC-DED codes. In this case syndrome patterns are assumed to have uniformly occurred. (*Hint*: Consider the syndrome spaces separately of odd number of errors and even number of errors.)

4.13 Design the modularized decoding circuit composed of eight identical subcircuits by using the rotational $(72, 64)$ SEC-DED code shown in Figure 4.2.

4.14 Suppose that the memory having codeword $(d_0\ d_1\ d_2\ d_3\ c_0\ c_1\ c_2\ c_3)$ defined by the following $\mathbf{H}$ matrix has a single soft error in position d_2 in addition to the already existing single hard error in position c_2. Explain how to correct these double errors using the mask error correction techniques

$$\mathbf{H} = \begin{array}{c} \begin{matrix} d_0 & d_1 & d_2 & d_3 & c_0 & c_1 & c_2 & c_3 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}.$$

4.15 Prove that Imai-Kamiyanagi code is a DEC code.

4.16 Prove that DEC-BCH codes satisfy Eq. (4.8).

4.17 Show that the key component of Eq. (4.9) represents the relation between the syndromes S_1 and S_3 in the DEC-BCH codes.

4.18 Prove that the augmented product code (APC) is a distance-5 code.

REFERENCES

[AICH84] F. J. Aichelmann Jr., "Fault-Tolerant Design Techniques for Semiconductor Memory Applications," *IBM J. Res. Dev.*, 28 (March 1984): 177–183.

[ARIM90] K. Arimoto, Y. Matsuda, K. Furutani, M. Tsukude, T. Ooishi, K. Mashiko, and K. Fujishima, "A Speed-Enhanced DRAM Array Architecture with Embedded ECC," *IEEE J. Solid-State Circ.*, 25 (February 1990): 11–17.

[ASAK90] M. Asakura, Y. Matsuda, H. Hidaka, Y. Tanaka, and K. Fujishima, "An Experimental 1-Mbit Cache DRAM with ECC," *IEEE J. Solid-State Circ.*, 25 (February 1990): 5–10.

[AZUM75] S. Azumi and T. Kasami, "On the Optimal Modified Hamming Codes" (in Japanese), *Trans. IECE Japan*, 58-A (June 1975): 325–330.

[BEAU73] W. F. Beausolail, "Monolithic Memory Utilizing Defective Storage Cells," US Patent 3781826 (December 25, 1973).

[BLAU93] M. Blaum, *Codes for Detecting and Correcting Unidirectional Errors*, IEEE Computer Society Press (1993).

[BOSE86] B.Bose and J.Metzner, "Coding Theory for Fault-Tolerant Systems," in *Fault-Tolerant Computing, Theory and Techniques*, D. K. Pradhan (ed.), Prentice Hall (1986), Ch. 4.

[BOSS80] D. C. Bossen and M. Y. Hsiao, "A System Solution to the Memory Soft Error Problem," *IBM J. Res. Dev.*, 24 (May 1980): 390–397.

[CARL86] R. O. Carlson and C. A. Neugebauer, "Future Trends in Wafer Scale Integration," *Proc. IEEE*, 74 (December 1986): 1741–1752.

[CHEN73] C. L. Chen and W. T. Warren, "A Note on One-Step Majority-Logic Decodable Codes," *IEEE Trans. Info. Theory*, IT-19 (January 1973): 135–137.

[CHEN84] C. L. Chen and M. Y. Hsiao, "Error-Correcting Codes for Semiconductor Memory Applications: A State of Art Review," *IBM J. Res. Dev.*, 28 (March 1984): 124–134.

[CHER86] J. S. Chern, P. Yang, P. Patnaik, and J. A. Seitchik, ''Alpha-Particle-Induced Charge Transfer between Closely Spaced Memory Cells,'' *IEEE Trans. Electr. Devices*, ED-33 (June 1986): 822–834.

[CLIF74] R. A. Cliff and T. R. N. Rao, ''Improving the Yield of LSI Memory Chips by the Application of Coding,'' *Proc. 8th Ann. Princeton Conf. Info. Sci. Syst.* (1974): 386–390.

[CLIF80] R. A. Cliff, ''Acceptable Testing of VLSI Components Which Contain Error Correctors,'' *IEEE J. Solid-State Circ.*, SC-15 (February 1980): 61–70.

[DAVI85] L. Davis, ''A Word-Wide 1Mb ROM with Error Correction,'' *Dig., 1985 IEEE Int. Solid-State Circ. Conf., WAM3.2* (February 1985): 40–41.

[DAVY91] A. A. Davydov and L. M. Tombak, ''An Alternative to the Hamming Code in the Class of SEC-DED Codes in Semiconductor Memory,'' *IEEE Trans. Info. Theory*, 37 (May 1991): 897–902.

[DEZA82] K. Dezaki and H. Imai, ''Error Control for Main Memories Using Parity Checkers in Memory Chips'' (in Japanese), *Trans. IECE Japan*, J65-D (August 1982): 1034–1040.

[EGAW80] Y. Egawa, T. Wanda, Y. Ohmori, N. Tsuda, and K. Masuda, ''A 1-Mbit Full-Wafer MOS RAM,'' *IEEE J. Solid-State Circ.*, SC-15 (August 1980): 677–686.

[FIFI91] J. A. Fifield and C. H. Stapper, ''High-Speed On-Chip ECC for Synergistic Fault-Tolerant Memory Chips,'' *IEEE J. Solid-State Circ.*, 26 (October 1991): 1449–1452.

[FUJA85] T. Fuja. C. Heegard, and R. Goodman, ''Some Linear Sum Codes for Random Access Memories,'' *Proc. IEEE Int. Symp. Info. Theory* (June 1985).

[FUJI75] E. Fujiwara and K. Aoki, ''Reliability of Main Memories'' (in Japanese), *J. Info. Proc. Soc. Japan*, 16 (April 1975): 295–304.

[FUJI76] E. Fujiwara and T. Kawakami, ''A Memory Having Double Error Correction Capability—Sequential Correction for BCH Code,'' (in Japanese). *Paper of Technical Group, IECE Japan*, EC 76–20 (June 1976).

[FUJI78] E. Fujiwara, ''Odd-Weigh-Column *b*-Adjacent Error Correcting Codes,'' *Trans. IECE Japan*, E61 (October 1978): 781–787.

[FUJI80] E. Fujiwara and K. Haruta, ''Design of Main Storage Error Checking and Correcting Circuit for LSI Implementation'' (in Japanese), *Trans. IECE Japan*, 63-D (February 1980): 129–136.

[FUJI81] E. Fujiwara, ''Error Correcting Code and its Application to Digital Systems'' (in Japanese), PhD dissertation, Tokyo Institute of Technology (April 1981).

[FUJI82] E. Fujiwara and S. Kaneda, ''Application of Error Correcting Codes for Increasing Computer System Reliability'' (in Japanese), *J. Info. Process. Soc. (IPS) Japan*, 23 (April 1982): 292–298.

[FUJI90] E. Fujiwara, in H. Imai (ed.), *Essentials of Error-Correcting Coding Techniques*, Academic Press (1990), ch. 4.

[FURU89] K. Furutani, K. Arimoto, H. Miyamoto, T. Kobayashi, K. Yasuda, and K. Mashiko, ''A Built-in Hamming Code ECC Circuit for DRAM's,'' *IEEE J. Solid-State Circ.*, 24 (February 1989): 50–56.

[GHAF84] K. A. Ghaffar and R. J. McEliece, ''Soft Error Correction for Increased Densities in VLSI Memories,'' *Proc. 11th IEEE Int. Symp. on Computer Architecture* (1984): 248–250.

[GOLA83] P. Golan and J. Hlavika, ''A Method for Parallel Decoding of Double-Error Correcting Group Codes,'' *Dig., 13th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1983): 338–341.

[HAMM50] R. W. Hamming, ''Error Detecting and Error Correcting Codes,'' *Bell Syst. Techn. J.*, 26 (April 1950): 147–160.

[HAN87] S. H. Han and M. Molek, ''A New Technique for Error Detection and Correction in Semiconductor Memories,'' *Proc. IEEE Int. Test Conf.* (1987): 864–870.

[HORI75] T. Horiguchi and K. Morita, "A Parallel Memory with Double Error Correction Capability—A Class of One-Step Majority-Logic Decodable Error Correction Codes" (in Japanese), *Paper of Technical Group IECE Japan*, EC 75–42 (November 1975).

[HORI76] T. Horiguchi, "A Double Error Correcting Code in Main Memory—On Parallel Decoding of DEC-Melas Codes and BCH Codes" (in Japanese), *Paper of Technical Group IECE Japan*, EC76-61 (November 1976).

[HOWE77] T. H. Howell, G. E. Cregg, and L. Rabins, "Table Lookup Direct Decoder for Double-Error Correcting DEC BCH Codes Using a Pair of Syndromes," US Patent 4030067 (June 14, 1977).

[HSIA69] M. Y. Hsiao and J. T. Tou, "Application of Error-Correcting Codes in Computer Reliability Studies," *IEEE Trans. Reliability*, R-18 (August 1969): 108–118.

[HSIA70a] M. Y. Hsiao, "A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes," *IBM J. Res. Dev.*, 14 (July 1970): 395–401.

[HSIA70b] M. Y. Hsiao, D. C. Bossen, and R. T. Chen, "Orthogonal Latin Square Codes," *IBM J. Res. Dev.*, 14 (July 1970): 390–394.

[IMAI77a] H. Imai and Y. Kamiyanagi, "On Parallel Decoders for Double-Error-Correcting BCH Codes" (in Japanese), *Trans. IECE Japan*, J60-D (September 1977): 761–762.

[IMAI77b] H. Imai and Y. Kamiyanagi, "A Construction Method for Double-Error Correcting Codes for Application to Main Memories" (in Japanese), *Trans. IECE Japan*, J60-D (October 1977): 861–868.

[IMAI79] H. Imai and H. Fujiya, "A Construction Method for Simple-Decodable Error-Correcting Codes" (in Japanese), *Trans. IECE Japan*, J62-A (May 1979): 271–277.

[KALT90] H. L. Kalter, C. H. Stapper, J. E. Barth, J. D. Lorenzo, C. E. Drake, J. A. Fifield, G. A. Kelley Jr., S. C. Lewis, W. B. Van den Hoeven, and J. A. Yankosky, "A 50-ns 16-Mb DRAM with a 10-ns Data Rate and On-Chip ECC," *IEEE J. Solid-State Circ.*, 25 (October 1990): 1118–1128.

[KANE84] S. Kaneda and H. Fukuda, "Reliability Design of Memory Systems Considering Soft-Error" (in Japanese), *Trans. IECE Japan*, J67-D (September 1984): 1036–1043.

[KITA80] Y. Kitano, S. Kohda, H. Kikuchi, and S. Sakai, "A 4-Mbit Full-Wafer ROM," *IEEE J. Solid-State Circ.*, SC-15 (August 1980): 686–693.

[LEIG82] F. T. Leighton and C. E. Leiserson, "Wafer Scale Integration of Systolic Arrays," *Proc. 23rd IEEE Symp. on Foundation of Computer Science* (1982): 297–311.

[MANG84a] T. E. Mangir, "Sources of Failures and Yield Improvement for VLSI and Restructurable Interconnects for RVLSI and WSI: Part I—Sources of Failures and Yield Improvement for VLSI," *Proc. IEEE*, 72 (June 1984): 690–708.

[MANG84b] T. E. Mangir, "Sources of Failures and Yield Improvement for VLSI and Restructurable Interconnects for RVLSI and WSI: Part II—Restructurable Interconnects for RVLSI and WSI," *Proc. IEEE*, 72 (December 1984): 1687–1694.

[MANO82] T. Mano, M. Wada, N. Ieda, and M. Tanimoto, "A Redundancy Circuit for a Fault-Tolerant 256K MOS RAM," *IEEE J. Solid-State Circ., SC-17* (August 1982): 726–731.

[MANO83] T. Mano, J. Yamada, J. Inoue, and S. Nakajima, "Circuit Techniques for a VLSI Memory," *IEEE J. Solid-State Circ.*, SC-18 (October 1983): 463–470.

[MANO87] T. Mano, T. Matsumura, J. Yamada, J. Inoue, S. Nakajima, K. Minegishi, K. Miura, T. Matsuda, C. Hashimoto, and H. Namatsu, "Circuit Technologies for 16Mb DRAMs," *Dig., 1987 IEEE Int. Solid-State Circuit Conf.* (February 1987): 22–23.

[MATS77] K. Matsuzawa and Y. Tohma, "A Way of Multiple Error Correction for Computer Main Memory" (in Japanese), *Trans. IECE Japan*, J60-D (October 1977): 869–876.

[MATS78] K. Matsuzawa and Y. Tohma, "An Adjacent-Error-Correcting Code Based on the Threshold Decoding," *Dig., 8th IEEE Int. Symp. Fault-Tolerant Computing* (1978): 225.

[MATS87] H. Matsuda and M. Kasahara, "A New Method of Constructing SEC-DED Codes with Reduced Number of Codewords of Weight 4," *Trans. IECE Japan*, J70-A (August 1987): 1036–1045.

[MATS88] K. Matsuzawa and E. Fujiwara, "Masking Asymmetric Line Faults Using Semi-Distance Codes," *Dig., 18th IEEE Int. Symp. Fault-Tolerant Computing* (June 1988): 354–359.

[MATS90] K. Matsuzawa and E. Fujiwara, "Masking Asymmetric Line Faults Using Semi-distance Codes," *Trans. IEICE*, E73 (August 1990): 1278–1286.

[MAY79] T. C. May, "Soft Errors in VLSI: Present and Future," *IEEE Trans. Comp. Hybrids Manuf. Technol.*, CHMT-2 (December 1979): 377–387.

[MAZU92] P. Mazumder, "On-Chip ECC Circuit for Correcting Soft Errors in DRAM's with Trench Capacitors," *IEEE J. Solid-State Circ.*, 27 (November 1992): 1623–1633.

[MELA60] C. M. Melas, "A Cyclic Code for Double Error Correction," *IBM J.* (July 1960): 142–143.

[MOOR86] W. R. Moore, "A Review of Fault-Tolerant Techniques for the Enhancement of Integrated Circuit Yield," *Proc. IEEE*, 74 (May 1986): 684–698.

[NOOR80] D. J. W. Noorlag, L. M. Terman, and A. G. Konhein, "The Effect of Alpha-Particle-Induced Soft Error on Memory Systems with Error Correction," *IEEE J. Solid-State Circ.*, SC-15 (June 1980): 319–325.

[OHNI81] N. Ohnishi, T. Ishikawa, and N. Mutoh, "System Configuration on Full Wafer LSI," *Proc. IEEE Int. Symp. Mini and Microcomputers* (January 1981): 7–11.

[OKAN87] H. Okano and H. Imai, "A Construction Method of High-Speed Decoders Using ROMs for Bose-Chaudhuri-Hocquenghem and Reed-Solomon Codes," *IEEE Trans. Comput.*, C-36 (October 1987): 1165–1171.

[OGOR96] T. J. O'Gorman, J. M. Rose and A. H. Taber, et al., "Field Testing for Cosmic Ray Soft Errors in Semiconductor Memories," *IBM J. Res. Dev.*, 40 (January 1996): 41–50.

[PATE72a] A. M. Patel and M. Y. Hsiao, "An Adaptive Error Correction Scheme for Computer Memory System," *Proc. IEEE Fall Joint Computer Conf. AFIPS* (1972): 83–87.

[PATE72b] A. M. Patel and S. J. Hong, "Syndrome Trapping Technique for Fast Double-Error Correction," *Proc. IEEE Int. Symp. on Info. Theory* (1972). (See also US Patent 3714629, January 1973.)

[PELT83] D. L. Peltzer, "Wafer-Scale Integration: The Limits of VLSI?" *VLSI Design* (September 1983): 43–47.

[PETE72] W. W. Peterson and E. J. Weldon Jr., *Error Correcting Codes*, 2d ed., MIT Press (1972).

[RAO74] R. N. Rao, *Error Coding for Arithmetic Processors*, Academic Press (1974).

[SAIH82] G. A. Sai-Halasz, M. R. Wordman, and R. H. Dennard, "Alpha-Particle-Induced Soft Error Rate in VLSI Circuits," *IEEE J. Solid-State Circ.*, SC-17 (April 1982): 355–361.

[SHIN83] T. Shinoda, Y. Ohnishi, H. Kawamoto, K. Takizawa, and K. Narita, "A 1 Mb ROM with On-Chip ECC for Yield Enhancement," *Dig., 1983 IEEE Int. Solid-State Circ. Conf.* (February 1983): 158–159.

[SMIT81] R. T. Smith, J. D. Chlipala, J. F. M. Bindels, R. G. Nelson, F. H. Fischer, and T. F. Mantz, "Laser Programmable Redundancy and Yield Improvement in a 64K DRAM," *IEEE J. Solid-State Circ.*, SC-16 (October 1981): 506–514.

[STAP80] C. H. Stapper, A. N. Mclaren, and M. Dreckmann, "Yield Model for Productivity Optimization of VLSI Memory Chips with Redundancy and Partially Good Product," *IBM J. Res. Dev.*, 24 (May 1980): 398–409.

[STAP83] C. H. Stapper, "Modelling of Integrated Circuit Defect Sensitivities," *IBM J. Res. Dev.*, 27 (November 1983): 549–557.

[STAP84] C. H. Stapper, "Yield Model for Fault Clusters within Integrated Circuits," *IBM J. Res. Dev.*, 28 (September 1984): 636–690.

[STAP85] C. H. Stapper, "The Effects of Wafer to Wafer Perfect Density Variations on Integrated Circuit Defect and Fault Distributions," *IBM J. Res. Dev.*, 29 (January 1985): 87–97.

[STAP86] C. H. Stapper, "On Yield, Fault Distributions and Clustering of Particles," *IBM J. Res. Dev.*, 30 (May 1986): 326–338.

[TAKE77] T. Takezono and H. Ando, "Error Correcting Circuit Arrangement Using Cube Circuits," US Patent 40064483 (December 1977).

[TANG69] D. T. Tang and R. T. Chien, "Coding for Error Control," *IBM Syst. J.*, 8 (1969): 48–86.

[TANN84] R. M. Tanner, "Fault-Tolerant 256K Memory Designs," *IEEE Trans. Comput.*, C-33 (April 1984): 314–322.

[WOOD86] W. K. S. Walker, C.-E. W. Sundberg, and C. J. Black, "A Reliable Spaceborne Memory with a Single Error and Erasure Correction Scheme," *IEEE Trans. Comput.*, C-28 (July 1979): 493–500.

[UEOK84] Y. Ueoka, C. Minagawa, M. Oka, and A. Ishimoto, "A Detect-Tolerant Design for Full-Wafer Memory LSI," *IEEE J. Solid-State Circ.*, SC-19 (June 1984): 319–324.

[YAMA80] A. Yamagishi and H. Imai, "A Construction Method for Decoders of BCH Codes Using ROM's" (in Japanese), *Trans. IECE Japan*, 63-D (December 1980): 1034–1041.

[YAMA84a] J. Yamada, T. Mano, J. Inoue, S. Nakajima, and T.Matsuda, "A Submicron VLSI Memory with a 4 Bit-at-a-Time Built-in ECC Circuit," *Dig., 1984 IEEE Int. Solid-State Circ. Conf.* (February 1984): 104–105.

[YAMA84b] J. Yamada, T. Mano, and S. Date, "Built-in ECC Techniques for LSI Memories" (in Japanese), *Trans. IEICE Japan*, J67-C (October 1984): 777–784. (Translated in *Electronics and Communications in Japan*, Part 2, 68 (1985): 19–26.)

[YAMA87a] K. Yamashita, A. Kanasugi, S. Hijiya, G. Goto, N. Matsumura, and T.Shirato, "A Wafer-Scale 170,000-Gate FFT Processor with Built-in Test Circuits," *Proc. IEEE 1987 Custom Integrated Circuits Conf.* (May 1987): 207–210.

[YAMA87b] J. Yamada, "Selector-Line Merged Built-in ECC Technique for DRAMs," *IEEE J. Solid-State Circ.*, SC-22 (October 1987): 868–873.

[YAMA88] T. Yamada, H. Kotani, J. Matsushima, and M. Inoue, "A 4-M Bit DRAM with 16-Bit Concurrent ECC," *IEEE J. Solid-State Circ.*, SC-23 (February 1988): 20–26.

CONTENTS

5

Codes for High-Speed Memories II: Byte Error Control Codes

In the application of error correcting codes to computer systems, there are a number of situations where an error-correcting code capable of correcting clusters of adjacent bits in error is uniquely suited. An example is errors due to a failure in a b-bit organized semiconductor memory chip, which is called a *byte-organized memory chip*. In this chapter we refer to this cluster of b bits, $b \geq 2$, as a *byte*. With the advent of high-density semiconductor memory chips, these b-bit organized RAM chips, for example, $b = 4, 8, 16,$ and 32 bits organized RAM chips, have been fabricated and are now marketed. If a failure occurs in such a chip, the resulting information read out from the memory is likely to have the b-bit cluster in error. In this kind of application it may be desirable to have an error control code capable of correcting / detecting byte errors as well as bit errors [FUJI82, HORI83, CHEN83, DENG87, FUJI90].

The recent high-density RAM chip with wide input / output (I / O) data of 8, 16, and 32 bits has an inside structure organized by multiple subarrays almost physically separated from each other. For this organization more suitable byte error control codes have been studied.

This chapter deals with design of practical byte error correcting / detecting codes for high-speed semiconductor memories. From a practical standpoint, it is about the code design method for controlling at most double-byte errors. These practical code classes are abbreviated and designated as follows:

1. SbEC codes: Single b-bit byte error correcting codes.
2. SbEC-DbED codes: Single b-bit byte error correcting and double b-bit byte error detecting codes.

Code Design for Dependable Systems: Theory and Practical Applications, by Eiji Fujiwara

3. SbEC-S$_{p\times b/B}$ED codes: Single b-bit byte error correcting and single p-byte within a B-bit block error detecting codes.

The SbEC-DbED codes have found many applications in recent large capacity semiconductor memory systems with $b = 4$ bits byte size. The SbEC-S$_{p\times b/B}$ED codes with $b = 4, p = 2$, and $B = 16$ have also been applied to large-capacity high-speed memory systems using RAM chips each having 16-bit I / O data.

5.1 SINGLE-BYTE ERROR CORRECTING (SbEC) CODES

In this section we discuss a class of single-symbol error correcting codes over $GF(2^b)$. Each symbol is a b-bit byte, and therefore the codes are called SbEC codes.

5.1.1 Hamming-Type Codes

It is well known that a Hamming single-error correcting code can be constructed by elements from any finite field [HAMM50]. If F is such a field, then the **H** matrix for the single-error correcting code with elements from F is constructed as follows: Choose as columns of the **H** matrix all the nonzero r-tuples of elements from F such that no column of **H** is a multiple of another column. Then, since every pair of columns is linearly independent, the code has $d_{\min} = 3$, that is, the code is capable of correcting single (symbol) errors.

If, in particular, F is $GF(2^b)$, these Hamming codes are an SbEC class of codes. To implement the Hamming-type SbEC code, it is necessary to transform the **H** matrix over $GF(2^b)$ to a binary form as follows [BOSS70, HONG72].

Definition 5.1 Given a binary primitive polynomial $\mathbf{g}(x)$ of degree b, the *companion matrix* **T** corresponding to $\mathbf{g}(x)$ is defined as

$$\mathbf{g}(x) = \sum_{i=0}^{b} g_i x^i, \qquad g_0 = g_b = 1,$$

$$\mathbf{T} = \begin{bmatrix} 0 & 0 \cdots 0 & g_0 \\ & \mathbf{I}_{b-1} & g_1 \\ & & \vdots \\ & & g_{b-1} \end{bmatrix}_{b\times b},$$

$\mathbf{I}_{b-1} : (b-1)\times(b-1)$ identity matrix. □

Let α be a primitive element in $GF(2^b)$ and a root of $\mathbf{g}(x)$. Its companion matrix **T** has as its columns $\left|\ \alpha^i\ \right|$ for $i = 1, 2, \ldots, b$, where $\left|\ \alpha^i\ \right|$ is the coefficient vector of $x^i \bmod \mathbf{g}(x)$.

The companion matrix of α^j is $\mathbf{T}^j$, and its column vectors are shown in Eq. (5.2). (Also see Example 5.1.)

$$\mathbf{T} = \begin{bmatrix} | & | & | & & | \\ \alpha & \alpha^2 & \alpha^3 & \dots & \alpha^b \\ | & | & | & & | \end{bmatrix}_{b\times b}, \qquad \mathbf{T}^j = \begin{bmatrix} | & | & & | \\ \alpha^j & \alpha^{j+1} & \dots & \alpha^{j+b-1} \\ | & | & & | \end{bmatrix}_{b\times b}. \tag{5.2}$$

Some of the properties of the companion matrix are covered next [HONG72].

Property 1 Let e be the exponent of $\mathbf{g}(x)$ (i.e., $y = e$ is the least positive solution of $x^y \equiv 1 \bmod \mathbf{g}(x)$).

(**a**) $\mathbf{T}$ is nonsingular.
(**b**) $\mathbf{T}^0 = \mathbf{T}^e = \mathbf{I}_b$.
(**c**) $\mathbf{T}^i = \mathbf{T}^j$ if and only if $i \equiv j \bmod e$.

Property 2 The i-th column of the $\mathbf{T}^j$ is the same as the coefficient vector of the $(b-1)$-th degree polynomial $x^{i+j-1} \bmod \mathbf{g}(x)$.

Property 3 Let V be the coefficient column vector of $\mathbf{v}(x) = \sum_{i=0}^{b-1} v_i x^i$ and V' for $\mathbf{v}'(x) = \sum_{i=0}^{b-1} \mathbf{v}'_i x^i$. Then $\mathbf{T}^i \cdot V = V'$ if and only if $x^i \mathbf{v}(x) = \mathbf{v}'(x) \bmod \mathbf{g}(x)$.

The set of companion matrices and the included zero matrix have the same structure as $GF(2^b)$ and are field isomorphic to $GF(2^b)$. Therefore we state

$$\{\mathbf{0}, \quad \mathbf{T}, \quad \mathbf{T}^2, \quad \mathbf{T}^3, \dots, \mathbf{T}^{2^b-2}, \mathbf{T}^{2^b-1} = \mathbf{I}\} = GF(2^b),$$

where $\mathbf{I}$ is the $b \times b$ identity matrix and $\mathbf{0}$ is the $b \times b$ zero matrix. In the following example we use the $\mathbf{T}$ matrices to represent $GF(2^b)$ as well as the vectors (b-tuples over binary) as required.

Example 5.1

The companion matrix $\mathbf{T}$ in $GF(2^4)$ defined by the primitive polynomial $\mathbf{g}(x) = x^4 + x + 1$ is given as follows:

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

This matrix is used to express all elements included in $GF(2^4)$ in binary form as follows:

$$\mathbf{0} = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix} \quad \mathbf{I} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} & & & 1 \\ 1 & & & 1 \\ & 1 & & \\ & & 1 & \end{bmatrix} \quad \mathbf{T}^2 = \begin{bmatrix} & & 1 & \\ & & 1 & 1 \\ 1 & & & 1 \\ & 1 & & \end{bmatrix}$$

$$\mathbf{T}^3 = \begin{bmatrix} & 1 & & \\ & 1 & 1 & \\ & & 1 & 1 \\ 1 & & & 1 \end{bmatrix} \quad \mathbf{T}^4 = \begin{bmatrix} 1 & & & 1 \\ 1 & 1 & & 1 \\ & 1 & 1 & \\ & & 1 & 1 \end{bmatrix} \quad \mathbf{T}^5 = \begin{bmatrix} & & 1 & 1 \\ 1 & & 1 & \\ 1 & 1 & & 1 \\ & 1 & 1 & \end{bmatrix} \quad \mathbf{T}^6 = \begin{bmatrix} & 1 & 1 & \\ & 1 & & 1 \\ 1 & & 1 & \\ 1 & 1 & & 1 \end{bmatrix}$$

$$\mathbf{T}^7 = \begin{bmatrix} 1 & 1 & & 1 \\ 1 & & 1 & 1 \\ & 1 & & 1 \\ 1 & & 1 & \end{bmatrix} \quad \mathbf{T}^8 = \begin{bmatrix} 1 & & 1 & \\ & 1 & 1 & 1 \\ 1 & & 1 & 1 \\ & 1 & & 1 \end{bmatrix} \quad \mathbf{T}^9 = \begin{bmatrix} & 1 & & 1 \\ 1 & 1 & 1 & 1 \\ & 1 & 1 & 1 \\ 1 & & 1 & 1 \end{bmatrix} \quad \mathbf{T}^{10} = \begin{bmatrix} 1 & & 1 & 1 \\ 1 & 1 & 1 & \\ 1 & 1 & 1 & 1 \\ & 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{T}^{11} = \begin{bmatrix} & 1 & 1 & 1 \\ 1 & 1 & & \\ 1 & 1 & 1 & \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad \mathbf{T}^{12} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & & & \\ 1 & 1 & & \\ 1 & 1 & 1 & \end{bmatrix} \quad \mathbf{T}^{13} = \begin{bmatrix} 1 & 1 & 1 & \\ & & & 1 \\ 1 & & & \\ 1 & 1 & & \end{bmatrix} \quad \mathbf{T}^{14} = \begin{bmatrix} 1 & 1 & & \\ & & 1 & \\ & & & 1 \\ 1 & & & \end{bmatrix}.$$

The addition and multiplication rules in $GF(2^4)$ are determined as follows:

$+$	$\mathbf{0}$	$\mathbf{I}$	$\mathbf{T}$	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{I}$	$\mathbf{T}$	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$
$\mathbf{I}$	$\mathbf{I}$	$\mathbf{0}$	$\mathbf{T}^4$	$\mathbf{T}^8$	$\mathbf{T}^{14}$	$\mathbf{T}$	$\mathbf{T}^{10}$	$\mathbf{T}^{13}$	$\mathbf{T}^9$	$\mathbf{T}^2$	$\mathbf{T}^7$	$\mathbf{T}^5$	$\mathbf{T}^{12}$	$\mathbf{T}^{11}$	$\mathbf{T}^6$	$\mathbf{T}^3$
$\mathbf{T}$	$\mathbf{T}$	$\mathbf{T}^4$	$\mathbf{0}$	$\mathbf{T}^5$	$\mathbf{T}^9$	$\mathbf{I}$	$\mathbf{T}^2$	$\mathbf{T}^{11}$	$\mathbf{T}^{14}$	$\mathbf{T}^{10}$	$\mathbf{T}^3$	$\mathbf{T}^8$	$\mathbf{T}^6$	$\mathbf{T}^{13}$	$\mathbf{T}^{12}$	$\mathbf{T}^7$
$\mathbf{T}^2$	$\mathbf{T}^2$	$\mathbf{T}^8$	$\mathbf{T}^5$	$\mathbf{0}$	$\mathbf{T}^6$	$\mathbf{T}^{10}$	$\mathbf{T}$	$\mathbf{T}^3$	$\mathbf{T}^{12}$	$\mathbf{I}$	$\mathbf{T}^{11}$	$\mathbf{T}^4$	$\mathbf{T}^9$	$\mathbf{T}^7$	$\mathbf{T}^{14}$	$\mathbf{T}^{13}$
$\mathbf{T}^3$	$\mathbf{T}^3$	$\mathbf{T}^{14}$	$\mathbf{T}^9$	$\mathbf{T}^6$	$\mathbf{0}$	$\mathbf{T}^7$	$\mathbf{T}^{11}$	$\mathbf{T}^2$	$\mathbf{T}^4$	$\mathbf{T}^{13}$	$\mathbf{T}$	$\mathbf{T}^{12}$	$\mathbf{T}^5$	$\mathbf{T}^{10}$	$\mathbf{T}^8$	$\mathbf{I}$
$\mathbf{T}^4$	$\mathbf{T}^4$	$\mathbf{T}$	$\mathbf{I}$	$\mathbf{T}^{10}$	$\mathbf{T}^7$	$\mathbf{0}$	$\mathbf{T}^8$	$\mathbf{T}^{12}$	$\mathbf{T}^3$	$\mathbf{T}^5$	$\mathbf{T}^{14}$	$\mathbf{T}^2$	$\mathbf{T}^{13}$	$\mathbf{T}^6$	$\mathbf{T}^{11}$	$\mathbf{T}^9$
$\mathbf{T}^5$	$\mathbf{T}^5$	$\mathbf{T}^{10}$	$\mathbf{T}^2$	$\mathbf{T}$	$\mathbf{T}^{11}$	$\mathbf{T}^8$	$\mathbf{0}$	$\mathbf{T}^9$	$\mathbf{T}^{13}$	$\mathbf{T}^4$	$\mathbf{T}^6$	$\mathbf{I}$	$\mathbf{T}^3$	$\mathbf{T}^{14}$	$\mathbf{T}^7$	$\mathbf{T}^{12}$
$\mathbf{T}^6$	$\mathbf{T}^6$	$\mathbf{T}^{13}$	$\mathbf{T}^{11}$	$\mathbf{T}^3$	$\mathbf{T}^2$	$\mathbf{T}^{12}$	$\mathbf{T}^9$	$\mathbf{0}$	$\mathbf{T}^{10}$	$\mathbf{T}^{14}$	$\mathbf{T}^5$	$\mathbf{T}^7$	$\mathbf{T}$	$\mathbf{T}^4$	$\mathbf{I}$	$\mathbf{T}^8$
$\mathbf{T}^7$	$\mathbf{T}^7$	$\mathbf{T}^9$	$\mathbf{T}^{14}$	$\mathbf{T}^{12}$	$\mathbf{T}^4$	$\mathbf{T}^3$	$\mathbf{T}^{13}$	$\mathbf{T}^{10}$	$\mathbf{0}$	$\mathbf{T}^{11}$	$\mathbf{I}$	$\mathbf{T}^6$	$\mathbf{T}^8$	$\mathbf{T}^2$	$\mathbf{T}^5$	$\mathbf{T}$
$\mathbf{T}^8$	$\mathbf{T}^8$	$\mathbf{T}^2$	$\mathbf{T}^{10}$	$\mathbf{I}$	$\mathbf{T}^{13}$	$\mathbf{T}^5$	$\mathbf{T}^4$	$\mathbf{T}^{14}$	$\mathbf{T}^{11}$	$\mathbf{0}$	$\mathbf{T}^{12}$	$\mathbf{T}$	$\mathbf{T}^7$	$\mathbf{T}^9$	$\mathbf{T}^3$	$\mathbf{T}^6$
$\mathbf{T}^9$	$\mathbf{T}^9$	$\mathbf{T}^7$	$\mathbf{T}^3$	$\mathbf{T}^{11}$	$\mathbf{T}$	$\mathbf{T}^{14}$	$\mathbf{T}^6$	$\mathbf{T}^5$	$\mathbf{I}$	$\mathbf{T}^{12}$	$\mathbf{0}$	$\mathbf{T}^{13}$	$\mathbf{T}^2$	$\mathbf{T}^8$	$\mathbf{T}^{10}$	$\mathbf{T}^4$
$\mathbf{T}^{10}$	$\mathbf{T}^{10}$	$\mathbf{T}^5$	$\mathbf{T}^8$	$\mathbf{T}^4$	$\mathbf{T}^{12}$	$\mathbf{T}^2$	$\mathbf{I}$	$\mathbf{T}^7$	$\mathbf{T}^6$	$\mathbf{T}$	$\mathbf{T}^{13}$	$\mathbf{0}$	$\mathbf{T}^{14}$	$\mathbf{T}^3$	$\mathbf{T}^9$	$\mathbf{T}^{11}$
$\mathbf{T}^{11}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^6$	$\mathbf{T}^9$	$\mathbf{T}^5$	$\mathbf{T}^{13}$	$\mathbf{T}^3$	$\mathbf{T}$	$\mathbf{T}^8$	$\mathbf{T}^7$	$\mathbf{T}^2$	$\mathbf{T}^{14}$	$\mathbf{0}$	$\mathbf{I}$	$\mathbf{T}^4$	$\mathbf{T}^{10}$
$\mathbf{T}^{12}$	$\mathbf{T}^{12}$	$\mathbf{T}^{11}$	$\mathbf{T}^{13}$	$\mathbf{T}^7$	$\mathbf{T}^{10}$	$\mathbf{T}^6$	$\mathbf{T}^{14}$	$\mathbf{T}^4$	$\mathbf{T}^2$	$\mathbf{T}^9$	$\mathbf{T}^8$	$\mathbf{T}^3$	$\mathbf{I}$	$\mathbf{0}$	$\mathbf{T}$	$\mathbf{T}^5$
$\mathbf{T}^{13}$	$\mathbf{T}^{13}$	$\mathbf{T}^6$	$\mathbf{T}^{12}$	$\mathbf{T}^{14}$	$\mathbf{T}^8$	$\mathbf{T}^{11}$	$\mathbf{T}^7$	$\mathbf{I}$	$\mathbf{T}^5$	$\mathbf{T}^3$	$\mathbf{T}^{10}$	$\mathbf{T}^9$	$\mathbf{T}^4$	$\mathbf{T}$	$\mathbf{0}$	$\mathbf{T}^2$
$\mathbf{T}^{14}$	$\mathbf{T}^{14}$	$\mathbf{T}^3$	$\mathbf{T}^7$	$\mathbf{T}^{13}$	$\mathbf{I}$	$\mathbf{T}^9$	$\mathbf{T}^{12}$	$\mathbf{T}^8$	$\mathbf{T}$	$\mathbf{T}^6$	$\mathbf{T}^4$	$\mathbf{T}^{11}$	$\mathbf{T}^{10}$	$\mathbf{T}^5$	$\mathbf{T}^2$	$\mathbf{0}$

$\times$	**0**	**I**	**T**	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
I	**0**	**I**	**T**	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$
T	**0**	**T**	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$	**I**
$\mathbf{T}^2$	**0**	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$	**I**	**T**
$\mathbf{T}^3$	**0**	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$	**I**	**T**	$\mathbf{T}^2$
$\mathbf{T}^4$	**0**	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$	**I**	**T**	$\mathbf{T}^2$	$\mathbf{T}^3$
$\mathbf{T}^5$	**0**	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$	**I**	**T**	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$
$\mathbf{T}^6$	**0**	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$	**I**	**T**	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$
$\mathbf{T}^7$	**0**	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$	**I**	**T**	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$
$\mathbf{T}^8$	**0**	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$	**I**	**T**	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$
$\mathbf{T}^9$	**0**	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$	**I**	**T**	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$
$\mathbf{T}^{10}$	**0**	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$	**I**	**T**	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$
$\mathbf{T}^{11}$	**0**	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$	**I**	**T**	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$
$\mathbf{T}^{12}$	**0**	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$	**I**	**T**	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$
$\mathbf{T}^{13}$	**0**	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$	**I**	**T**	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$
$\mathbf{T}^{14}$	**0**	$\mathbf{T}^{14}$	**I**	**T**	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$

By way of these elements expressed in binary form, the single-symbol error correcting Hamming codes over $GF(2^b)$ are transformed into Hamming-type SbEC codes. This transformation is illustrated in Example 5.2 below.

Example 5.2

Our objective in this example is to obtain a $(72, 64)$ S2EC code from a $(36, 32)$ code over $GF(2^2)$. The companion matrix $\mathbf{T}$ is derived by using the primitive polynomial $\mathbf{g}(x) = x^2 + x + 1$. The following $\mathbf{H}$ matrix shows such a $(72, 64)$ S2EC code whose weight (i.e., total number of 1's in the $\mathbf{H}$ matrix) is near minimal [BOSS70, © 1970 IBM]. Note that every column here has at least one identity element $\mathbf{I}$.

$$\mathbf{H} = \left[\begin{array}{cccccccccccccccccccccccccccccccccccc}
\mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{T}^2 & \mathbf{T} & \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{T}^2 & \mathbf{T} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{T}^2 & \mathbf{T} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{T}^2 & \mathbf{T} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\
\mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{T}^2 & \mathbf{T} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T} & \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}
\end{array}\right],$$

$$\mathbf{0} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \mathbf{T}^2 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}. \tag{5.3}$$

In the Hamming-type code over $GF(2^b)$ the parameters n and k denote the code length (in bytes) and the information length (in bytes), respectively. So the SbEC code derived from these parameters will be an (N, K) code in binary form, where $N = n \times b$ bits and $K = k \times b$ bits. Also the number of check bits is $R = r \times b = (n - k) \times b$.

The maximum length (in bits) of this class of SbEC codes is given by

$$N_H = b \times n = \frac{b(2^{br} - 1)}{2^b - 1}. \tag{5.4}$$

Figure 5.1 shows the relation between the information-bit length K and the check-bit length R of this code for $b = 2, 3$, and 4.

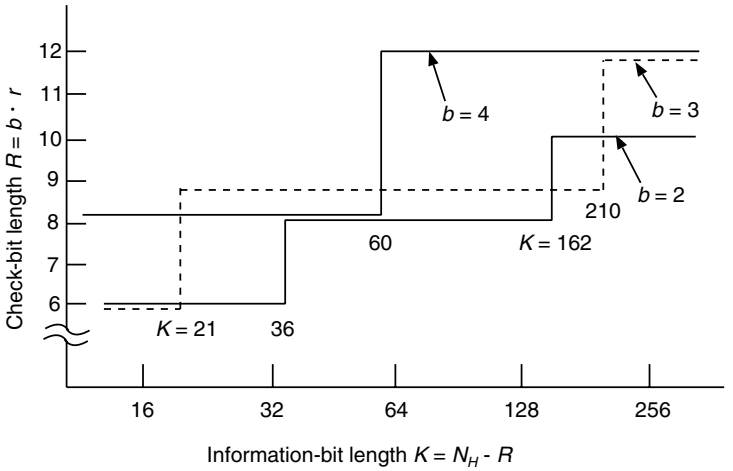


Figure 5.1 *Comparison of check-bit lengths and information-bit lengths of the Hamming-type SbEC codes.*

Next a possible method of implementing a single-error correction with Hamming-type codes is shown. Let syndrome be equal to $S = (s_0\ s_1\ \ldots\ s_{r-1})$ and the i-th column vector of the **H** matrix be $(h_{0,i}\ h_{1,i}\ \ldots\ h_{r-1,i})^T$, where $h_{j,i} \in GF(2^b), j = 0, 1, \ldots, r-1$. Then an error of value E $(E \in GF(2^b))$ in symbol i yields a syndrome that is equal to

$$S = (E \cdot h_{0,i}\ \ E \cdot h_{1,i}\ \ \ldots\ \ E \cdot h_{r-1,i}).$$

Since every column of **H** contains at least one identity element **I**, then the original syndrome contains the error magnitude in one of the positions $s_0, s_1, \ldots, s_{r-1}$. Without loss of generality, assume that $h_{j,i}$ is equal to **I**. Then

$$s_j = E \cdot h_{j,i} = E \cdot \mathbf{I} = E.$$

That is, the error pattern E equals S_j. Under this condition the error byte pointer g_i is a scalar that indicates the location of the i-th byte and is given by the Boolean expression

$$g_i = \bigcap_{\substack{m=0 \\ m \neq j}}^{r-1} [s_m = s_j \cdot h_{m,i}], \qquad \text{where } \bigcap : \text{ AND.}$$

This means that g_i equals 1 only if every relation of $s_m = s_j \cdot h_{m,i}$ is satisfied for $m = 0, 1, 2, \ldots, r-1$, $m \neq j$.

The corrected i-th byte data $\widehat{D}_i$ can be obtained as

$$\widehat{D}_i = D_i + g_i \cdot E \qquad \text{for } i = 0, 1, \ldots, k-1.$$

In this equation the $+$ expresses addition in $GF(2^b)$.

5.1.2 Burton Code and Its Generalized 2-Redundant Codes

There is an interesting subclass of Hamming codes over $GF(2^b)$ that has only two check symbols and is capable of correcting single errors in $GF(2^b)$. These codes are known as *2-redundant code* for this reason [BOSS70]. The $\mathbf{H}$ matrix of the Hamming-type 2-redundant code has the following form:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} & \mathbf{I} & \mathbf{0} \\ \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \cdots & \mathbf{T}^{2^b-2} & \mathbf{0} & \mathbf{I} \end{bmatrix}.$$

It is clear that no two columns of $\mathbf{H}$ are linearly dependent. The code has distance-3 and is therefore a single-error correcting code over $GF(2^b)$.

Burton code [BURT71] is also a 2-redundant SbEC code. This code is called a class of *phased-burst error correcting cyclic code*. The generator polynomial of the code is expressed as

$$\mathbf{g}(x) = (x^b + 1) \cdot \mathbf{p}(x). \tag{5.5}$$

Here $\mathbf{p}(x)$ is an irreducible polynomial of degree b. The maximal code length N (bits) of this code is given by

$$N = \mathrm{LCM}(e, b),$$

where LCM denotes the least common multiple and e the exponent (or the period) of $\mathbf{p}(x)$. The code defined by Eq. (5.5) can be expressed as the following $\mathbf{H}$ matrix:

$$\mathbf{H} = \left[\begin{array}{cccccc} \mathbf{I}_b & \mathbf{I}_b & \mathbf{I}_b & \mathbf{I}_b & \cdots & \mathbf{I}_b \\ \hline & \mathbf{H}' & \mathbf{H}' & \cdots & \mathbf{H}' & \end{array}\right], \tag{5.6}$$

where $\mathbf{I}_b$ is the $b \times b$ identity matrix and $\mathbf{H}'$ is an $\mathbf{H}$ matrix of the binary Hamming code generated by the polynomial $\mathbf{p}(x)$. $\mathbf{H}'$ is given by

$$\mathbf{H}' = \begin{bmatrix} | & | & | & & | \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{e-1} \\ | & | & | & & | \end{bmatrix},$$

where $\begin{vmatrix} \\ \alpha^i \\ \\ \end{vmatrix}$ is the coefficient vector of $x^i \bmod \mathbf{p}(\mathbf{x})$. If $\mathbf{p}(x)$ is primitive, then $e = 2^b - 1$. This matrix design is indicated in Subsection 2.3.7. On the other hand, the companion matrix $\mathbf{T}$, defined by the primitive polynomial $\mathbf{p}(x)$ of degree b, and $\mathbf{T}^i$ (for $i = 0, 1, \ldots, 2^b - 2$) are expressed as in Eq. (5.2). Therefore the $\mathbf{H}$ matrix shown in Eq. (5.6) can be rewritten using these $\mathbf{T}$ and $\mathbf{T}^i$ as

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_b & \mathbf{I}_b & \mathbf{I}_b & \cdots & \mathbf{I}_b & \cdots & \mathbf{I}_b \\ \mathbf{I}_b & \mathbf{T}^b & \mathbf{T}^{2b} & \cdots & \mathbf{T}^{ib} & \cdots & \mathbf{T}^{(e-1)b} \end{bmatrix}.$$

This matrix can be reduced to *echelon canonical form* as

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_b & \mathbf{0} & \mathbf{h}_{0,2} & \mathbf{h}_{0,3} & \dots & \mathbf{h}_{0,i} & \dots & \mathbf{h}_{0,e-1} \\ \mathbf{0} & \mathbf{I}_b & \mathbf{h}_{1,2} & \mathbf{h}_{1,3} & \dots & \mathbf{h}_{1,i} & \dots & \mathbf{h}_{1,e-1} \end{bmatrix},$$

where $\mathbf{0}, \mathbf{I}_b, \mathbf{h}_{0,2}, \dots, \mathbf{h}_{0,i}$, and $\mathbf{h}_{1,i}, \dots, \mathbf{h}_{1,e-1}$ are all elements in $GF(2^b)$.

The systematic form of the matrix has the following properties for $j = 0, 1, 2, \dots, e-1$ [FUJI77a]:

Property 1

$$\mathbf{h}_{0,j} + \mathbf{h}_{1,j} = \mathbf{I}_b. \tag{5.7}$$

Property 2

$$\mathbf{H}_j = \begin{bmatrix} \mathbf{h}_{0,j} \\ \mathbf{h}_{1,j} \end{bmatrix} = \mathbf{H}_{j-1} + \begin{bmatrix} \mathbf{T}^{(j-1)b} \\ \mathbf{T}^{(j-1)b} \end{bmatrix}. \tag{5.8}$$

Property 3

$$\mathbf{h}_{0,2} = \mathbf{T}^b. \tag{5.9}$$

The $+$ expresses addition in $GF(2^b)$.

Example 5.3 [FUJI76]

The (21, 15) Burton code is defined by the generator polynomial $\mathbf{g}(x) = (x^3 + 1) \cdot \mathbf{p}(x)$, where $\mathbf{p}(x) = x^3 + x + 1$ is a primitive polynomial with exponent $e = 7$. Its $\mathbf{H}$ matrix in *canonical form* (i.e., in systematic form) is given below:

$$\mathbf{H} = \begin{bmatrix} | & | & | & | & & | \\ \beta^0 & \beta^1 & \beta^2 & \beta^3 & & \beta^{20} \\ | & | & | & | & & | \end{bmatrix} = \begin{array}{cccccc|ccccccccccccccc} 0&1&2&3&4&5&6&7&8&9&10&11&12&13&14&15&16&17&18&19&20 \\ \hline 1 & & & & & & 1 & & 1 & & 1 & 1 & & & 1 & 1 & 1 & & 1 & 1 & 1 \\ & 1 & & & & & 1 & 1 & 1 & 1 & 1 & & 1 & & 1 & & & 1 & 1 & & \\ & & 1 & & & & & 1 & 1 & 1 & 1 & 1 & & 1 & & 1 & & & 1 & 1 & \\ & & & 1 & & & & & 1 & 1 & 1 & 1 & 1 & & 1 & & 1 & & & 1 & 1 \\ & & & & 1 & & 1 & & 1 & 1 & & & 1 & 1 & 1 & & 1 & 1 & 1 & 1 & \\ & & & & & 1 & & 1 & & 1 & 1 & & & 1 & 1 & 1 & & 1 & 1 & 1 & 1 \end{array}.$$

As before, $\begin{matrix} | \\ \beta^i \\ | \end{matrix}$ is the coefficient vector of x^i mod $\mathbf{g}(x)$. Next we can divide this $\mathbf{H}$ matrix by 3×3 square matrices as follows:

$$\mathbf{H} = \left[\begin{array}{ccc|ccc|ccc|ccc|ccc|ccc|ccc} 1 & & & & & & 1 & & 1 & & 1 & 1 & & & 1 & 1 & 1 & & 1 & 1 & 1 \\ & 1 & & & & & 1 & 1 & 1 & 1 & 1 & & 1 & & 1 & & & 1 & 1 & & \\ & & 1 & & & & & 1 & 1 & 1 & 1 & 1 & & 1 & & 1 & & & 1 & 1 & \\ \hline & & & 1 & & & & & 1 & 1 & 1 & 1 & 1 & & 1 & & 1 & & & 1 & 1 \\ & & & & 1 & & 1 & & 1 & 1 & & & 1 & 1 & 1 & & 1 & 1 & 1 & 1 & \\ & & & & & 1 & & 1 & & 1 & 1 & & & 1 & 1 & 1 & & 1 & 1 & 1 & 1 \end{array}\right]. \tag{5.10}$$

On the other hand, $\mathbf{H}'$ can be given by

$$\mathbf{H}' = \begin{bmatrix} | & | & | & | & | & | & | \\ 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ | & | & | & | & | & | & | \end{bmatrix},$$

where α is a root of $\mathbf{p}(x)$. From the polynomial $\mathbf{p}(x)$ the following companion matrix and the addition table on $GF(2^3)$ can be derived:

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

$+$	$\mathbf{0}$	$\mathbf{I}$	$\mathbf{T}$	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{I}$	$\mathbf{T}$	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$
$\mathbf{I}$	$\mathbf{I}$	$\mathbf{0}$	$\mathbf{T}^3$	$\mathbf{T}^6$	$\mathbf{T}$	$\mathbf{T}^5$	$\mathbf{T}^4$	$\mathbf{T}^2$
$\mathbf{T}$	$\mathbf{T}$	$\mathbf{T}^3$	$\mathbf{0}$	$\mathbf{T}^4$	$\mathbf{I}$	$\mathbf{T}^2$	$\mathbf{T}^6$	$\mathbf{T}^5$
$\mathbf{T}^2$	$\mathbf{T}^2$	$\mathbf{T}^6$	$\mathbf{T}^4$	$\mathbf{0}$	$\mathbf{T}^5$	$\mathbf{T}$	$\mathbf{T}^3$	$\mathbf{I}$
$\mathbf{T}^3$	$\mathbf{T}^3$	$\mathbf{T}$	$\mathbf{I}$	$\mathbf{T}^5$	$\mathbf{0}$	$\mathbf{T}^6$	$\mathbf{T}^2$	$\mathbf{T}^4$
$\mathbf{T}^4$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^2$	$\mathbf{T}$	$\mathbf{T}^6$	$\mathbf{0}$	$\mathbf{I}$	$\mathbf{T}^3$
$\mathbf{T}^5$	$\mathbf{T}^5$	$\mathbf{T}^4$	$\mathbf{T}^6$	$\mathbf{T}^3$	$\mathbf{T}^2$	$\mathbf{I}$	$\mathbf{0}$	$\mathbf{T}$
$\mathbf{T}^6$	$\mathbf{T}^6$	$\mathbf{T}^2$	$\mathbf{T}^5$	$\mathbf{I}$	$\mathbf{T}^4$	$\mathbf{T}^3$	$\mathbf{T}$	$\mathbf{0}$

Hence the $\mathbf{H}$ matrix of this code can be expressed as

$$\mathbf{H} = \left[\begin{array}{ccccccc} \mathbf{I}_3 & \mathbf{I}_3 & \mathbf{I}_3 & \mathbf{I}_3 & \mathbf{I}_3 & \mathbf{I}_3 & \mathbf{I}_3 \\ & \mathbf{H}' & & \mathbf{H}' & & \mathbf{H}' & \end{array}\right]$$

$$= \left[\begin{array}{ccc:ccc:ccc:ccc:ccc:ccc:ccc} 1 & & & 1 & & & 1 & & & 1 & & & 1 & & & 1 & & & 1 & & \\ & 1 & & & 1 & & & 1 & & & 1 & & & 1 & & & 1 & & & 1 & \\ & & 1 & & & 1 & & & 1 & & & 1 & & & 1 & & & 1 & & & 1 \\ \hline | & | & | & | & | & | & | & | & | & | & | & | & | & | & | & | & | & | & | \\ 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \\ | & | & | & | & | & | & | & | & | & | & | & | & | & | & | & | & | & | \end{array}\right]$$

$$= \left[\begin{array}{ccccccc} \mathbf{I}_3 & \mathbf{I}_3 & \mathbf{I}_3 & \mathbf{I}_3 & \mathbf{I}_3 & \mathbf{I}_3 & \mathbf{I}_3 \\ \mathbf{I}_3 & \mathbf{T}^3 & \mathbf{T}^6 & \mathbf{T}^2 & \mathbf{T}^5 & \mathbf{T} & \mathbf{T}^4 \end{array}\right]$$

$$= \left[\begin{array}{ccccccc} \mathbf{I}_3 & \mathbf{0} & \mathbf{T}^3 & \mathbf{T}^4 & \mathbf{T} & \mathbf{T}^6 & \mathbf{T}^5 \\ \mathbf{0} & \mathbf{I}_3 & \mathbf{T} & \mathbf{T}^5 & \mathbf{T}^3 & \mathbf{T}^2 & \mathbf{T}^4 \end{array}\right].$$

The resultant $\mathbf{H}$ matrix in an echelon canonical form is equivalent to the matrix shown in Eq. (5.10). It is clear that this matrix satisfies the properties shown in Eqs. (5.7), (5.8), and (5.9).

The interesting properties of the Burton code, especially property 1 in Eq. (5.7), can make this an odd-weight-column S*b*EC code. Actually every column vector of the $\mathbf{H}$ matrix shown in Eq. (5.10) is of odd weight, and property 1 in Eq. (5.7) satisfies Definition 3.4 of the odd-weight-column code (see Section 3.2).

Property 2 in Eq. (5.8) and property 3 in Eq. (5.9) express the sequential structure of this code. These properties are not particularly important when parallel decoding is employed, as required, for high-speed memories.

Next we consider the double-byte error detection capability of these 2-redundant SbEC codes.

Theorem 5.1 [FUJI77b] *The double-byte error detection capability, P_d, of the 2-redundant $((k+2)b, kb)$ SbEC code is given by*

$$P_d = 1 - \frac{k}{2^b - 1}.$$

Proof The probability P_d is calculated by counting the fraction of double-byte errors that are detected by this code. Double-byte errors, say E_1 and E_2, generate a syndrome that may equal the syndrome caused by a single-byte error, say E_3. This will result in a miscorrection as stated before. We analyze the cases occurred by these byte errors of E_1, E_2, and E_3, in the following:

Case 1. E_1, E_2, and E_3 are all in the information-bit part.

Case 2. E_1 and E_2 are in the information-bit part, and E_3 in the check-bit part.

Case 3. E_1 is in the information-bit part, and E_2 and E_3 in the check-bit part.

Case 4. E_1, E_2, and E_3 are all in the check-bit part. (The miscorrection will be harmless if it occurs.)

For each case the number of miscorrections can be counted as follows:

$$\begin{aligned}
&\text{Case 1:} \quad 3(2^b - 1)\binom{k}{3},\\
&\text{Case 2:} \quad 2 \times 3(2^b - 1)\binom{k}{2},\\
&\text{Case 3:} \quad 3(2^b - 1)\binom{k}{1},\\
&\text{Case 4:} \quad 0.
\end{aligned}$$

Therefore the detection ability, P_d is as follows:

$$P_d = 1 - \frac{3(2^b - 1)\left\{\binom{k}{3} + 2\binom{k}{2} + \binom{k}{1}\right\}}{\binom{k+2}{2}\sum_{i,j=1}^{b}\binom{b}{i}\cdot\binom{b}{j}}.$$

By simple algebra, the equation above reduces to $1 - \{k/(2^b - 1)\}$, which completes the proof. Q.E.D.

Example 5.4

The 2-redundant $(80, 64)$ S8EC code has the following double-byte error detection probability $P_d = 1 - 8/(2^8 - 1) = 0.9686$. That is, 96.86% of all double-byte errors are detected, and therefore it is "very nearly" an S8EC-D8ED code.

Theorem 5.1 gives us an important result that for a large value of b (e.g., $b = 8$ and 16 bits), the 2-redundant SbEC code is very nearly an SbEC-DbED code. Also this result is practical and suitable for high-speed memories because the information-bit length of these memories is rather short [ARLA84]. Thus the class of SbEC-DbED codes discussed later may not be all that necessary for the case where b is large.

Here, we consider a generalized class of Burton codes that can be constructed by using a generator polynomial $\mathbf{g}(x) = (x^b + 1)\mathbf{p}(x)$, where the degree of $\mathbf{p}(x)$ is l. It is important that if l is equal to b, the codes defined by the polynomial $\mathbf{g}(x)$ are equal to the Burton codes. If l is greater than b, the codes can also correct all single-byte errors [VARA83]. If $\mathbf{p}(x)$ is a primitive polynomial of degree l, then the code length N (bits) and the check-bit length R can be expressed as

$$N = b(2^l - 1),$$
$$R = l + b.$$

If l is less than b, the codes can correct single-bit errors and detect both double-bit errors and single b-bit burst errors [VARA83]. This will be shown in Section 6.1.

5.1.3 Odd-Weight-Column Codes — Fujiwara Codes —

From the property of the Burton code Fujiwara derives a new class of SbEC codes that has an odd-weight-column characteristic [FUJI77b, FUJI78]. This generalized code over $GF(2^b)$ includes an excellent odd-weight-column SEC-DED code, especially for $b = 1$.

In an odd-weight-column matrix code over $GF(2^b)$ (see Definition 3.4 in Section 3.2), no two columns are identical and no column is all zero or a multiple of another column. The last is true because if a column vector h_i is a multiple of h_j (i.e., $h_i = \beta \cdot h_j$, where $\beta \in GF(2^b)$, and $\beta \neq 0$, $\beta \neq 1$), then sum of the elements of h_i equals β, contradicting the odd-weight-column property. Therefore the columns h_i and h_j are a linearly independent pair and the code is of distance-3 or higher. And we have proved the following.

Theorem 5.2 *An odd-weight-column matrix code over $GF(2^b)$ is an SbEC code.*

Lemma 5.1 *There exists exactly $2^{b(r-1)}$ odd-weight-column vectors, each having r elements over $GF(2^b)$.*

Proof Let h_i be the i-th odd-weight-column vector over $GF(2^b)$ in the parity-check matrix of this code. Also let the sum of arbitrary $r - 1$ elements in h_i be $\gamma \in GF(2^b)$. Then the remaining one element can be determined as $\gamma + \mathbf{I}$, where $\mathbf{I}$ is an identity element in $GF(2^b)$; that is, the remaining one element in h_i is uniquely determined from the other $r - 1$ elements. Since each element can have any one of 2^b values, there are exactly $2^{b(r-1)}$ such column vectors. Q.E.D.

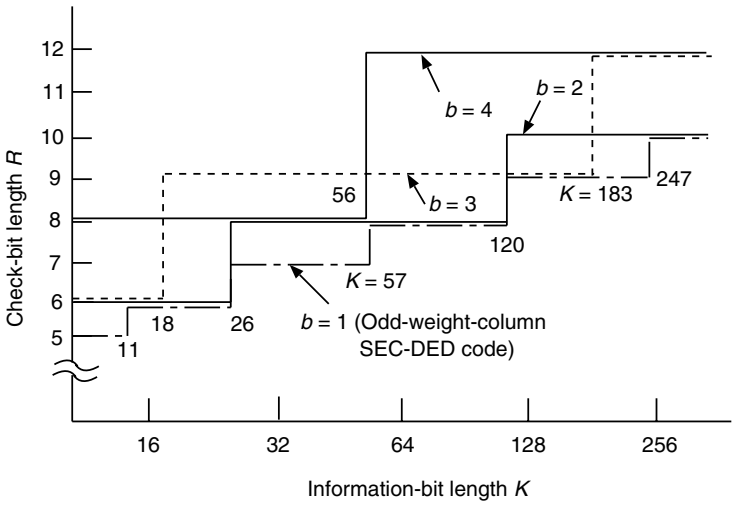


Figure 5.2 *Comparison of check-bit lengths and information-bit lengths of the Fujiwara SbEC codes.* Source: *[FUJI78, 81]. © 1978 IECE Japan.*

This gives the maximal code length N_F (bits) of the Fujiwara code as

$$N_F = b \cdot n = b \cdot 2^{b(r-1)}. \tag{5.11}$$

Compared to the Hamming-type SbEC code, this code is shorter in code length. From Eqs. (5.4) and (5.11) the ratio N_F/N_H can be written as follows:

$$\begin{aligned} \frac{N_F}{N_H} &= 1 - \frac{2^{b(r-1)} - 1}{2^{br} - 1} \\ &\approx 1 - \frac{1}{2^b} \qquad \text{for } r \cdot b \gg 1. \end{aligned}$$

This code is equivalent to the popular odd-weight-column SEC-DED code (see Section 4.1.1), especially for $b = 1$. Therefore this type of codes can be said to include the odd-weight-column SEC-DED code as a special case of $b = 1$. Figure 5.2 shows the relation between the information-bit length K and the check-bit length R of this code for $b = 1, 2, 3$, and 4 bits.

Example 5.5

Consider a (72, 64) S2EC code having parameters $b = 2$ and $r = 4$. The companion matrix $\mathbf{T}$ is determined by the polynomial $\mathbf{g}(x) = x^2 + x + 1$. Its $\mathbf{H}$ matrix is given as follows:

$$\left[\begin{array}{cccccccccccccccccccccccccccccccccc}
\mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T} & \mathbf{T} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T} & \mathbf{T}^2 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{T}^2 & \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{I} & \mathbf{T}^2 & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\
\mathbf{I} & \mathbf{I} & \mathbf{T} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{I} & \mathbf{T}^2 & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T} & \mathbf{T} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T} & \mathbf{T}^2 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{T}^2 & \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\
\mathbf{T}^2 & \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{I} & \mathbf{T}^2 & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T} & \mathbf{T} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T} & \mathbf{T}^2 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\
\mathbf{T} & \mathbf{T}^2 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{T}^2 & \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{I} & \mathbf{T}^2 & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T} & \mathbf{T} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I}
\end{array}\right],$$

$$\mathbf{0} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{T} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \mathbf{T}^2 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}. \tag{5.12}$$

The decoding procedure of this code is presented as follows: Let the i-th column vector in the **H** matrix be $(h_{0,i}\ h_{1,i}\ \ldots\ h_{r-1,i})^T$, where $h_{m,i} \in GF(2^b)$, $m = 0, 1, \ldots, r-1$, and the syndrome due to error in the i-th position be $(S_0\ S_1\ \ldots\ S_{r-1})$. Then an error E_i in symbol i yields a syndrome that is equal to

$$S_m = E_i \cdot h_{m,i} \qquad \text{for } m = 0, 1, \ldots, r-1.$$

Clearly, for odd-weight-column codes the sum of syndrome components must equal E_i, that is,

$$E_i = \sum_{i=0}^{r-1} S_i, \qquad \sum : \text{addition in } GF(2^b).$$

The error byte pointer g_i is given by substituting E_i, for example, by $S_0 \cdot h_{0,i}^{-1}$,

$$g_i = \bigcap_{m=1}^{r-1} [S_m = S_0 \cdot h_{0,i}^{-1} \cdot h_{m,i}].$$

From these operations, the corrected i-th byte data $\widehat{D}_i$ can be obtained as

$$\widehat{D}_i = D_i + g_i \cdot E_i \qquad \text{for } i = 0, 1, \ldots k-1.$$

In the equation above, $+$ expresses addition in $GF(2^b)$. Figure 5.3 shows the decoding circuit of the odd-weight-column SbEC code.

Another important feature of this odd-weight-column SbEC code exists in its error detection capability for certain double-byte errors.

Theorem 5.3 *Odd-weight-column SbEC codes can detect double-byte errors in positions i and j provided that their error values E_i and E_j are equal.*

Proof The condition for detecting double-byte errors is given by

$$E_i \cdot h_i^T + E_j \cdot h_j^T \neq E_k \cdot h_k^T \tag{5.13}$$

for all nonzero E_i, E_j, and E_k existing in three distinct byte positions i, j, and k, respectively. The corresponding column vectors of the **H** matrix are h_i, h_j, and h_k.

Since the code is an SbEC code, $E_i \cdot h_i^T + E_j \cdot h_j^T \neq 0$. If $E_i = E_j$, the sum of all elements in the vector $E_i \cdot h_i^T + E_j \cdot h_j^T$ is equal to the error value $E_i + E_j$, and hence the sum equals zero. However, the sum of the elements in the vector $E_k \cdot h_k^T$ is equal to the error value E_k and does not equal zero. Therefore Eq. (5.13) holds for $E_i = E_j$, and the theorem is proved. Q.E.D.

Theorem 5.3 guarantees the partial double-byte error detection of the codes. In particular, all bit errors in two bytes, meaning $2b$-bit errors over any two bytes, can be detected.

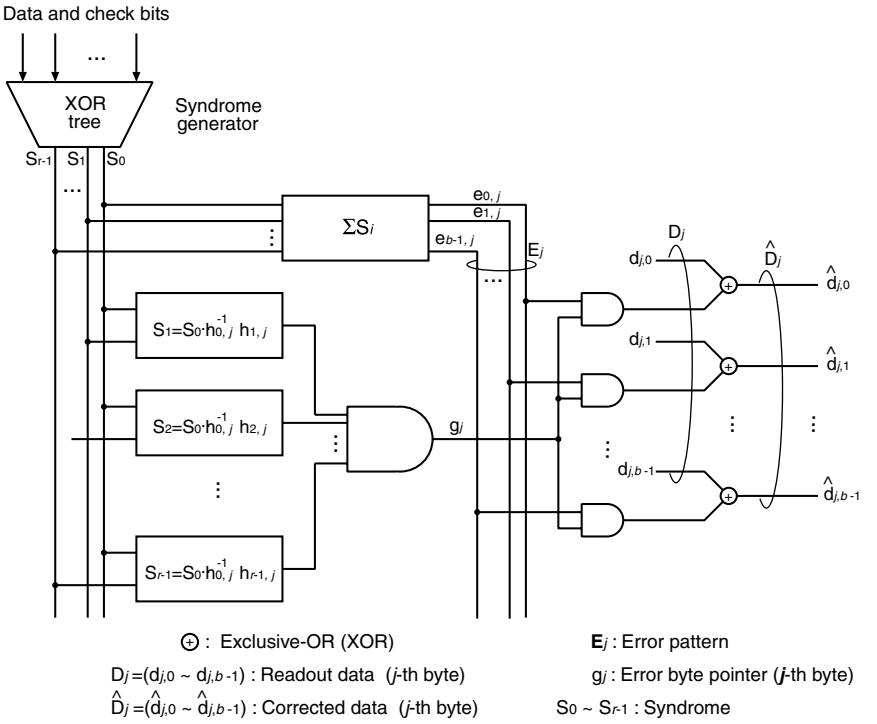


Figure 5.3 *Error correction circuit for byte j for Fujiwara code.* Source: *[FUJI78, 81]. © 1978 IECE Japan.*

In addition, from the odd-weight-column characteristic of this code, the following relations are valid:

1. The even number of bit errors over two bytes cannot be miscorrected to the odd number of bit errors over single bytes.
2. The odd number of bit errors over two bytes cannot be miscorrected to the even number of bit errors over single bytes.

It is important that the S*b*EC codes minimize the probability of miscorrection of the double-byte errors. Table 5.1 compares the miscorrection probability of the typical (72, 64) S2EC codes. According to these results the odd-weight-column S2EC codes have better error detection capability for double-byte errors than the non–odd-weight-column S2EC codes. For more details, refer to [FUJI78].

TABLE 5.1 Miscorrection Probabilities of (72, 64) S2EC Codes

Codes	Probability of miscorrected double 2-bit byte errors (%)	Probability of miscorrected 2-bit errors over double bytes (%)
Hamming-type (72, 64) S2EC code shown in Eq. (5.3)	45.9	48.3
Odd-weight-column (72, 64) S2EC code shown in Eq. (5.12)	33.6	25.2

Rotational Fujiwara Codes We can also derive rotational odd-weight-column S*b*EC codes [FUJI77b, FUJI78]. Before obtaining the code length of this code, we need the following *modified Möbius inversion formula*.

Theorem 5.4 [FUJI78] *If the relation*

$$f(r) = \sum_{(r/m):odd} g(m) \tag{5.14}$$

is valid between two functions, $f(r)$ and $g(r)$, for any positive integer r, then $g(r)$ can be expressed by $f(r)$ as

$$g(r) = \sum_{\substack{m|r \\ m:odd}} \mu(m) \cdot f\left(\frac{r}{m}\right). \tag{5.15}$$

Here $\mu(m)$ is the Möbius function defined in Section 3.5, and $\sum_{\substack{m|r \\ m:odd}}$ means the summation over all odd m that divides r.

Proof Because Eq. (5.14) is valid for any positive integer, the following relation should hold for any divisor m of r:

$$f\left(\frac{r}{m}\right) = \sum_{(r/m)/c:odd} g(c).$$

This is substituted into the right-hand side of Eq. (5.14). The group of all combinations of m and c that satisfies $\begin{matrix} m|r \\ m:odd \end{matrix}$ and $(r/m)/c$: *odd* is equivalent to the group of all combinations of m and c that satisfies $c|r$ and $\begin{matrix} m|(r/c) \\ (r/c):odd \end{matrix}$. Then the following equation is valid:

$$\sum_{\substack{m|r \\ m:odd}} \mu(m) \cdot f\left(\frac{r}{m}\right) = \sum_{\substack{m|r \\ m:odd}} \mu(m) \cdot \sum_{(r/m)/c:odd} g(c) = \sum_{c|r} g(c) \cdot \sum_{\substack{m|(r/c) \\ (r/c):odd}} \mu(m). \tag{5.16}$$

The property of a Möbius function says that

$$\sum_{m|n} \mu(m) = \begin{cases} 1, & n = 1, \\ 0, & n > 1. \end{cases}$$

As a result the right-hand side of Eq. (5.16) equals $g(r)$, and the relation in Eq. (5.15) is proved. Q.E.D.

The discussion that follows is related to the rotational error control codes (see Section 3.5).

The **H** matrix of the rotational odd-weight-column codes has r rows and n columns. The column vector that has r elements is assumed to have some *nondegenerate cyclic equivalence classes*, each of which has m elements of $GF(2^b)$. The summation of m elements of the class is equal to **I**. Moreover the whole summation of r elements of the column vector should be equal to **I**. Thus r divided by m should be an odd integer.

Let $n(r, b)$ be a total number of nondegenerate cyclic equivalence classes that have m elements of $GF(2^b)$. Then the following equation is valid:

$$2^{b(r-1)} = \sum_{(r/m):odd} m \cdot n(m, b).$$

From the *modified Möbius inversion formula* of Theorem 5.4, $n(r, b)$ can be easily obtained as

$$n(r, b) = \frac{1}{r} \sum_{\substack{m|r \\ m:odd}} \mu(m) \cdot 2^{b(r/m-1)}.$$

Here $n(r, b)$ means the length of each $\mathbf{H}_i$ of the rotational **H** matrix. Thus the code length (in bytes) of the rotational odd-weight-column SbEC code is expressed as

$$n = r \cdot n(r, b) = \sum_{\substack{m|r \\ m:odd}} \mu(m) \cdot 2^{b(r/m-1)}.$$

Table 5.2 shows the code length n (bytes) of this code. Note that there exists no significant difference between the code lengths of the rotational and the nonrotational codes. And for

TABLE 5.2 Code Lengths (in Bytes) of Rotational Fujiwara S*b*EC Codes Compared to the Nonrotational Codes

r \ b	1	2	3	4	6	8
2	$\frac{2}{2}$	$\frac{4}{4}$	$\frac{8}{8}$	$\frac{16}{16}$	$\frac{64}{64}$	$\frac{256}{256}$
3	$\frac{3}{4}$	$\frac{15}{16}$	$\frac{63}{64}$	$\frac{255}{256}$	$\frac{4,095}{4,096}$	$\frac{65,535}{65,536}$
4	$\frac{8}{8}$	$\frac{64}{64}$	$\frac{512}{512}$	$\frac{4,096}{4,096}$	.	.
5	$\frac{15}{16}$	$\frac{255}{256}$	$\frac{4,095}{4,096}$	$\frac{65,535}{65,536}$	.	.
6	$\frac{30}{32}$	$\frac{1,020}{1,024}$	$\frac{32,760}{32,768}$	.	.	.
7	$\frac{63}{64}$	$\frac{4,095}{4,096}$	.	.	.	.
8	$\frac{128}{128}$	$\frac{16,384}{16,384}$	.	.	.	.

Note: The codes with $b = 1$ are equal to the rotational odd-weight-column SEC-DED codes with code length in bits $n = \sum_{\substack{m|r \\ m:odd}} \mu(m) \cdot 2^{(r/m-1)}$. n/n': The numerator n gives the rotational code length in bytes and the denominator n' the nonrotational code length in bytes.

$$\mathbf{H}_0 = \begin{bmatrix} \mathbf{T} & \mathbf{T}^2 & \mathbf{I} & \mathbf{T} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{T} & \mathbf{I} \\ \mathbf{T} & \mathbf{T}^2 & \mathbf{T} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T} & \mathbf{0} & \mathbf{0} \\ \mathbf{T} & \mathbf{T}^2 & \mathbf{T} & \mathbf{T} & \mathbf{T}^2 & \mathbf{0} & \mathbf{0} & \mathbf{T}^2 & \mathbf{0} \\ \mathbf{T}^2 & \mathbf{T} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \quad \mathbf{H}_0 = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T} & \mathbf{T} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{I} \\ \mathbf{I} & \mathbf{I} & \mathbf{T} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{I} & \mathbf{T}^2 & \mathbf{0} \\ \mathbf{T}^2 & \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{I} & \mathbf{0} \\ \mathbf{T} & \mathbf{T}^2 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

$$\mathbf{T} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \quad \mathbf{T}^2 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \quad \mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{0} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{I} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} \end{bmatrix}$$

$$\mathbf{H} = \left[\mathbf{H}_0 \,\middle|\, \mathbf{R} \cdot \mathbf{H}_0 \,\middle|\, \mathbf{R}^2 \cdot \mathbf{H}_0 \,\middle|\, \mathbf{R}^3 \cdot \mathbf{H}_0 \right]$$

Figure 5.4 *Two examples of rotational Fujiwara (72, 64) S2EC codes.* Source: *[FUJI78]. © 1978 IECE Japan.*

$b = 1$, this is equal to the code length of the rotational odd-weight-column SEC-DED codes. Figure 5.4 shows two examples of the rotational Fujiwara (72, 64) S2EC codes.

5.1.4 Maximal Codes — Hong-Patel Codes —

We study here a class of maximal SbEC codes over $GF(2^b)$ called *Hong-Patel codes* [HONG72]. A byte is not equated to a symbol from $GF(2^b)$ but instead is treated as a convenient cluster of b individual bits. The number of check bits may or may not be a multiple of b, and sometimes it may be arbitrary. Hong-Patel codes contain subclasses that are equivalent to all single-symbol error correcting codes over $GF(2^b)$, including the binary Hamming codes. Furthermore these codes are easily implementable and expandable, and they are either *perfect* or *maximal*. Hence they are called the *general class of maximal codes* [HONG72]. In this context, a *maximal code* is defined such that no longer code with the same error-correcting capability for a given check-bit length exists.

Given a check-bit length R and a byte-length b, consider the matrix $\mathbf{H}_{R,b}$ shown in Figure 5.5, where $R \geq 2b$, α is a primitive element in $GF(2^{R-b})$; that is, it is a root of a primitive polynomial $\mathbf{g}(x)$ of degree $R - b$, and hence $\begin{vmatrix} \\ \alpha^i \\ \\ \end{vmatrix}$ is the coefficient vector of $x^i \bmod \mathbf{g}(x)$.

Lemma 5.2 *The code given by the following* $\boldsymbol{H}$ *matrix corrects all single-byte errors:*

$$\boldsymbol{H} = [\boldsymbol{H}_{R,b} \,|\, \boldsymbol{I}_R].$$

Proof The information bits can be grouped as $2^{R-b} - 1$ bytes, $D_0, D_1, \ldots, D_{2^{R-b}-2}$. The check bits are similarly grouped as check bytes $C_0\ C_1\ \ldots\ C_{r-1}$, where $r = \lceil R/b \rceil$ and C_{r-1} is the last check byte, which may be of a length less than b. For the codeword $W = [D_0\ D_1\ \ldots\ D_{2^{R-b}-2}\ C_0\ C_1\ \ldots\ C_{r-1}]$, we have $W \cdot \mathbf{H}^T = 0$. The erroneous word W' then produces a syndrome S given by

$$S = W' \cdot \mathbf{H}^T = [S_0\ S_1\ \ldots\ S_{r-1}].$$

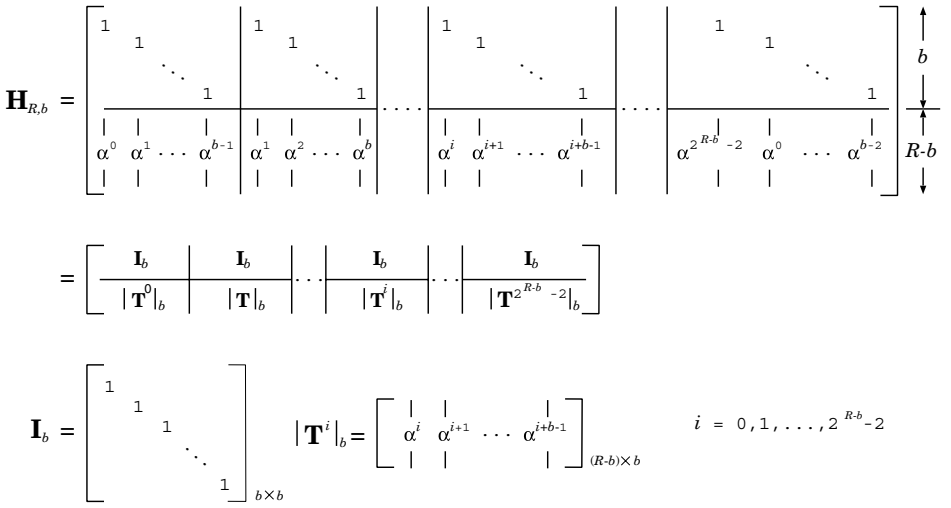


Figure 5.5 *Matrix $\mathbf{H}_{R,b}$ of Hong-Patel codes.*

The syndrome consists of r bytes called syndrome bytes. With the considerations above, we can now proceed to prove Lemma 5.2.

First, any error byte in the information portion, say, error pattern $E_i \neq 0$ in the i-th byte, gives the following syndrome:

$$S_0 = E_i$$

and

$$[S_1, S_2, \ldots, S_{r-1}] = E_i \cdot |\mathbf{T}^i|_b^T.$$

Here note that $|\mathbf{T}^i|_b$ is an $(R-b) \times b$ matrix $(R \geq 2b)$ after deleting the last $R-b$ columns from the original $(R-b) \times (R-b)$ matrix $\mathbf{T}^i$, which will be defined in Definition 6.4 as a *slimmed matrix*. Clearly, $S_0 = E_i \neq 0$ and $[S_1, S_2, \ldots, S_{r-1}] \neq 0$. The error byte in the check portion, however, gives the following syndromes: Let $E_j \neq 0$ in the j-th check byte. Then

$$S_\lambda = 0, \quad \lambda \neq j,$$
$$S_j = E_j \neq 0.$$

Hence an error in the information portion must give at least two types of nonzero syndromes, and an error in the check portion gives only one nonzero syndrome byte. Distinct errors in the check portion obviously yield distinct syndromes. Now suppose that byte errors $E_i \neq 0$ and $E_j \neq 0$ in the i-th and the j-th $(i \neq j)$ information bytes generate identical syndromes. Then we have

$$E_i = E_j \quad \text{and} \quad E_i \cdot |\mathbf{T}^i|_b^T = E_j \cdot |\mathbf{T}^j|_b^T.$$

Since $\mathbf{T}^i \neq \mathbf{T}^j$ for $i \neq j$, this cannot occur. Therefore errors in information bytes have distinct syndromes and hence are correctable. Q.E.D.

Lemma 5.3 *The code described by the following* $\boldsymbol{H}$ *matrix corrects all single-byte errors:*

$$\boldsymbol{H} = \left[\boldsymbol{H}_{R,b} \left| \begin{matrix} \boldsymbol{0}_b\ \boldsymbol{0}_b \cdots \boldsymbol{0}_b \\ \hline \boldsymbol{H}_{R-b,b} \end{matrix} \right| \boldsymbol{I}_R \right], \qquad R \geq 3b, \tag{5.17}$$

where $\boldsymbol{0}_b$ *is a* $b \times b$ *zero matrix.*

Proof Let the information portions corresponding to $\mathbf{H}_{R,b}$ and $\mathbf{H}_{R-b,b}$ be called the first and the second partition of information bytes. An error byte in the first partition yields $S_0 \neq 0$ and at least one more nonzero syndrome byte. An error byte in the second partition yields $S_0 = 0$, $S_1 \neq 0$, and at least one more nonzero syndrome byte. This is because $\mathbf{H}_{R-b,b}$ itself is a single-byte error correcting code having $R - b$ check bits. An error byte in the check portion yields one and only one nonzero syndrome byte. Distinct byte errors in the same partition yield distinct syndromes due to Lemma 5.2. Q.E.D.

Lemma 5.3 suggests an iterative concatenation of partitions as defined in Eq. (5.17), maintaining the single-byte error correcting capability. From the two lemmas a new class of code can be defined as the code given by the following $\mathbf{H}$ matrix:

$$\begin{aligned} \mathbf{H} &= \left[\mathbf{H}_{R,b} \left| \begin{matrix} \mathbf{0}_b\ \mathbf{0}_b \cdots \mathbf{0}_b \\ \hline \mathbf{H}_{(R-b),b} \end{matrix} \right| \begin{matrix} \mathbf{0}_b\ \mathbf{0}_b \cdots \mathbf{0}_b \\ \hline \mathbf{0}_b\ \mathbf{0}_b \cdots \mathbf{0}_b \\ \hline \mathbf{H}_{(R-2b),b} \end{matrix} \left| \cdots \right| \begin{matrix} \mathbf{0}_b\ \mathbf{0}_b \cdots \mathbf{0}_b \\ \hline \mathbf{0}_b\ \mathbf{0}_b \cdots \mathbf{0}_b \\ \hline \vdots \\ \hline \mathbf{H}_{(2b+c),b} \end{matrix} \right| \mathbf{I}_R \right] \\ &= \left[\mathbf{P}_0 \left| \mathbf{P}_1 \right| \mathbf{P}_2 \left| \cdots \right| \mathbf{P}_{r-2} \left| \mathbf{I}_R \right. \right]. \end{aligned} \tag{5.18}$$

The second form shown above is to define $r - 1$ partitions for the information portion. Each partition $\mathbf{P}_j$ contains $b \cdot (2^{(r-j-1)b+c} - 1)$ columns.

Theorem 5.5 *The code defined by the* $\boldsymbol{H}$ *matrix of Eq. (5.18) corrects all single-byte errors.*

Proof Any two distinct errors within a partition or within the check portion yield distinct syndromes due to Lemma 5.3. A single error $E \neq 0$ in the i-th byte of partition $\mathbf{P}_j$ yields the syndrome

$$\begin{aligned} S_0 = S_1 = \cdots = S_{j-1} &= 0, \\ S_j &= E, \\ [S_{j+1}\ \ldots\ S_{r-1}] &= E \cdot |\mathbf{T}^i|_b^T \neq 0, \end{aligned}$$

which is distinct from the syndrome of any single-byte error in another partition or in the check portion. Q.E.D.

A few remarks about the structure of this code follow.

1. When $b = 1$, the code length n (bits) becomes

$$\begin{aligned} n &= \sum_{j=1}^{r-1}(2^{(r-j)} - 1) + r = \sum_{i=1}^{r-1} 2^i + 1 \\ &= 2^r - 1, \end{aligned} \tag{5.19}$$

which is the code length of the Hamming SEC code. Therefore this new code defines an alternative structure for a single-bit error correcting Hamming code.

2. The structure of each partition $\mathbf{H}_{(R-jb),b}$ resembles very closely that of a Fire code [FIRE59] shown in Subsection 2.3.7. In fact, when b and $2^{R-b} - 1$ are relatively prime, the Fire code given by the following $\mathbf{H}$ matrix, $\mathbf{H}'_{R,b}$, has the same properties as $\mathbf{H}_{R,b}$:

$$\mathbf{H}'_{R,b} = \left[\begin{array}{cccc} \mathbf{I}_b & \mathbf{I}_b & \cdots & \mathbf{I}_b \\ \hline \mathbf{H}' & \mathbf{H}' & \cdots & \mathbf{H}' \end{array}\right],$$

where $\mathbf{H}'$ denotes the binary Hamming code generated by a primitive polynomial $\mathbf{g}(x)$ of degree $R - b$. There are b repetitions of the $\mathbf{H}'$ in $\mathbf{H}'_{R,b}$.

3. The code construction reveals a systematic and regular method of code concatenation whereby the code length is increased in an iterative fashion.

We turn next to the length of this code. We saw in Eq. (5.19) that this is a *perfect code* when $b = 1$. A code is called *perfect* if all possible 2^R syndromes are used to correct 2^R different error patterns.

In this code, how the check bits are divided into bytes (in case b does not divide R exactly) gives rise to the following two classes of codes MC_1 and MC_2, respectively:

$$\mathbf{I}_R = \left[\begin{array}{cccc} \mathbf{I}_b & & & \\ & \mathbf{I}_b & & \\ & & \ddots & \\ & & & \mathbf{I}_b \\ & & & \mathbf{I}_{b+c} \end{array}\right] \begin{array}{l} \Big\} \\ \Big\} \end{array} \begin{array}{l} (r-1)\ \mathbf{I}_b \text{ matrices}, \\ 1\ \mathbf{I}_{b+c} \text{ matrix}, \end{array} \tag{5.20}$$

$$\mathbf{I}_R = \left[\begin{array}{cccc} \mathbf{I}_b & & & \\ & \mathbf{I}_b & & \\ & & \ddots & \\ & & & \mathbf{I}_b \\ & & & \mathbf{I}_c \end{array}\right] \begin{array}{l} \Big\} \\ \Big\} \end{array} \begin{array}{l} r\ \mathbf{I}_b \text{ matrices}, \\ 1\ \mathbf{I}_c \text{ matrix}. \end{array} \tag{5.21}$$

In general, the check-bit length $R = r \cdot b + c$, where $0 \leq c < b$. The leftover c check bits, if any, may form a special check byte. Another way is to form $r - 1$ regular size check bytes and allow a special check byte of length $b + c$.

Here MC_1 is defined as the code given by the **H** matrix of Eq. (5.18), where the check portion $\mathbf{I}_R$ is divided into bytes according to Eq. (5.20). MC_2 is given by the same matrix except that the check portion is divided into bytes according to Eq. (5.21).

Theorem 5.6 *MC_1 is a perfect code.*

Proof Given R, define M_1 to be the number of distinct error patterns that MC_1 can correct. From Eqs. (5.18) and (5.20) we see that for $R = r \cdot b + c \geq 2b$,

$$\begin{aligned} M_1 &= \sum_{j=1}^{r-1}(2^b - 1)(2^{(r-j)b+c} - 1) + (r-1)(2^b - 1) + (2^{b+c} - 1) + 1 \\ &= (2^b - 1)\sum_{i=1}^{r-1} 2^{ib+c} + 2^{b+c} \\ &= 2^{b+c}(2^b - 1)\frac{2^{(r-1)b} - 1}{2^b - 1} + 2^{b+c} \\ &= 2^{rb+c} = 2^R. \end{aligned}$$

Q.E.D.

As for MC_2, it is perfect only when $c = 0$, and it is maximal whenever $c = 1$ for $R = r \cdot b + c$. For the special case of $c = 0$ and 1, the code length (in bits) of the MC_2 can be expressed as follows [HONG72]:

$$N = b\frac{2^R - 1 - 2^b(2^c - 1)}{2^b - 1}.$$

Note that whenever b divides the given number of check bits R, MC_1, and MC_2 are exactly the same, $M_1 = M_2 = 2^R$ (M_2: number of distinct error patterns that MC_2 can correct), and the code length becomes equal to that of the Hamming-type code, expressed as Eq. (5.4).

The number of information bits, K, is the same in both MC_1 and MC_2.

$$\begin{aligned} K &= b \cdot \left(\sum_{j=1}^{r-1}(2^{(r-j)b+c} - 1)\right) \\ &= b \cdot \left(\frac{2^R - 2^{b+c}}{2^b - 1} - (r-1)\right) \\ &= b\frac{2^R - 1 - 2^b(2^c - 1)}{2^b - 1} - R + c. \end{aligned}$$

Example 5.6

Consider a Hong-Patel code with $b = 2$ bits and $K = 76$ bits. This can have check-bit length $R = 7$. First, for $R - b = 5$, obtain the primitive element α_0 of the primitive

polynomial $\mathbf{g}(x) = x^5 + x^2 + 1$ in $GF(2^{R-b}) = GF(2^5)$. From this information we can construct $\mathbf{H}_{7,2}$ as

$$\mathbf{H}_{7,2} = \left[\begin{array}{ccccc} \mathbf{I}_2 & \mathbf{I}_2 & \mathbf{I}_2 & \cdots & \mathbf{I}_2 \\ \hline \mathbf{A}_{0,0} & \mathbf{A}_{0,1} & \mathbf{A}_{0,2} & \cdots & \mathbf{A}_{0,30} \end{array}\right],$$

where

$$\begin{array}{l} \mathbf{A}_{0,0} = [I \ \alpha_0] \\ \mathbf{A}_{0,1} = [\alpha_0 \ \alpha_0^2] \\ \mathbf{A}_{0,2} = [\alpha_0^2 \ \alpha_0^3] \\ \quad \vdots \\ \mathbf{A}_{0,30} = [\alpha_0^{30} \ I] \end{array} \quad I = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \alpha_0 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \alpha_0^2 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \ldots, \quad \alpha_0^{30} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

Next, for $R - 2b = 3$, we obtain the primitive element α_1 of the primitive polynomial $\mathbf{g}(x) = x^3 + x + 1$ in $GF(2^{R-2b}) = GF(2^3)$. Similarly we obtain

$$\mathbf{H}_{5,2} = \left[\begin{array}{ccccc} \mathbf{I}_2 & \mathbf{I}_2 & \mathbf{I}_2 & \cdots & \mathbf{I}_2 \\ \hline \mathbf{A}_{1,0} & \mathbf{A}_{1,1} & \mathbf{A}_{1,2} & \cdots & \mathbf{A}_{1,6} \end{array}\right],$$

where

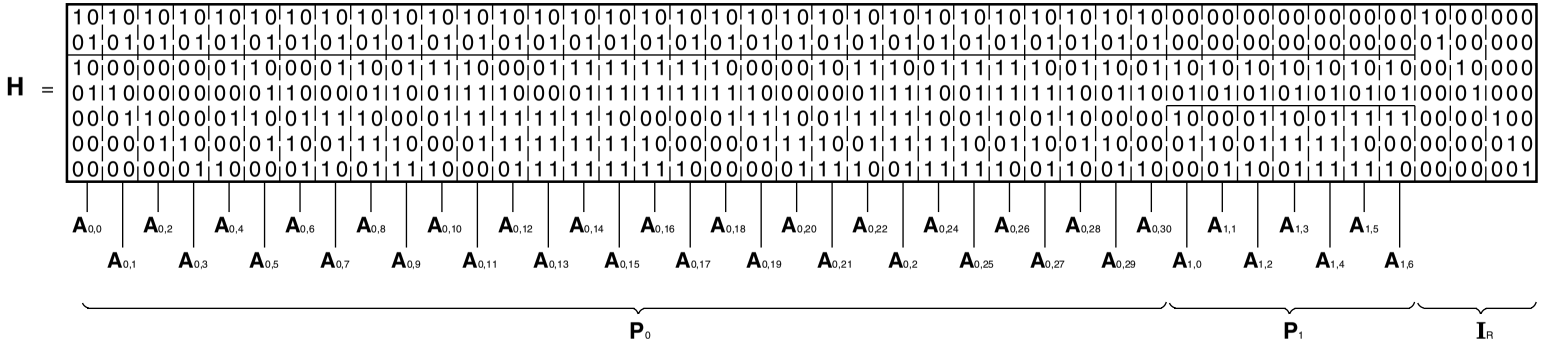
$$\begin{array}{l} \mathbf{A}_{1,0} = [I \ \alpha_1] \\ \mathbf{A}_{1,1} = [\alpha_1 \ \alpha_1^2] \\ \mathbf{A}_{1,2} = [\alpha_1^2 \ \alpha_1^3] \\ \quad \vdots \\ \mathbf{A}_{1,6} = [\alpha_1^6 \ I] \end{array} \quad I = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \alpha_1 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \alpha_1^2 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \ldots, \quad \alpha_1^6 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

This is how the (83, 76) S2EC code shown in Figure 5.6 is designed.

In this example the maximal code with byte length $b = 2$ bits and information-bit length $K = 64$ has one less check bit, compared to the equivalent Hamming-type code. For $b = 4$ bits, the code has 3 less check bits. Table 5.3 gives the maximal code lengths (in bits) of the SEC codes, the maximal S2EC codes, the maximal S4EC codes, and the SEC-DED codes for the given R ranging from 2 to 12 bits.

5.2 SINGLE-BYTE ERROR CORRECTING AND DOUBLE-BYTE ERROR DETECTING (S*b*EC-D*b*ED) CODES

This section deals with a class of single b-bit byte error correcting and double b-bit byte error detecting (SbEC-DbED) codes. From practical point of view, the SbEC codes have a problem such that detection of random double-bit errors spanning over two bytes is not guaranteed. That is, large-capacity semiconductor memory systems tend to have errors caused by two RAM chips failures, such as when hard errors occur in one chip and soft



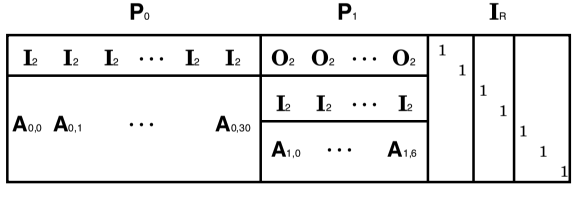


Figure 5.6 *Hong-Patel (83, 76) S2EC code.*

TABLE 5.3 Maximum Code Length *N* (Bits) Compared

Check-bit length R	SEC code length in bits $N = 2^R - 1$	Maximal SbEC code length in bits $N = b\dfrac{2^R - 1 - 2^b(2^c - 1)}{2^b - 1} + c$		SEC-DED code length in bits $N = 2^{R-1}$
		$b = 2$ bits	$b = 4$ bits	
2	3	—	—	2
3	7	—	—	4
4	15	10	—	8
5	31	19	—	16
6	63	42	—	32
7	127	83	—	64
8	255	170	68	128
9	511	339	133	256
10	1,023	682	262	512
11	2,047	1,363	519	1,024
12	4,095	2,730	1,092	2,048

Source: *[HONG72]. © 1972 IEEE.*

errors in another chip simultaneously. Therefore they need at least to detect the double-byte errors. For this reason SbEC-DbED codes have been applied to computer memory systems with byte length $b = 4$ bits [NARA80, KANE82, BISH96]. This is because 4-bit-byte organized semiconductor DRAM chips have been popularly used in high-speed memory systems.

5.2.1 Reed-Solomon (RS) Codes

Reed-Solomon codes are a general class of codes having any distance d over $GF(q)$ [REED60]. We can design the SbEC-DbED codes as the RS codes with distance 4 over $GF(2^b)$. The codes can be expressed by the $\mathbf{H}$ matrix with three rows:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & & \mathbf{I} & & \mathbf{I} \\ \mathbf{I} & \mathbf{T} & \cdots & \mathbf{T}^i & \cdots & \mathbf{T}^{2^b-2} \\ \mathbf{I} & \mathbf{T}^2 & & \mathbf{T}^{2i} & & \mathbf{T}^{2(2^b-2)} \end{bmatrix}, \tag{5.22}$$

where $\{\mathbf{0}, \mathbf{T}, \mathbf{T}^2, \ldots, \mathbf{T}^{2^b-2}, \mathbf{T}^{2^b-1} = \mathbf{I}\} \in GF(2^b)$ and $\mathbf{T}$ is the companion matrix defined by Eq. (5.1).

Wolf [WOLF69] suggested lengthening distance-4 RS codes by appending three columns to the $\mathbf{H}$ matrix without weakening the error control capability of the codes. The appended columns are the 3×3 identity matrix

$$\mathbf{H} = \left[\begin{array}{cccccc|ccc} \mathbf{I} & \mathbf{I} & & \mathbf{I} & & \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{I} & \mathbf{T} & \cdots & \mathbf{T}^i & \cdots & \mathbf{T}^{2^b-2} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{I} & \mathbf{T}^2 & & \mathbf{T}^{2i} & & \mathbf{T}^{2(2^b-2)} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{array}\right]. \tag{5.23}$$

TABLE 5.4 Code Parameters for RS S*b*EC-D*b*ED Codes

b (bits)	N_{max} (bits)	K_{max} (bits)
2	12	6
3	30	21
4	72	60

The code length N (bits) of this modified code is equal to

$$N = b \cdot (2^b + 2). \tag{5.24}$$

An implementation of the high-speed parallel encoder / decoder of the extended RS codes is given in [BHAT78].

Today's high-speed computer systems have adopted the information-bit lengths $K = 64$, 128, or 256 bits. The conventional Reed-Solomon S*b*EC-D*b*ED codes, hereafter abbreviated as RS S*b*EC-D*b*ED codes, however, cannot be used in memory systems with a byte lengths of $b = 2$, 3, and 4 bits. This is because the maximum information-bit lengths of these codes are 6, 21, and 60 bits for $b = 2$, 3, and 4 bits, respectively. This is shown in Table 5.4 and calculated using Eq. (5.24). From this reason modified RS S*b*EC-D*b*ED codes must take any values of byte size b and code length N [CART74, CART80, KANE82, ITOH83, CHEN86a, CHEN86b].

5.2.2 Kaneda-Fujiwara Codes

The RS S*b*EC-D*b*ED codes always require three check bytes, and hence do not have the flexibility to extend the code length. Here a new class of S*b*EC-D*b*ED codes is shown for an arbitrary code length and byte length [KANE82].

The **H** matrix shown in Eq. (5.23) can be converted to the **H** matrix whose first row has all **I**'s. This can be accomplished by the following algorithm:

Step 1. *The second row of* ***H*** *shown in Eq. (5.23) is multiplied by a suitable nonzero element* $\boldsymbol{T}^a$*, where* $0 \leq a \leq 2^b - 2$.

Step 2. *The multiplied result and the third row are added to the first row of* ***H*** *in Eq. (5.23).*

Step 3. *If an element of the resultant (first) row is not equal to* ***I****, then this can be made equal to* ***I*** *by multiplying the column by a nonzero scalar. Note that multiplying a column of* ***H*** *by a nonzero scalar does not change the distance of the code.*

For example, let **H** be the following (12, 6) S2EC-D2ED codes:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{I} & \mathbf{T}^2 & \mathbf{T} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{5.25}$$

Here **T** is the companion matrix defined by the primitive polynomial $\mathbf{g}(x) = x^2 + x + 1$, **I** is a 2×2 identity matrix, and **0** is a 2×2 zero matrix. According to the algorithm mentioned above, the **H** matrix can be converted to the matrix having all **I**'s in the first row by the following two steps.

Step 1. Multiply the second row by $\mathbf{T}$ and add the second and third rows to the first. We then get the following $\mathbf{H}'$:

$$\mathbf{H}' = \begin{bmatrix} \mathbf{T} & \mathbf{T} & \mathbf{I} & \mathbf{I} & \mathbf{T} & \mathbf{I} \\ \mathbf{T} & \mathbf{I} & \mathbf{T}^2 & \mathbf{0} & \mathbf{T} & \mathbf{0} \\ \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}.$$

Step 2. To obtain all $\mathbf{I}$'s in the first row, we need to multiply columns 1, 2, and 5 by $\mathbf{T}^2$. We then get $\mathbf{H}''$ shown below ($\because \mathbf{T}^2 \cdot \mathbf{T}^2 \cdot \mathbf{T}^2 = \mathbf{T}^6 = \mathbf{I}$):

$$\mathbf{H}'' = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{T}^2 & \mathbf{I} & \mathbf{T}^2 & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{5.26}$$

In general, we can find at least one $a (a \leq 2^b - 2)$ to multiply the second row by $\mathbf{T}^a$, and then we can finally get the matrix including the first row having all $\mathbf{I}$'s. This is easy to prove.

Here we can write the converted matrix of Eq. (5.26), in general, as

$$\mathbf{H}_1 = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \dots & \mathbf{I} \\ \mathbf{h}_0 & \mathbf{h}_1 & \mathbf{h}_2 & \dots & \mathbf{h}_{n-1} \end{bmatrix}, \qquad n = 2^b + 2, \tag{5.27}$$

where $\mathbf{h}_0, \mathbf{h}_1, \dots, \mathbf{h}_{n-1}$ are column vectors each having two elements. For example, in Eq. (5.26) we can express the second and the third rows in $\mathbf{H}''$ by the following:

$$\mathbf{h}_0 = \begin{pmatrix} \mathbf{I} \\ \mathbf{T}^2 \end{pmatrix}, \quad \mathbf{h}_1 = \begin{pmatrix} \mathbf{T}^2 \\ \mathbf{I} \end{pmatrix}, \quad \mathbf{h}_2 = \begin{pmatrix} \mathbf{T}^2 \\ \mathbf{T}^2 \end{pmatrix}, \quad \mathbf{h}_3 = \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \end{pmatrix}, \quad \mathbf{h}_4 = \begin{pmatrix} \mathbf{I} \\ \mathbf{0} \end{pmatrix}, \quad \mathbf{h}_5 = \begin{pmatrix} \mathbf{0} \\ \mathbf{I} \end{pmatrix}.$$

By this approach and notation, a new class of SbEC-DbED codes is obtained as shown in the following theorem.

Theorem 5.7 *Let $\boldsymbol{H}_1$ be the converted $\boldsymbol{H}$ matrix of an $(N_1, N_1 - R_1)$ SbEC-DbED code whose first row is an all-$\boldsymbol{I}$'s vector of the following form:*

$$\boldsymbol{H}_1 = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{I} & \dots & \boldsymbol{I} \\ \boldsymbol{h}_0 & \boldsymbol{h}_1 & \boldsymbol{h}_2 & \dots & \boldsymbol{h}_{n_1-1} \end{bmatrix},$$

where $N_1 = n_1 b$, $R_1 = r_1 b$. Let $\boldsymbol{H}_2$ be the nonconverted $\boldsymbol{H}$ matrix of an $(N_2, N_2 - R_2)$ SbEC-DbED code, where $N_2 = n_2 b$, $R_2 = r_2 b$. The linear code defined by the following $\boldsymbol{H}$ matrix is an SbEC-DbED code of length $n_1 n_2$ bytes with $r_1 + r_2 - 1$ check bytes.

$$\boldsymbol{H} = \left[\begin{array}{cccc:cccc:c:cccc} & \boldsymbol{H}_2 & & & & \boldsymbol{H}_2 & & & & & & \boldsymbol{H}_2 & & \\ \boldsymbol{h}_0 & \boldsymbol{h}_0 & \dots & \boldsymbol{h}_0 & \boldsymbol{h}_1 & \boldsymbol{h}_1 & \dots & \boldsymbol{h}_1 & \cdots & \boldsymbol{h}_{n_1-1} & \boldsymbol{h}_{n_1-1} & \dots & \boldsymbol{h}_{n_1-1} \end{array}\right]. \tag{5.28}$$

The theorem can be easily proved such that every combination of three or fewer columns in the **H** matrix shown in Eq. (5.28) is linearly independent. The details of the proof are left to the reader.

Theorem 5.8 *Let $\boldsymbol{H}_0$ be the $\boldsymbol{H}$ matrix of an $(N, N-R)$ SbEC-DbED code, where $N = nb$, $R = rb$. The code defined by the following $\boldsymbol{H}$ matrix is a $(2N, 2N-R-b)$ SbEC-DbED code:*

$$\boldsymbol{H} = \left[\begin{array}{c|c} \boldsymbol{H}_0 & \boldsymbol{H}_0 \\ \hline \boldsymbol{0}\,\boldsymbol{0}\,\boldsymbol{0}\,\cdots\boldsymbol{0} & \boldsymbol{I}\,\boldsymbol{I}\,\boldsymbol{I}\,\cdots\boldsymbol{I} \end{array}\right], \qquad \{\boldsymbol{0}, \boldsymbol{I}\} \in GF(2^b). \tag{5.29}$$

Proof It is simple to show that the **H** matrix defined by Eq. (5.29) has an SbEC property. If there are double-byte errors, E_1 and E_2, in the received word such that one byte error E_1 is in first portion in **H** that has all **0**'s in the bottom row, and the other byte error E_2 is in the second portion that has all **I**'s in the bottom row, the resultant syndrome can be expressed as

$$E_1 \cdot \begin{bmatrix} \mathbf{h}_{0,i} \\ \mathbf{h}_{1,i} \\ \vdots \\ \mathbf{h}_{r-1,i} \\ \mathbf{0} \end{bmatrix} + E_2 \cdot \begin{bmatrix} \mathbf{h}_{0,j} \\ \mathbf{h}_{1,j} \\ \vdots \\ \mathbf{h}_{r-1,j} \\ \mathbf{I} \end{bmatrix} = \begin{bmatrix} E_1 \cdot \mathbf{h}_{0,i} + E_2 \cdot \mathbf{h}_{0,j} \\ E_1 \cdot \mathbf{h}_{1,i} + E_2 \cdot \mathbf{h}_{1,j} \\ \vdots \\ E_1 \cdot \mathbf{h}_{r-1,i} + E_2 \cdot \mathbf{h}_{r-1,j} \\ E_2 \end{bmatrix},$$

where $[\mathbf{h}_{0,i}\ \mathbf{h}_{1,i}\ \ldots\ \mathbf{h}_{r-1,i}\ \mathbf{0}]^T$ is the i-th column vector included in the first portion in **H** and $[\mathbf{h}_{0,j}\ \mathbf{h}_{1,j}\ \ldots\ \mathbf{h}_{r-1,j}\ \mathbf{I}]^T$ is the j-th column vector included in the second portion. It is easy to see that this syndrome is nonzero and not equal to that of the single-byte errors. Therefore the code defined by Eq. (5.29) is an SbEC-DbED code. Q.E.D.

Example 5.7 [KANE82]

Let $\mathbf{H}_2$ be the following $(12, 6)$ S2EC-D2ED code:

$$\mathbf{H}_2 = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{I} & \mathbf{T}^2 & \mathbf{T} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{5.30}$$

As was mentioned before, the **H** matrix indicated in Eq. (5.30) can be converted to the form of $\mathbf{H}_1$ in Eq. (5.31) by choosing $\mathbf{T}^a = \mathbf{T}$ as follows:

$$\mathbf{H}_1 = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}. \tag{5.31}$$

The following $(72, 62)$ S2EC-D2ED code is designed [KANE82] by combining $\mathbf{H}_1$ and $\mathbf{H}_2$:

$$\mathbf{H} = \left[\begin{array}{cccccc|cccccc|cccccc|cccccc|cccccc|cccccc}
\mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & & & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & & & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & & & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & & & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & & \\
\mathbf{I} & \mathbf{T} & \mathbf{T}^2 & & \mathbf{I} & & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & & \mathbf{I} & & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & & \mathbf{I} & & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & & \mathbf{I} & & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & & \mathbf{I} & \\
\mathbf{I} & \mathbf{T}^2 & \mathbf{T} & & & \mathbf{I} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T} & & & \mathbf{I} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T} & & & \mathbf{I} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T} & & & \mathbf{I} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T} & & & \mathbf{I} \\
\mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & & & & & & & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \\
\mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & & & & & & & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I}
\end{array}\right].$$

In the same manner the S2EC-D2ED codes, whose $\mathbf{H}$ matrices have seven rows, can be designed from two S2EC-D2ED codes—one having five rows and the other having three rows. In general, the SbEC-DbED codes, whose $\mathbf{H}$ matrices have odd number of rows, can be obtained according to this method.

If an even number of rows, for example, six rows, is required, then Theorem 5.8 is used to design the code with six rows, as shown in Figure 5.7 [KANE82].

The $\mathbf{H}$ matrix shown in the figure does not indicate the check byte positions clearly. The fourth, fifth, and sixth rows in this matrix are added to the first row. Then we obtain the position of the check bytes, whose columns have exactly one $\mathbf{I}$ and five zeros in the remaining positions. The resultant matrix is shown in the lower panel, where the check-byte positions are indicated by circled $\mathbf{I}$'s [KANE82].

The code length N (bits) of this code is given as follows:

$$N = \begin{cases} b \cdot (2^b + 2)^{(r-1)/2} & r: \text{ odd } (\geq 3), \\ 2b \cdot (2^b + 2)^{(r-2)/2} & r: \text{ even } (\geq 4). \end{cases}$$

Table 5.5 shows the code length N (bits) for byte length b (bits) and the number of rows r. For the byte length $b = 1$ bit, the code is reduced to the SEC-DED code.

Figure 5.8 shows the check-bit length of the SbEC-DbED codes for byte lengths $b = 2, 3$, and 4 bits. The decoding procedure of this code is simple. A single-byte error is corrected by using the same procedure as that of the SbEC codes. A double-byte error is detected when the syndrome is nonzero and none of the error byte pointers indicate an error.

Rotational / Modularized SbEC-DbED Codes In theory, derivation of rotational SbEC-DbED codes is difficult, and no derivation has yet been obtained for arbitrary values of b and r. However, in practice, some code parameters of byte length $b = 4$ bits and information lengths $K = 64$ and 128 bits have been tried and produced some excellent codes having the property of the minimum-weight & equal-weight-row code (see Section 3.1) [KANE82]. Although such *modularized codes* are not identical to the rotational codes, they provide modularized organization of the encoding / decoding circuits, and hence some modularized codes have been applied to commercial computer systems.

Definition 5.2 The SbEC-DbED codes whose encoding / decoding circuits can be organized using p identical circuit modules are called *p-modularized SbEC-DbED codes.* □

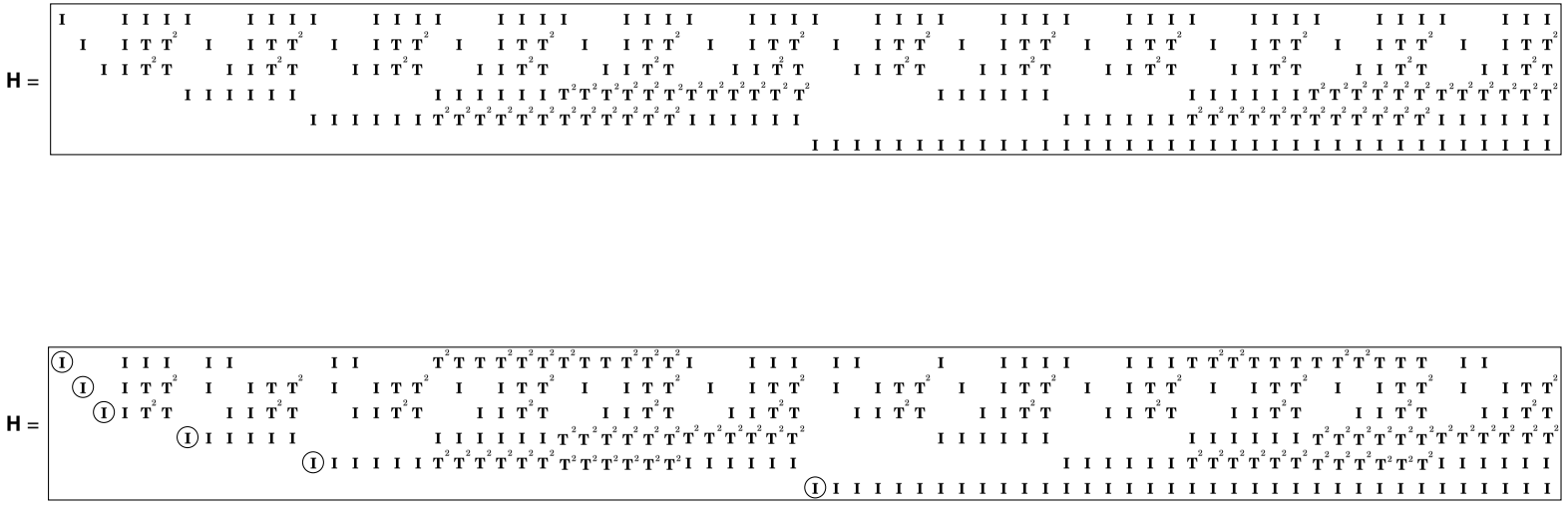


Figure 5.7 *(144,122) S2EC-D2ED codes. (nonsystematic code (upper) and systematic code (lower)).* Source: *[KANE82]. © 1982 IEEE.*

TABLE 5.5 Code Length *N* (Bits) of Kaneda-Fujiwara S*b*EC-D*b*ED Code

Number of rows	*b*: Byte length (bits)				
r	1[a]	2	3	4	5
3	4	12	30	72	170
4	8	24	60	144	340
5	16	72	300	1,296	5,780
6	32	144	600	2,592	11,560
7	64	432	3,000	23,328	196,520

Source: *[KANE82]. © 1982 IEEE.*

Note: Number of check bits $R = b \times r$.

[a] The codes with $b = 1$ are equal to the SEC-DED codes.

Theorem 5.9 *The code shown in Eq. (5.32) presents 2-modularized SbEC-DbED codes:*

$$H = \left[\begin{array}{cccc|cccc|c} \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{I} & \dots \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{I} & \dots \boldsymbol{I} & \boldsymbol{I} \\ \boldsymbol{T} & \boldsymbol{T}^2 & \boldsymbol{T}^3 & \dots \boldsymbol{T}^p & \boldsymbol{T}^{-p} & \boldsymbol{T}^{-p+1} & \boldsymbol{T}^{-p+2} & \dots \boldsymbol{T}^{-1} & \boldsymbol{I} \\ \boldsymbol{T}^{-1} & \boldsymbol{T}^{-2} & \boldsymbol{T}^{-3} & \dots \boldsymbol{T}^{-p} & \boldsymbol{T}^{p} & \boldsymbol{T}^{p-1} & \boldsymbol{T}^{p-2} & \dots \boldsymbol{T} & \boldsymbol{I} \end{array}\right], \qquad (5.32)$$

module 0 *module 1*

where $\boldsymbol{0}$ *and* $\boldsymbol{I}$ *are zero element and identity element in* $GF(2^b)$*, respectively, and*

$$\{\boldsymbol{0}, \boldsymbol{I}, \boldsymbol{T}, \boldsymbol{T}^2, \boldsymbol{T}^3, \dots, \boldsymbol{T}^p, \boldsymbol{T}^{-1}, \boldsymbol{T}^{-2}, \boldsymbol{T}^{-3}, \dots, \boldsymbol{T}^{-p}\} \in GF(2^b) \qquad \text{for } p = 2^{b-1} - 1.$$

Proof In the original Reed-Solomon S*b*EC-D*b*ED code shown in Eq. (5.22), the second column of its **H** matrix is divided by **T**, and the third column is divided by $\mathbf{T}^2$. In the same

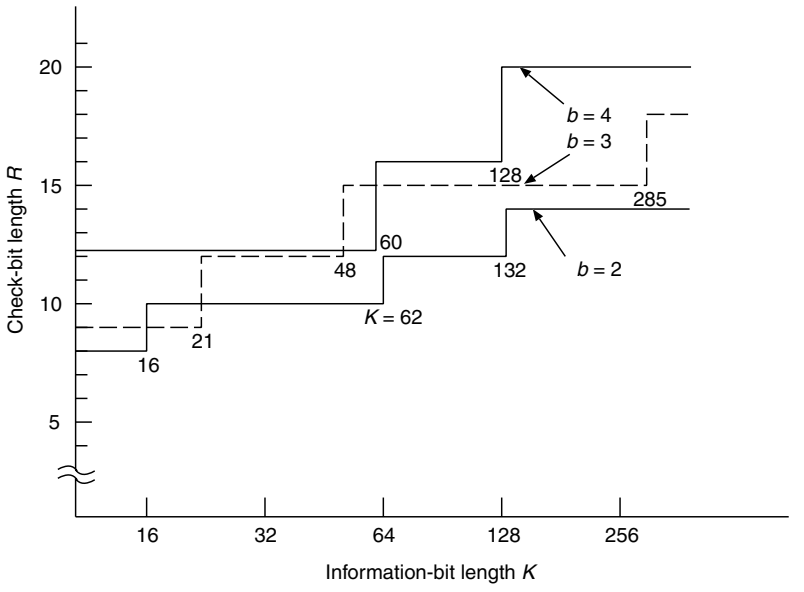


Figure 5.8 *Comparison of check-bit lengths and information-bit lengths of the Kaneda-Fujiwara SbEC-DbED codes.*

manner the i-th column is divided by $\mathbf{T}^{i-1}$. In this case the first column (all **I** vectors) is removed. Next each row of the obtained matrix is cyclically shifted upward by one place. Finally the 3×3 identity matrix is appended to the resultant matrix to derive the **H** matrix of Eq. (5.32). Therefore this code satisfies the SbEC-DbED conditions. The module 0 and the module 1 shown in Eq. (5.32) have the same circuits for encoding / decoding because the same three row vectors are used in each module. Thus the code shown in Eq. (5.32) is a 2-modularized SbEC-DbED code. Q.E.D.

The code length (in bits) of this 2-modularized code is given by

$$N = b \cdot (2^b + 1).$$

In the **H** matrix shown in Eq. (5.32), the product value of the second row element and the third row element in each column is constant, meaning all **I**'s, except for the columns of check bytes. This constant value can be selected from the nonzero elements in $GF(2^b)$. If there is a column vector whose second row element is same as the third row element, then this column should be removed. The **H** matrix shown in Eq. (5.33) is an example of this type of 2-modularized $(68, 56)$ S4EC-D4ED code whose constant product value is $\mathbf{T}^{14}$:

$$\mathbf{H} = \left[\begin{array}{ccccc|ccccc|cc} \mathbf{I} & \mathbf{I} & \mathbf{I} & \dots & \mathbf{I} & \mathbf{I} & \mathbf{I} & \dots & \mathbf{I} & \mathbf{I} & & \\ \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \dots & \mathbf{T}^6 & \mathbf{T}^{14} & \mathbf{T}^{13} & \dots & \mathbf{T}^8 & & \mathbf{I} & \\ \mathbf{T}^{14} & \mathbf{T}^{13} & \mathbf{T}^{12} & \dots & \mathbf{T}^8 & \mathbf{I} & \mathbf{T} & \dots & \mathbf{T}^6 & & & \mathbf{I} \end{array}\right]. \tag{5.33}$$

In this case the companion matrix **T** is derived from the primitive polynomial $\mathbf{g}(x) = x^4 + x + 1$.

Another interesting type of modularized code is introduced in [NARA80]. The code shown in Figure 5.9 is an example of the modularized $(80, 64)$ S4EC-D4ED code. This matrix also

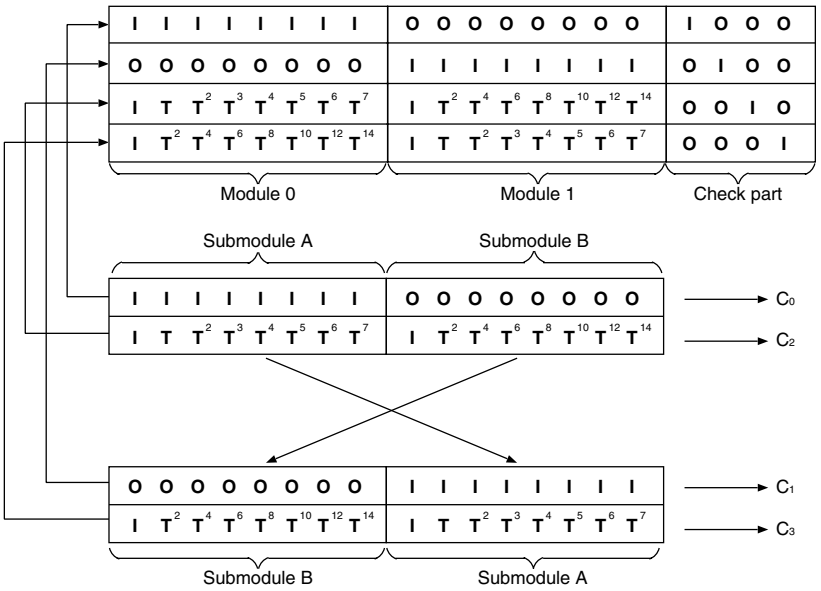


Figure 5.9 *Modularized (80, 64) S4EC-D4ED code.*

TABLE 5.6 Number of Rotational (144, 128) S4EC-D4ED Codes

Product value	$\mathbf{I}$	$\mathbf{T}$	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$
Number of codes	0	4	4	11	4	0	11	11	4	11	0	11	11	11	11

Source: *[KANE82]. © 1982 IEEE.*

has two identical modules, module 0 and module 1 of Figure 5.9. In addition to this, each module can be comprised of two types of submodules, submodule A and submodule B in the figure. Hence this organization presents an easy implementation of the encoding / decoding circuit. In this case the columns for the check bytes can be appended to the matrix obtained.

Next we consider the rotational SbEC-DbED codes. These codes are sometimes derived via an exhaustive computer search, and hence they are optimized as minimum-weight & equal-weight-row codes (see Section 3.1). To increase the modularity of the matrix organization, the 2-modularized technique shown in Theorem 5.9 can be applied to the rotational SbEC-DbED code. That is, we can apply this technique to the basic submatrix $\mathbf{H}_0$ in the $\mathbf{H}$ matrix of the rotational code. Let submatrix $\mathbf{H}_0$ have an all-$\mathbf{I}$ row vector in the first row. The product of the second row element and the third row element provides a constant element in each column of $\mathbf{H}_0$, except for the columns of check bytes. Thus the submatrix $\mathbf{H}_0$ itself has a 2-modularized organization.

To see this property, take the following simple example where the companion matrix $\mathbf{T}$ is derived from the polynomial $\mathbf{g}(x) = x^4 + x + 1$. Eight column vectors and one check column vector are selected from the $\mathbf{H}$ matrix of Eq. (5.33) as shown in Eq. (5.34). Note that to have four submatrices in the $\mathbf{H}$ matrix of the rotational code, one all-$\mathbf{0}$ row vector is added.

$$\mathbf{H}_0 = \left[\begin{array}{cccc|cccc|c} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \mathbf{T}^{14} & \mathbf{T}^{13} & \mathbf{T}^{12} & \mathbf{T}^{11} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^3 & \mathbf{0} \\ \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^3 & \mathbf{T}^{14} & \mathbf{T}^{13} & \mathbf{T}^{12} & \mathbf{T}^{11} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array}\right] \tag{5.34}$$

The rotational code organized by the submatrix shown in Eq. (5.34) does not always have the D4ED property. This can be verified by a computer program. This is the way we were able to select 35 $\left(= \binom{7}{4}\right)$ submatrices and 15 product values. Table 5.6 shows the number of rotational (144, 128) S4EC-D4ED codes for each product value. Figure 5.10 shows an example of the rotational (144, 128) S4EC-D4ED code with minimum-weight & equal-weight-row property. The weight of this matrix is equal to 592.

Figure 5.11 shows four $\mathbf{H}_0$ submatrices satisfying the minimum-weight & equal-weight-row rotational (144, 128) S4EC-D4ED codes that do not have constant product values in each column. The weight of each $\mathbf{H}$ matrix is equal to 568. These codes, including the code shown in Figure 5.10, finally give an 8-modularized organization of their encoding / decoding circuits.

As another practical example, let us look at a minimum-weight & equal-weight-row rotational (80, 64) S4EC-D4ED code provided in the $\mathbf{H}$ matrix shown in Eq. (5.35):

$$\mathbf{H} = \left[\begin{array}{ccccc|ccccc|ccccc|ccccc} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & & & & & & \mathbf{T}^2 & \mathbf{T} & \mathbf{I} & \mathbf{T}^{14} & & \mathbf{T}^{14} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \\ \mathbf{T}^{14} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & & & & & & \mathbf{T}^2 & \mathbf{T} & \mathbf{I} & \mathbf{T}^{14} & \\ \mathbf{T}^2 & \mathbf{T} & \mathbf{I} & \mathbf{T}^{14} & & \mathbf{T}^{14} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & & & & & \\ & & & & & \mathbf{T}^2 & \mathbf{T} & \mathbf{I} & \mathbf{T}^{14} & & \mathbf{T}^{14} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \end{array}\right] \cdot \tag{5.35}$$

$\mathbf{H}_0$ $\mathbf{H}_1$ $\mathbf{H}_2$ $\mathbf{H}_3$ (b=4)

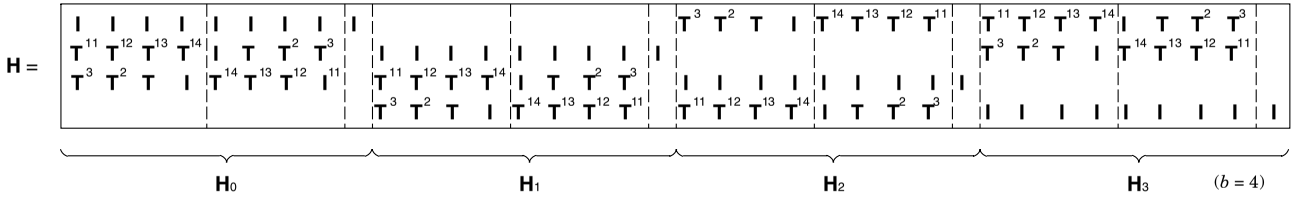


Figure 5.10 *Minimum-weight & equal-weight-row rotational (144,128) S4EC-D4ED code.* Source: *[KANE82]. © 1982 IEEE.*

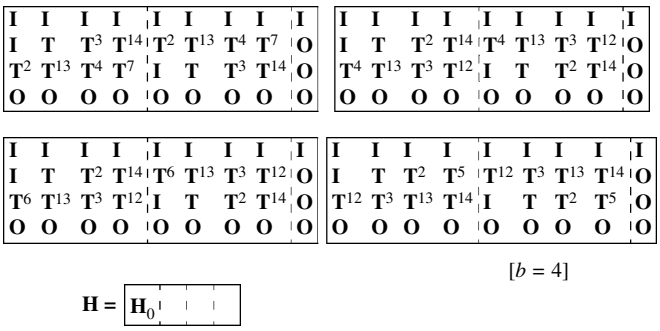


Figure 5.11 *Basic submatrices $\mathbf{H_0}$'s for rotational (144,128) S4EC-D4ED codes.* Source: *[KANE82]. © 1982 IEEE.*

Figure 5.12 shows an example of the syndrome decoder corresponding to the basic submatrix $\mathbf{H}_0$ of the code in Eq. (5.35). Note that the circuit complexity and the propagation delay of the decoder for the SbEC-DbED codes with $K = 64$ and 128 bits represent a 20% to 30% increase compared to those of the SEC-DED codes [KANE82].

5.2.3 Chen Codes

We now introduce a new class of SbEC-DbED codes that is more efficient than the Kaneda-Fujiwara codes [CHEN86a, CHEN86b]. The Chen code design is based on Theorems 5.7 and 5.8 in the Kaneda-Fujiwara codes. In this code the $\mathbf{H}_1$ and $\mathbf{H}_2$ in Theorem 5.7 are replaced by codes based on the theory of a *generalized BCH bound* [HART72].

The following theorem is a special case of the generalized BCH bound:

Theorem 5.10 [HART72] *Let β be a primitive root of $x^n - 1$, and let*

$$\{\beta^{\alpha_0}, \beta^{\alpha_0+\alpha_1}, \beta^{\alpha_0+\alpha_2}, \beta^{\alpha_0+\alpha_1+\alpha_2}\}$$

be a subset of the zeros of the generator polynomial of a cyclic code of length n over $GF(q)$, where $q = 2^b$. If α_1 and α_2 are relatively prime to n, then the minimum distance of the code is equal to or greater than 4.

Theorem 5.11 [CHEN86a] *Let $\boldsymbol{C}$ be a cyclic code over $GF(q)$ of length $n = q^m + 1$ bytes and generated by the minimal polynomial of β, where $q = 2^b$, m is even, and β is a primitive root of $x^n - 1$. Then $\boldsymbol{C}$ is an SbEC-DbED code with $2m$ check bytes.*

Proof Since the generator polynomial $\mathbf{g}(x)$ of the cyclic code over $GF(q)$ is the minimal polynomial of β, the roots of $\mathbf{g}(x)$ can be expressed as β^i, where

$$i = 1, q, q^2, \ldots, q^m, q^{m+1}, \ldots, q^{2m-1}.$$

Since $\beta^n = 1$, then $\beta^{q^m} = \beta^{-1}$, and $\beta^{q^{m+1}} = \beta^{-q}$. Thus $\mathbf{g}(x)$ contains a subset of the roots β, β^q, β^{-q}, and β^{-1}. Let $\alpha_0 = 1$, $\alpha_1 = q - 1$, and $\alpha_2 = -(q + 1)$. Since m is even, α_1 and α_2 are relatively prime to n. According to Theorem 5.10, the code is an SbEC-DbED code. Since the degree of $\mathbf{g}(x)$ is $2m$, the number of check bytes of the code is $2m$. Q.E.D.

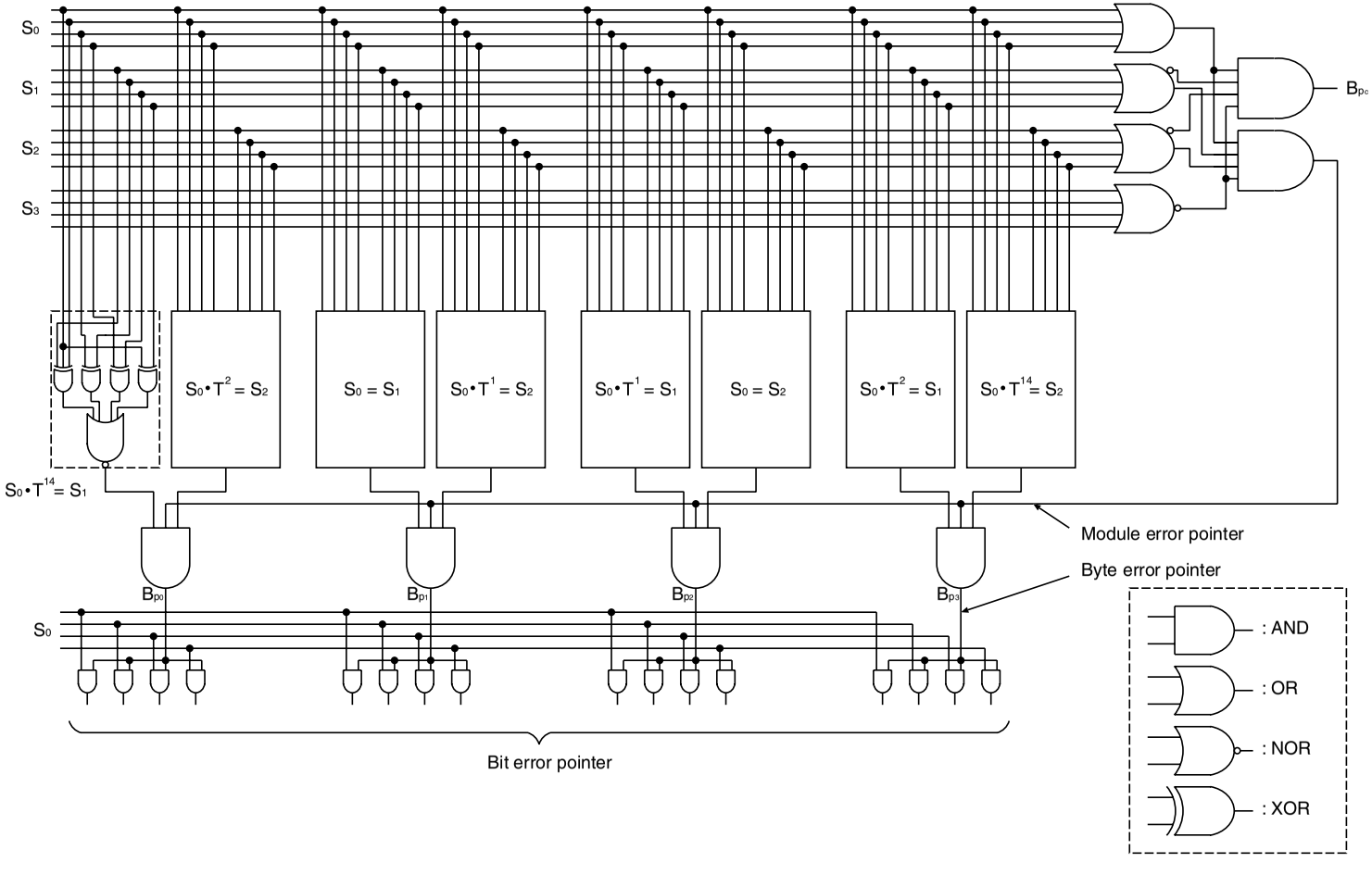


Figure 5.12 *Syndrome decoder LSI for the* $\boldsymbol{H}_0$ *module of the code shown in Eq. (5.35).* Source: *[KANE82]. © 1982 IEEE.*

Example 5.8

Let $b = 2$, $q = 2^2$, and $m = 2$, then $n = 17$. There exists a cyclic SbEC-DbED code over $GF(2^2)$ of byte length 17. The code contains 13 information bytes in each codeword.

Theorem 5.12 [CHEN86a] *A cyclic code over $GF(q)$ of length $n = q^{2m} - 1$ bytes generated by the product of the minimal polynomials of 1, β, and β^{q^m+1}, where $q = 2^b$ and β is a primitive root of $x^n - 1$, is an SbEC-DbED code with $3m + 1$ check bytes.*

Proof The generator polynomial is the product of $x - 1$ and the minimal polynomials of β and β^{q^m+1}. The degrees of the minimal polynomials of β and β^{q^m+1} are $2m$ and m, respectively. Thus the number of check bytes is $3m + 1$. The generator polynomial contains as roots the elements 1, β, β^{q^m}, and β^{q^m+1}. Let $\alpha_0 = 0$, $\alpha_1 = 1$, and $\alpha_2 = q^m$. Since α_1 and α_2 are relatively prime to n, the cyclic code is an SbEC-DbED code according to Theorem 5.10. Q.E.D.

Example 5.9

Let $b = 3$, $q = 2^3 = 8$, and $m = 2$, then $n = 4095$. A cyclic SbEC-DbED code over $GF(2^3)$ of length 4095 with seven check bytes can be constructed. The generator polynomial can be taken as the product of the minimal polynomials of 1, β, and β^{65}, where β is a primitive root of $x^{4095} - 1$.

Since the generator polynomial of a code in Theorem 5.12 contains 1 as the root, the $\mathbf{H}$ matrix of the code can be arranged so that the first row is a vector of all ones. By this $\mathbf{H}$ matrix form, the code can be extended by one byte in adding to the matrix a column vector of a one followed by $3m$ zeros. It can be shown that the extended code is also an SbEC-DbED code. We state this result as the next theorem.

Theorem 5.13 [CHEN86a] *An SbEC-DbED code of byte length $n = q^{2m}$ with $3m + 1$ check bytes, where $q = 2^b$, can be obtained by extending the code of Theorem 5.12.*

The codes obtained by Theorems 5.11, 5.12, and 5.13 can be used as $\mathbf{H}_1$ in Theorem 5.7. Theorem 5.8 can also be applied to these codes to double the code length. The code length obtained is given in Table 5.7.

Table 5.8 shows the code length $N = n \cdot b$ bits of the Chen code. Figure 5.13 shows the check-bit length of the Chen SbEC-DbED code for byte lengths $b = 2, 3$, and 4 bits.

TABLE 5.7 Code Length n (Bytes) of Chen Codes

Class	$\mathbf{H}_1$	$\mathbf{H}_2$	n	r
1	Extended RS code	Code of Theorem 5.11	$(q+2)(q^{m_1}+1)$	$2m_1 + 2$
2	Extended RS code	Code of Theorem 5.13	$(q+2)q^{2m_3}$	$3m_3 + 3$
3	Code of Theorem 5.12	Code of Theorem 5.11	$(q^{2m_2}-1)(q^{2m_1}+2)$	$3m_2 + 2m_1$
4	Code of Theorem 5.12	Code of Theorem 5.13	$(q^{2m_2}-1)q^{2m_3}$	$3m_2 + 3m_3 + 1$

Note: $\mathbf{H}_1$ and $\mathbf{H}_2$ are defined in Theorem 5.7. m_1: m = even in Theorem 5.11, m_2: m in Theorem 5.12, m_3: m in Theorem 5.13.

TABLE 5.8 Code Length *N* (Bits) of Chen Codes

b (bits) / *r*	2	3	4	5	
3	12	30	72	170	Extended RS code [WOLF69]
4	34	195	1,028	5,125	Theorem 5.11
5	68[a]	390[a]	2,056	10,250	Theorem 5.8
6	204	1,950	18,504	174,250	Class 1
7	876	12,672	264,192	5,253,120	Construction 4 in [CHEN86b]

Source: [CHEN86a, CHEN86b]. © 1986 IEEE.

Note: Check-bit length $R = r \cdot b$.

[a](82, 72) S2EC-D2ED code and (399, 384) S3EC-D3ED code are obtained by computer experiments in [ITOH83] and [CHEN86b], respectively.

Converted Chen Codes A simple code design technique for constructing efficient S*b*EC-D*b*ED codes was proposed by [CHEN92]. The code designed by this technique requires fewer check bits than the existing codes.

First, the **H** matrix of the existing S*b*EC-D*b*ED codes, such as the former Chen codes, are converted to a normalized form of **H** matrix whose first row is an all-**I** vector. That is, the first nonzero element in $GF(2^b)$; for example, $\mathbf{T}_{o,j} \in GF(2^b), j = 0, 1, \cdots, n-1$, in each column of the original **H** matrix is transformed into an identity element and the other elements of the column are calculated by $\mathbf{T}_{i,j} \cdot \mathbf{T}_{o,j}^{-1}$, where $\mathbf{T}_{i,j}$ is an i-th row and j-th column element, $i = 0, 1, \cdots, r-1$. The resultant matrix has a normalized form of **H**. It can be easily proved that the code function of the resultant matrix is preserved even in this transformation.

Second, the conversion procedure is performed to the **H** matrix of the existing S*b*EC-D*b*ED code as follows.

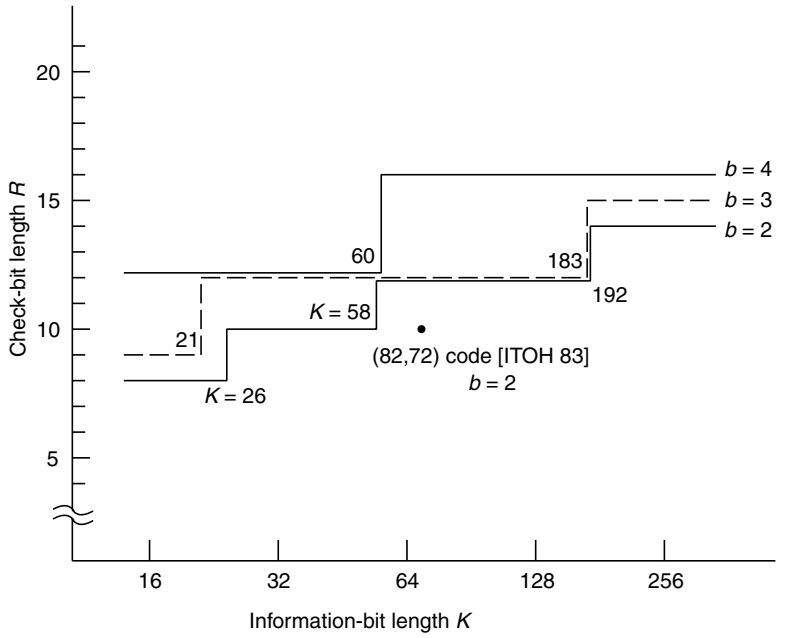


Figure 5.13 *Comparison of check-bit lengths and information-bit lengths of the Chen SbEC-DbED codes.*

Conversion Procedure

Step 1. *Transform the* $\boldsymbol{H}$ *matrix of SbEC-DbED code into a normalized form* $\boldsymbol{H}_1$.

Step 2. *Let* $\Omega = \{i_1, i_2, \cdots, i_s\}$ *be a subset of* $\{1, 2, \cdots, b\}$.

Step 3. *For each of the n columns of* $\boldsymbol{H}_1$*, delete binary columns whose position numbers belong to* Ω*. Let the number of deleted columns be s.*

Step 4. *Delete s binary rows that contain all zeros from the binary matrix obtained in step 3. Let* $\boldsymbol{H}_2$ *be the resultant matrix.*

It can be easily proved that the resultant matrix $\mathbf{H}_2$ is the $\mathbf{H}$ matrix of the SbEC-DbED codes with symbol size $b - s$ (bits), code length n (bytes), and number of check bits $\geq br - s$.

Example 5.10 (136, 122) S4EC-D4ED Code

The code design technique mentioned in this example starts from the code with $b = 5$ bits that uses $3b$ check bits. The parity-check matrix with normalized form is presented by

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} & \cdots & \mathbf{I} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{T} & \cdots & \mathbf{T}^i & \cdots & \mathbf{T}^{30} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{I} & \mathbf{T}^2 & \cdots & \mathbf{T}^{2i} & \cdots & \mathbf{T}^{29} \end{bmatrix}, \tag{5.36}$$

where $\mathbf{I}$ is a 5×5 identity matrix, $\mathbf{0}$ is a 5×5 zero matrix, and $\mathbf{T}$ is a companion matrix determined by the binary primitive polynomial $x^5 + x^2 + 1$. Note that $\mathbf{H}$ represents a 15×170 binary matrix. To reduce symbol size to 4 bits, we delete the last column of each of the 15×5 binary submatrices in Eq. (5.36). That is, $s = 1$ and $\Omega = \{5\}$ in step 2 of the conversion procedure. The resultant matrix has an all zeros in the fifth row; these zeros can be deleted without affecting the error correction capability. As a result the new 14×136 binary matrix presents a parity-check matrix of the S4EC-D4ED code. Selecting a set of 78 from the 136 available columns forms the $\mathbf{H}$ matrix of an (78, 64) S4EC-D4ED code. This is shown in Figure 5.14. The error detection capability of this code is shown in Table 5.9.

$$\mathbf{H}= \left[\begin{array}{c}
1000\,0000\,0000\,1000\,1000\,1000\,1000\,1000\,1000\,1000\,1000\,1000\,1000\,1000\,1000\,1000\,1000\,1000\,1000\,10 \\
0100\,0000\,0000\,0100\,0100\,0100\,0100\,0100\,0100\,0100\,0100\,0100\,0100\,0100\,0100\,0100\,0100\,0100\,0100\,01 \\
0010\,0000\,0000\,0010\,0010\,0010\,0010\,0010\,0010\,0010\,0010\,0010\,0010\,0010\,0010\,0010\,0010\,0010\,0010\,00 \\
0001\,0000\,0000\,0001\,0001\,0001\,0001\,0001\,0001\,0001\,0001\,0001\,0001\,0001\,0001\,0001\,0001\,0001\,0001\,00 \\
\text{one row deleted} \\
\overline{0000}\;\overline{1000}\;\overline{0000}\;\overline{1000}\;\overline{0000}\;\overline{0001}\;\overline{0010}\;\overline{0100}\;\overline{1001}\;\overline{0010}\;\overline{0101}\;\overline{1011}\;\overline{0110}\;\overline{1100}\;\overline{1001}\;\overline{0011}\;\overline{0111}\;\overline{1111}\;\overline{1111}\;\overline{11} \\
0000\,0100\,0000\,0100\,1000\,0000\,0001\,0010\,0100\,1001\,0010\,0101\,1011\,0110\,1100\,1001\,0011\,0111\,1111\,11 \\
0000\,0010\,0000\,0010\,0100\,1001\,0010\,0101\,1011\,0110\,1100\,1001\,0011\,0111\,1111\,1111\,1110\,1100\,1000\,00 \\
0000\,0001\,0000\,0001\,0010\,0100\,1001\,0010\,0101\,1011\,0110\,1100\,1001\,0011\,0111\,1111\,1111\,1110\,1100\,10 \\
0000\,0000\,0000\,0000\,0001\,0010\,0100\,1001\,0010\,0101\,1011\,0110\,1100\,1001\,0011\,0111\,1111\,1111\,1110\,11 \\
\overline{0000}\;\overline{0000}\;\overline{1000}\;\overline{1000}\;\overline{0001}\;\overline{0100}\;\overline{0010}\;\overline{1011}\;\overline{1100}\;\overline{0011}\;\overline{1111}\;\overline{1110}\;\overline{1000}\;\overline{0011}\;\overline{1101}\;\overline{0111}\;\overline{1101}\;\overline{0101}\;\overline{0100}\;\overline{00} \\
0000\,0000\,0100\,0100\,0000\,0010\,1001\,0101\,0110\,1001\,0111\,1111\,1100\,0001\,0110\,1011\,1110\,1010\,1010\,10 \\
0000\,0000\,0010\,0010\,1001\,0101\,0110\,1001\,0111\,1111\,1100\,0001\,0110\,1011\,1110\,1010\,1010\,1000\,0001\,01 \\
0000\,0000\,0001\,0001\,0100\,0010\,1011\,1100\,0011\,1111\,1110\,1000\,0011\,1101\,0111\,1101\,0101\,0100\,0000\,00 \\
0000\,0000\,0000\,0000\,0010\,1001\,0101\,0110\,1001\,0111\,1111\,1100\,0001\,0110\,1011\,1110\,1010\,1010\,1000\,00
\end{array}\right]$$

Figure 5.14 *(136, 122) S4EC-D4ED code.*

TABLE 5.9 Error Detection Capability of the (78, 64) S4EC-D4ED Chen Code Shown in Figure 5.14

Errors	Error detection capability (%)
Single-bit plus double-byte errors	98.41
Triple-byte errors	98.37

The existing SbEC-DbED codes with $b = 4$ bits and $K = 64$ bits require at least 16 check bits, whereas this code requires 14 check bits. This type of SbEC-DbED code can be constructed from the existing SbEC-DbED codes by following the previous procedure. Some of the converted codes designed by this technique are more efficient than the existing known codes. Table 5.10 provides a list of some efficient converted SbEC-DbED codes, compared to other best known SbEC-DbED codes.

5.3 SINGLE-BYTE ERROR CORRECTING AND SINGLE *p*-BYTE WITHIN A BLOCK ERROR DETECTING (SbEC-S$_{p\times b/B}$ED) CODES

Present-day high-density RAM chips have a multi-bank architecture. Each bank usually has a number of memory subarrays that are somewhat separate from one another [NUMA89]. It is therefore advantageous to consider the entire chip output as a B-bit block and subarray output as a b-bit byte. Figure 5.15 shows how this concept's organization corresponds to the bit, byte, and block of a codeword.

For the chip's organization we need to develop suitable byte error control codes. We discuss here such codes that can correct single-byte errors caused by single-subarray faults and that can detect multiple-byte (p-byte) errors in a block caused by p-subarray faults in a chip, where $2 \leq p \leq B/b$, denoted as SbEC-S$_{p\times b/B}$ED codes.

This class of codes is important for the recent large capacity memory systems using high-density RAM chips with wide I/O data, such as 8-bit, 16-bit, and 32-bit DRAM chips.

5.3.1 Code Conditions and Bounds

First, we consider the necessary and sufficient conditions of the SbEC-S$_{p\times b/B}$ED code.

TABLE 5.10 Converted SbEC-DbED Codes Compared to Existing Codes

Converted Chen codes					Existing codes
Byte size b (bits)	Code length n (bytes)	Number of check bits R	Details of R	Original byte size ($>b$) (bits)	Code length n (bytes)[a]
2	65	11	$2+3+3+3$	3	41
2	1,025	17	$2+5+5+5$	5	640
3	18	11	$3+4+4$	4	10
3	257	15	$3+4+4+4$	4	133
3	1,025	18	$3+5+5+5$	5	650
4	34	14	$4+5+5$	5	18
4	1,025	19	$4+5+5+5$	5	257
5	130	19	$5+7+7$	7	34
6	130	20	$6+7+7$	7	66

[a] For byte size b and check-bit length $b \cdot \lfloor R/b \rfloor$.

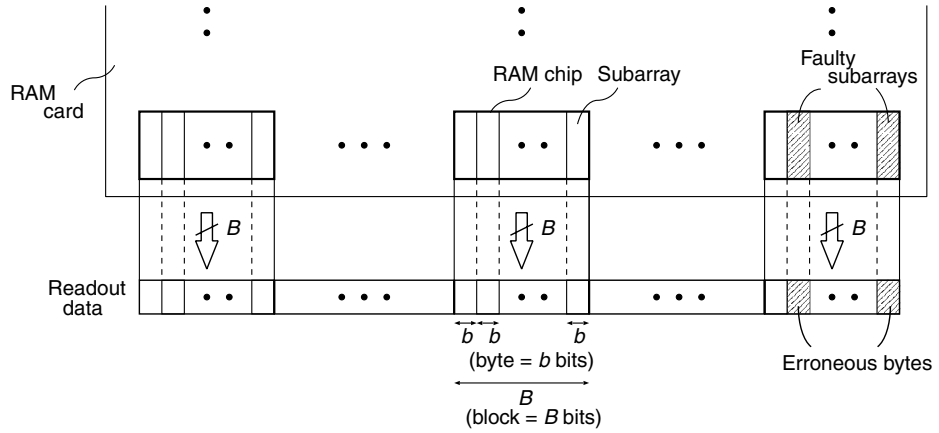


Figure 5.15 *Bits, bytes, and blocks of the corresponding organization in each high-density RAM chip with multiple subarrays.*

Theorem 5.14 *The null space of* $\boldsymbol{H}$ *is an SbEC-S*$_{p\times b/B}$*ED code if and only if:*

1. $E \cdot \boldsymbol{H}^T \neq 0$ *for all* $E \in \{\boldsymbol{E}_b \cup \boldsymbol{E}_{p\times b/B}\}$,
2. $E_1 \cdot \boldsymbol{H}^T \neq E_2 \cdot \boldsymbol{H}^T$ *for all* $E_1, E_2 \in \boldsymbol{E}_b, E_1 \neq E_2$,
3. $E_1 \cdot \boldsymbol{H}^T \neq E_3 \cdot \boldsymbol{H}^T$ *for all* $E_1 \in \boldsymbol{E}_b$, *and for all* $E_3 \in \boldsymbol{E}_{p\times b/B}$,

where $\boldsymbol{E}_b$ *is a set of single-byte errors,* $\boldsymbol{E}_{p\times b/B}$ *is a set of p-byte errors in a block, where* $\boldsymbol{E}_b \cap \boldsymbol{E}_{p\times b/B} = \phi$, $2 \leq p \leq B/b$, *and* $\boldsymbol{H}^T$ *denotes the transpose of matrix* $\boldsymbol{H}$.

This theorem can be easily proved, and therefore the proof is omitted.

Next we consider the bound of this code.

Theorem 5.15 *An SbEC-S*$_{p\times b/B}$*ED code requires at least* $b(p+1)$ *check bits.*

Theorem 5.16 *A binary* $(N, N-R)$ *SbEC-S*$_{p\times b/B}$*ED code exists only if*

$$2^R \geq \frac{N}{b}(2^b - 1) + \left(\frac{B}{b} - p + 1\right)\left(2^{pb} - 1 - p(2^b - 1)\right) + 1.$$

It is left to the reader to prove these theorems.

5.3.2 Design for S*b*EC-S$_{p\times b/B}$ED Codes

1. Design Method I

Theorem 5.17 *The null space of*

$$\boldsymbol{H}_n = \left[\begin{array}{c|c|c|c|c|c} \boldsymbol{I}\cdots\boldsymbol{I} & \boldsymbol{T}\cdots\boldsymbol{T} & \cdots & \boldsymbol{T}^i\cdots\boldsymbol{T}^i & \cdots & \boldsymbol{T}^{q-2}\cdots\boldsymbol{T}^{q-2} \;\; \boldsymbol{O}\cdots\boldsymbol{O} \\ \boldsymbol{H}_{n-1} & \boldsymbol{H}_{n-1} & & \boldsymbol{H}_{n-1} & & \boldsymbol{H}_{n-1} \;\; \boldsymbol{H}_{n-1} \end{array}\right]$$

is an SbEC-$S_{p\times b/B}$ED code with check-bit length R and code length in bits $N = B \cdot \lfloor N_{RS}/B \rfloor \cdot 2^{b(R/b-p-1)}$, *where* $n = R/b - p - 1$. *Here* $\boldsymbol{H}_0$ *is an* $\boldsymbol{H}$ *matrix of RS code with minimum distance* $d = p + 2$, $2 \le p \le B/b$, $\boldsymbol{I}$ *is a* $b \times b$ *identity matrix,* $\boldsymbol{O}$ *is a* $b \times b$ *zero matrix,* $\boldsymbol{T}$ *is a companion matrix defined by the binary primitive polynomial with degree b,* $0 \le i \le q - 2$, $q = 2^b$, $\lfloor x \rfloor$ *means the largest integer smaller than or equal to x, and* N_{RS} *is a code length (in bits) of distance-4 RS codes.*

Proof Since $\mathbf{H}_0$ is a parity-check matrix of RS code with distance $d = p + 2$, this satisfies the code function of SbEC-$S_{p\times b/B}$ED. Next assume that $\mathbf{H}_{n-1}$ is a parity-check matrix of an SbEC-$S_{p\times b/B}$ED code. Then we need to prove that $\mathbf{H}_n$ is also a parity-check matrix of an SbEC-$S_{p\times b/B}$ED code. It is apparent that $\mathbf{H}_n$ has the code function of SbEC. Also $\mathbf{H}_n$ has the code function of $S_{p\times b/B}$ED because every $\mathbf{H}_{n-1}$ in $\mathbf{H}_n$ has this function from the assumption. Therefore $\mathbf{H}_n$ is a parity-check matrix of an SbEC-$S_{p\times b/B}$ED code, and $n = R/b - p - 1$.

As for the maximum code length, $B \cdot \lfloor N_{RS}/B \rfloor$ shows the length of $\mathbf{H}_0$, and $2^{b(R/b-p-1)}$ shows the maximum number of matrix elements $\mathbf{I}, \mathbf{T}, \ldots, \mathbf{T}^{q-2}, \mathbf{O}$. Therefore the maximum code length in bits can be expressed as $N = B \cdot \lfloor N_{RS}/B \rfloor \cdot 2^{b(R/b-p-1)}$.

Q.E.D.

Example 5.11

The following shows (1024, 1008) S4EC-$S_{2\times 4/16}$ED code with $R = 16$, $b = 4$, $B = 16$, and $p = 2$, where $\mathbf{H}_{\mathrm{RS}}$ is a parity-check matrix of an S4EC-D4ED RS code:

$$\mathbf{H} = \left[\begin{array}{ccc:ccc:c:ccc:ccc} \mathbf{I} & \cdots & \mathbf{I} & \mathbf{T} & \cdots & \mathbf{T} & & \mathbf{T}^{14} & \cdots & \mathbf{T}^{14} & \mathbf{O} & \cdots & \mathbf{O} \\ & & & & & & \cdots & & & & & & \\ \mathbf{H}_{\mathrm{RS}} & & & \mathbf{H}_{\mathrm{RS}} & & & & \mathbf{H}_{\mathrm{RS}} & & & \mathbf{H}_{\mathrm{RS}} & & \end{array}\right],$$

$$\mathbf{H}_{\mathrm{RS}} = \left[\begin{array}{cccc|cccc|cccc|cccc} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^3 & \mathbf{T}^4 & \mathbf{T}^5 & \mathbf{T}^6 & \mathbf{T}^7 & \mathbf{T}^8 & \mathbf{T}^9 & \mathbf{T}^{10} & \mathbf{T}^{11} & \mathbf{T}^{12} & \mathbf{T}^{13} & \mathbf{T}^{14} & \mathbf{O} \\ \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^4 & \mathbf{T}^6 & \mathbf{T}^8 & \mathbf{T}^{10} & \mathbf{T}^{12} & \mathbf{T}^{14} & \mathbf{T} & \mathbf{T}^3 & \mathbf{T}^5 & \mathbf{T}^7 & \mathbf{T}^9 & \mathbf{T}^{11} & \mathbf{T}^{13} & \mathbf{O} \end{array}\right].$$

$\mathbf{T}$: defined by $\mathbf{g}(x) = x^4 + x + 1$.

More efficient codes can be designed by using distance-4 converted Chen code mentioned in the previous subsection.

For the practical code parameters of $b = 4$ bits, $B = 16$ bits, $p = 2$, and $K = 64$ bits, the S4EC-$S_{2\times 4/16}$ED code with check-bit length 16 can be designed from this example. Here the S4EC-$S_{2\times 4/16}$ED code with check-bit length $R = 12$ shown by $\mathbf{H}_{\mathrm{RS}}$ in Example 5.11 can be lengthened by adding three columns to $\mathbf{H}_{\mathrm{RS}}$. That is, the following matrix presents the (76, 64) S4EC-$S_{2\times 4/16}$ED code, where $\mathbf{T}$ is a companion matrix defined by the primitive polynomial $\mathbf{g}(x) = x^4 + x + 1$:

$$\mathbf{H}_{\mathrm{RS}} = \left[\begin{array}{cccc|cccc|cccc|cccc|ccc} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^3 & \mathbf{T}^4 & \mathbf{T}^5 & \mathbf{T}^6 & \mathbf{T}^7 & \mathbf{T}^8 & \mathbf{T}^9 & \mathbf{T}^{10} & \mathbf{T}^{11} & \mathbf{T}^{12} & \mathbf{T}^{13} & \mathbf{T}^{14} & \mathbf{O} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 \\ \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^4 & \mathbf{T}^6 & \mathbf{T}^8 & \mathbf{T}^{10} & \mathbf{T}^{12} & \mathbf{T}^{14} & \mathbf{T} & \mathbf{T}^3 & \mathbf{T}^5 & \mathbf{T}^7 & \mathbf{T}^9 & \mathbf{T}^{11} & \mathbf{T}^{13} & \mathbf{O} & \mathbf{T}^{10} & \mathbf{T}^{12} & \mathbf{T}^2 \end{array}\right]. \tag{5.37}$$

The last three columns are generated by computer search. This code has been practically applied to the high-speed memory systems using RAM chips with 16-bit I/O data.

2. Design Method II

More efficient and sophisticated code design method of an SbEC-S$_{p\times b/B}$ED code is presented here.

Theorem 5.18 [JOHJ97] *The null space of the following matrix* $\boldsymbol{H}$ *is an SbEC-*$S_{p\times b/B}$*ED code:*

$$\begin{aligned}\boldsymbol{H} &= \left[\begin{array}{c|c} \boldsymbol{H}_1 & \boldsymbol{H}_2 \end{array}\right] \\ &= \left[\begin{array}{cccc|cccc} \boldsymbol{H}_{1,1} & \boldsymbol{H}_{1,2} & \cdots & \boldsymbol{H}_{1,m} & \boldsymbol{H}_{2,1} & \boldsymbol{H}_{2,2} & \cdots & \boldsymbol{H}_{2,m} \end{array}\right],\end{aligned}$$

where

$$\boldsymbol{H}_{1,i} = \begin{array}{c} \overleftarrow{\qquad\qquad B \qquad\qquad}\overrightarrow{\qquad} \\ \left[\begin{array}{cccc} \boldsymbol{I} & \boldsymbol{I} & \cdots & \boldsymbol{I} \\ \boldsymbol{T}^{\alpha_{i,1}} & \boldsymbol{T}^{\alpha_{i,2}} & \cdots & \boldsymbol{T}^{\alpha_{i,B/b}} \\ \vdots & \vdots & & \vdots \\ \boldsymbol{T}^{(p-1)\alpha_{i,1}} & \boldsymbol{T}^{(p-1)\alpha_{i,2}} & \cdots & \boldsymbol{T}^{(p-1)\alpha_{i,B/b}} \\ \boldsymbol{T}^{p\alpha_{i,1}} & \boldsymbol{T}^{p\alpha_{i,2}} & \cdots & \boldsymbol{T}^{p\alpha_{i,B/b}} \end{array}\right] \end{array},$$

$$\boldsymbol{H}_{2,i} = \begin{array}{c} \overleftarrow{\qquad\qquad B \qquad\qquad}\overrightarrow{\qquad} \\ \left[\begin{array}{cccc} \boldsymbol{I} & \boldsymbol{I} & \cdots & \boldsymbol{I} \\ \boldsymbol{T}^{\alpha_{i,1}} & \boldsymbol{T}^{\alpha_{i,2}} & \cdots & \boldsymbol{T}^{\alpha_{i,B/b}} \\ \vdots & \vdots & & \vdots \\ \boldsymbol{T}^{(p-1)\alpha_{i,1}} & \boldsymbol{T}^{(p-1)\alpha_{i,2}} & \cdots & \boldsymbol{T}^{(p-1)\alpha_{i,B/b}} \\ \boldsymbol{T}^{p\alpha_{i,1}}+\boldsymbol{I} & \boldsymbol{T}^{p\alpha_{i,2}}+\boldsymbol{I} & \cdots & \boldsymbol{T}^{p\alpha_{i,B/b}}+\boldsymbol{I} \end{array}\right] \end{array},$$

$i = 1, 2, \ldots, m, \Omega_i = \{\boldsymbol{T}^{\alpha_{i,1}}, \boldsymbol{T}^{\alpha_{i,2}}, \ldots, \boldsymbol{T}^{\alpha_{i,B/b}}\}$, $|\Omega_i| = B/b$, $\boldsymbol{T}^{\alpha_{i,j}}$ *is a* $b \times b$ *companion matrix included in* $GF(2^b), j = 1, 2, \ldots, B/b$, *and where for* $\{\boldsymbol{T}^{\alpha_1}, \boldsymbol{T}^{\alpha_2}, \ldots, \boldsymbol{T}^{\alpha_p}\} \in \Omega_i$ *and* $\boldsymbol{T}^{\beta} \in \Omega_j$ $(i \neq j)$, *the following relation is satisfied:*

$$(\boldsymbol{T}^{\alpha_1} + \boldsymbol{T}^{\beta})(\boldsymbol{T}^{\alpha_2} + \boldsymbol{T}^{\beta}) \cdots (\boldsymbol{T}^{\alpha_p} + \boldsymbol{T}^{\beta}) \neq \boldsymbol{I}. \tag{5.38}$$

Proof It is apparent that the matrix $\mathbf{H}$ indicated in this theorem satisfies SbEC function. The matrix with upper p rows in $\mathbf{H}$ is equal to the parity-check matrix of the RS code with distance $p+1$. That is, each submatrix of $\mathbf{H}_{1,i}$, and $\mathbf{H}_{2,i}$ has function of $p-1$ bytes error detection as well as SbEC function in each block with length B bytes. In addition to this, the following mentions that the matrix $\mathbf{H}$ has the function of p-byte error detection.

1. The matrix $\mathbf{H}_1$ is an RS code with distance $p+2$, and hence the syndrome by single-byte errors is distinct to that by p-byte errors in the block. Next assume that the syndrome by single-byte errors in $\mathbf{H}_2$ is equal to that by p-byte errors in $\mathbf{H}_2$. Then for any

distinct $p+1$ elements in $\mathbf{H}_2$, such as $\mathbf{T}^{\alpha_1}, \mathbf{T}^{\alpha_2}, \ldots, \mathbf{T}^{\alpha_p}, \mathbf{T}^{\alpha_{p+1}}$, the following relations hold:

$$\begin{gathered} E_1 + E_2 + \cdots + E_p + E_{p+1} = 0 \\ E_1\mathbf{T}^{\alpha_1} + E_2\mathbf{T}^{\alpha_2} + \cdots + E_p\mathbf{T}^{\alpha_p} + E_{p+1}\mathbf{T}^{\alpha_{p+1}} = 0 \\ \vdots \\ E_1(\mathbf{T}^{p\alpha_1} + \mathbf{I}) + E_2(\mathbf{T}^{p\alpha_2} + \mathbf{I}) + \cdots + E_p(\mathbf{T}^{p\alpha_p} + \mathbf{I}) + E_{p+1}(\mathbf{T}^{p\alpha_{p+1}} + \mathbf{I}) = 0. \end{gathered}$$

These relations are reduced to

$$E_1 \cdot (\mathbf{T}^{\alpha_1} + \mathbf{T}^{\alpha_2})(\mathbf{T}^{\alpha_1} + \mathbf{T}^{\alpha_3}) \cdots (\mathbf{T}^{\alpha_p} + \mathbf{T}^{\alpha_{p+1}}) = 0,$$

which does not hold because $\mathbf{T}^{\alpha_1} \neq \mathbf{T}^{\alpha_2} \neq \cdots \neq \mathbf{T}^{\alpha_p} \neq \mathbf{T}^{\alpha_{p+1}}$. Therefore the syndrome by single-byte errors in $\mathbf{H}_2$ is distinct to that by p-byte errors in $\mathbf{H}_2$.

2. Assume that the syndrome caused by p-byte errors in $\mathbf{H}_1$ is equal to that caused by single-byte errors in $\mathbf{H}_2$.

$$\begin{gathered} E_1 + E_2 + \cdots + E_p + E_{p+1} = 0 \\ E_1\mathbf{T}^{\alpha_1} + E_2\mathbf{T}^{\alpha_2} + \cdots + E_p\mathbf{T}^{\alpha_p} + E_{p+1}\mathbf{T}^{\beta} = 0 \\ \vdots \\ E_1\mathbf{T}^{p\alpha_1} + E_2\mathbf{T}^{p\alpha_2} + \cdots + E_p\mathbf{T}^{p\alpha_p} + E_{p+1}(\mathbf{T}^{p\beta} + \mathbf{I}) = 0, \end{gathered}$$

where $\mathbf{T}^{\alpha_1}, \mathbf{T}^{\alpha_2}, \ldots, \mathbf{T}^{\alpha_p}$ are elements in $\mathbf{H}_1$, and $\mathbf{T}^{\beta}$ is an element in $\mathbf{H}_2$. These relations are reduced to

$$E_1 \cdot \{(\mathbf{T}^{\alpha_1} + \mathbf{T}^{\beta})(\mathbf{T}^{\alpha_2} + \mathbf{T}^{\beta}) \cdots (\mathbf{T}^{\alpha_p} + \mathbf{T}^{\beta}) + \mathbf{I}\} = 0,$$

which contradicts the relation (5.38). Therefore the syndrome by p-byte errors in $\mathbf{H}_1$ is distinct to that by single-byte errors in $\mathbf{H}_2$. This result also holds for the relation between p-byte errors in $\mathbf{H}_2$ and single-byte errors in $\mathbf{H}_1$.

From the discussion above, the matrix $\mathbf{H}$ satisfies the code functions of $\mathrm{S}_{p\times b/B}\mathrm{ED}$ as well as SbEC. Q.E.D.

Algorithm for Finding Block Elements Here we provide an algorithm used to find the elements in $GF(2^b)$ that satisfy the relation (5.38) and to obtain the sets $\Omega_1, \Omega_2, \ldots, \Omega_m$, each with B/b elements. First, we prepare the table in which elements $\mathbf{T}^{\alpha_2}, \mathbf{T}^{\alpha_3}, \ldots, \mathbf{T}^{\alpha_p}$, satisfy the relation (5.38) for an element $\mathbf{T}^{\alpha_1}$ and for any element $\mathbf{T}^{\beta}$. Table 5.11 shows an example of the elements $\mathbf{T}^{\alpha_1}$ and $\mathbf{T}^{\alpha_2}$ satisfying the relation (5.38), with parameters of $p = 2$ bytes and $b = 4$ bits. The table lists $\mathbf{I}, \mathbf{T}^1, \mathbf{T}^2, \ldots, \mathbf{T}^{14}, \mathbf{O}$ in order for $\mathbf{T}^{\alpha_1}$, and gives the elements for $\mathbf{T}^{\alpha_2}$, where $\alpha_1 < \alpha_2$, which satisfy the relation (5.38) for any $\mathbf{T}^{\beta}$. Then we determine the B/b block elements from the following algorithm:

Step 1. *All elements are not marked at the first stage. The determined elements are marked in the following steps.*

TABLE 5.11 Elements Satisfying Relation (5.38) for $p = 2$ and $b = 4$

$\mathbf{T}^{\alpha_1}$	$\mathbf{T}^{\alpha_2}$							
$\mathbf{I}$	$\mathbf{T}$	$\mathbf{T}^2$	$\mathbf{T}^4$	$\mathbf{T}^7$	$\mathbf{T}^8$	$\mathbf{T}^{11}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$
$\mathbf{T}$	$\mathbf{T}^3$	$\mathbf{T}^5$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{13}$	$\mathbf{O}$	
$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{O}$	
$\mathbf{T}^3$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{13}$	$\mathbf{O}$		
$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^7$	$\mathbf{T}^{10}$	$\mathbf{T}^{12}$	$\mathbf{T}^{14}$	$\mathbf{O}$	
$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{11}$	$\mathbf{T}^{14}$			
$\mathbf{T}^6$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{14}$	$\mathbf{O}$				
$\mathbf{T}^7$	$\mathbf{T}^{10}$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{14}$				
$\mathbf{T}^8$	$\mathbf{T}^9$	$\mathbf{T}^{10}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{T}^{14}$	$\mathbf{O}$		
$\mathbf{T}^9$	$\mathbf{T}^{11}$	$\mathbf{T}^{12}$	$\mathbf{T}^{14}$	$\mathbf{O}$				
$\mathbf{T}^{10}$	$\mathbf{T}^{12}$	$\mathbf{T}^{13}$						
$\mathbf{T}^{11}$	$\mathbf{T}^{13}$							
$\mathbf{T}^{12}$	$\mathbf{T}^{13}$	$\mathbf{O}$						
$\mathbf{T}^{13}$	$\mathbf{T}^{14}$							
$\mathbf{T}^{14}$								
$\mathbf{O}$								

Source: [JOHJ97]. © 1997 IEICE Japan.

Step 2. *Determine the element $\boldsymbol{T}^{\alpha_2}$ with the largest exponent. Next, find a row in the table that includes $\boldsymbol{T}^{\alpha_2}$, and determine the nonmarked element $\boldsymbol{T}^{\alpha_1}$ with the largest exponent in the row. The result should be a set $\Omega = \{\boldsymbol{T}^{\alpha_1}, \boldsymbol{T}^{\alpha_2}\}$. If a nonmarked elements cannot be found, then stop.*

Step 3. *Find the rows that include all the elements of Ω in the $\boldsymbol{T}^{\alpha_2}$ column, and determine the nonmarked element $\boldsymbol{T}^{\alpha_1}$ with the largest exponent. Include the element $\boldsymbol{T}^{\alpha_1}$ in Ω, that is, $\Omega = \Omega \cup \boldsymbol{T}^{\alpha_1}$. If such an element cannot be found, then go to step 5.*

Step 4. *If the number of elements in Ω (i.e., $|\Omega|$) is equal to B/b, then $\Omega = \phi$ and go to step 1. If not, go to step 3.*

Step 5. *Remove the element last determined from Ω, and then go to step 3.*

Table 5.12 presents the relation between the elements $\mathbf{T}^{\alpha_1}$ and the elements ($\mathbf{T}^{\alpha_2}$, $\mathbf{T}^{\alpha_3}$) satisfying the relation (5.38) with parameters of $p = 3$ bytes and $b = 4$ bits.

Example 5.12 [JOHJ97]

With using Table 5.11, we try to find the block elements for $p = 2$ bytes, $b = 4$ bits, and $B = 16$ bits.

TABLE 5.12 Elements Satisfying Relation (5.38) for $p = 3$ and $b = 4$

$\mathbf{T}^{\alpha_1}$	$(\mathbf{T}^{\alpha_2}, \mathbf{T}^{\alpha_3})$							
$\mathbf{I}$	$(\mathbf{T}^1, \mathbf{T}^2)$	$(\mathbf{T}^1, \mathbf{T}^3)$	$(\mathbf{T}^1, \mathbf{T}^5)$	$(\mathbf{T}^1, \mathbf{T}^7)$	$(\mathbf{T}^1, \mathbf{T}^8)$	$(\mathbf{T}^1, \mathbf{T}^{10})$	$(\mathbf{T}^2, \mathbf{T}^4)$	$(\mathbf{T}^2, \mathbf{T}^5)$
	$(\mathbf{T}^2, \mathbf{T}^6)$	$(\mathbf{T}^2, \mathbf{T}^{10})$	$(\mathbf{T}^2, \mathbf{T}^{14})$	$(\mathbf{T}^3, \mathbf{T}^7)$	$(\mathbf{T}^3, \mathbf{T}^{10})$	$(\mathbf{T}^3, \mathbf{T}^{11})$	$(\mathbf{T}^3, \mathbf{T}^{14})$	$(\mathbf{T}^3, \mathbf{O})$
	$(\mathbf{T}^4, \mathbf{T}^5)$	$(\mathbf{T}^4, \mathbf{T}^8)$	$(\mathbf{T}^4, \mathbf{T}^{10})$	$(\mathbf{T}^4, \mathbf{T}^{12})$	$(\mathbf{T}^4, \mathbf{T}^{13})$	$(\mathbf{T}^5, \mathbf{T}^6)$	$(\mathbf{T}^5, \mathbf{T}^7)$	$(\mathbf{T}^5, \mathbf{T}^8)$
	$(\mathbf{T}^5, \mathbf{T}^9)$	$(\mathbf{T}^5, \mathbf{T}^{13})$	$(\mathbf{T}^6, \mathbf{T}^7)$	$(\mathbf{T}^6, \mathbf{T}^{13})$	$(\mathbf{T}^6, \mathbf{T}^{14})$	$(\mathbf{T}^6, \mathbf{O})$	$(\mathbf{T}^7, \mathbf{T}^9)$	$(\mathbf{T}^7, \mathbf{O})$
	$(\mathbf{T}^8, \mathbf{T}^9)$	$(\mathbf{T}^8, \mathbf{T}^{10})$	$(\mathbf{T}^8, \mathbf{T}^{11})$	$(\mathbf{T}^9, \mathbf{T}^{11})$	$(\mathbf{T}^9, \mathbf{T}^{13})$	$(\mathbf{T}^9, \mathbf{O})$	$(\mathbf{T}^{10}, \mathbf{T}^{11})$	$(\mathbf{T}^{10}, \mathbf{T}^{12})$
	$(\mathbf{T}^{10}, \mathbf{T}^{14})$	$(\mathbf{T}^{11}, \mathbf{T}^{12})$	$(\mathbf{T}^{11}, \mathbf{O})$	$(\mathbf{T}^{12}, \mathbf{T}^{13})$	$(\mathbf{T}^{12}, \mathbf{T}^{14})$	$(\mathbf{T}^{12}, \mathbf{O})$	$(\mathbf{T}^{13}, \mathbf{O})$	$(\mathbf{T}^{14}, \mathbf{O})$
$\mathbf{T}^1$	$(\mathbf{T}^2, \mathbf{T}^5)$	$(\mathbf{T}^2, \mathbf{T}^6)$	$(\mathbf{T}^2, \mathbf{T}^7)$	$(\mathbf{T}^2, \mathbf{T}^9)$	$(\mathbf{T}^2, \mathbf{T}^{10})$	$(\mathbf{T}^2, \mathbf{T}^{13})$	$(\mathbf{T}^2, \mathbf{O})$	$(\mathbf{T}^3, \mathbf{T}^4)$
	$(\mathbf{T}^3, \mathbf{T}^7)$	$(\mathbf{T}^3, \mathbf{T}^8)$	$(\mathbf{T}^3, \mathbf{T}^{12})$	$(\mathbf{T}^3, \mathbf{T}^{14})$	$(\mathbf{T}^4, \mathbf{T}^6)$	$(\mathbf{T}^4, \mathbf{T}^7)$	$(\mathbf{T}^4, \mathbf{T}^9)$	$(\mathbf{T}^4, \mathbf{T}^{11})$
	$(\mathbf{T}^4, \mathbf{T}^{12})$	$(\mathbf{T}^4, \mathbf{T}^{13})$	$(\mathbf{T}^4, \mathbf{T}^{14})$	$(\mathbf{T}^5, \mathbf{T}^8)$	$(\mathbf{T}^5, \mathbf{T}^{13})$	$(\mathbf{T}^5, \mathbf{T}^{14})$	$(\mathbf{T}^5, \mathbf{O})$	$(\mathbf{T}^6, \mathbf{T}^9)$
	$(\mathbf{T}^6, \mathbf{T}^{11})$	$(\mathbf{T}^6, \mathbf{T}^{13})$	$(\mathbf{T}^6, \mathbf{O})$	$(\mathbf{T}^7, \mathbf{T}^9)$	$(\mathbf{T}^7, \mathbf{T}^{13})$	$(\mathbf{T}^8, \mathbf{T}^{10})$	$(\mathbf{T}^8, \mathbf{T}^{11})$	$(\mathbf{T}^8, \mathbf{T}^{12})$
	$(\mathbf{T}^8, \mathbf{T}^{14})$	$(\mathbf{T}^8, \mathbf{O})$	$(\mathbf{T}^9, \mathbf{T}^{10})$	$(\mathbf{T}^9, \mathbf{T}^{12})$	$(\mathbf{T}^{10}, \mathbf{T}^{12})$	$(\mathbf{T}^{10}, \mathbf{O})$	$(\mathbf{T}^{11}, \mathbf{T}^{12})$	$(\mathbf{T}^{11}, \mathbf{T}^{14})$
	$(\mathbf{T}^{11}, \mathbf{O})$	$(\mathbf{T}^{13}, \mathbf{T}^{14})$						
$\mathbf{T}^2$	$(\mathbf{T}^3, \mathbf{T}^4)$	$(\mathbf{T}^3, \mathbf{T}^5)$	$(\mathbf{T}^3, \mathbf{T}^8)$	$(\mathbf{T}^3, \mathbf{T}^9)$	$(\mathbf{T}^3, \mathbf{T}^{12})$	$(\mathbf{T}^3, \mathbf{T}^{14})$	$(\mathbf{T}^4, \mathbf{T}^5)$	$(\mathbf{T}^4, \mathbf{T}^{10})$
	$(\mathbf{T}^4, \mathbf{T}^{11})$	$(\mathbf{T}^4, \mathbf{T}^{12})$	$(\mathbf{T}^4, \mathbf{T}^{14})$	$(\mathbf{T}^4, \mathbf{O})$	$(\mathbf{T}^5, \mathbf{T}^9)$	$(\mathbf{T}^5, \mathbf{O})$	$(\mathbf{T}^6, \mathbf{T}^8)$	$(\mathbf{T}^6, \mathbf{T}^9)$
	$(\mathbf{T}^6, \mathbf{T}^{13})$	$(\mathbf{T}^6, \mathbf{T}^{14})$	$(\mathbf{T}^7, \mathbf{T}^8)$	$(\mathbf{T}^7, \mathbf{T}^9)$	$(\mathbf{T}^7, \mathbf{T}^{12})$	$(\mathbf{T}^7, \mathbf{T}^{13})$	$(\mathbf{T}^7, \mathbf{O})$	$(\mathbf{T}^8, \mathbf{T}^9)$
	$(\mathbf{T}^8, \mathbf{T}^{11})$	$(\mathbf{T}^8, \mathbf{T}^{12})$	$(\mathbf{T}^8, \mathbf{T}^{13})$	$(\mathbf{T}^8, \mathbf{T}^{14})$	$(\mathbf{T}^{10}, \mathbf{T}^{11})$	$(\mathbf{T}^{10}, \mathbf{T}^{13})$	$(\mathbf{T}^{10}, \mathbf{O})$	$(\mathbf{T}^{11}, \mathbf{T}^{12})$
	$(\mathbf{T}^{11}, \mathbf{T}^{13})$	$(\mathbf{T}^{11}, \mathbf{T}^{14})$	$(\mathbf{T}^{12}, \mathbf{O})$					
$\mathbf{T}^3$	$(\mathbf{T}^4, \mathbf{T}^6)$	$(\mathbf{T}^4, \mathbf{T}^{10})$	$(\mathbf{T}^4, \mathbf{T}^{12})$	$(\mathbf{T}^4, \mathbf{T}^{14})$	$(\mathbf{T}^5, \mathbf{T}^9)$	$(\mathbf{T}^5, \mathbf{T}^{10})$	$(\mathbf{T}^5, \mathbf{T}^{11})$	$(\mathbf{T}^5, \mathbf{T}^{14})$
	$(\mathbf{T}^5, \mathbf{O})$	$(\mathbf{T}^6, \mathbf{T}^7)$	$(\mathbf{T}^6, \mathbf{T}^9)$	$(\mathbf{T}^6, \mathbf{T}^{10})$	$(\mathbf{T}^6, \mathbf{T}^{11})$	$(\mathbf{T}^6, \mathbf{T}^{12})$	$(\mathbf{T}^7, \mathbf{T}^{11})$	$(\mathbf{T}^7, \mathbf{T}^{12})$
	$(\mathbf{T}^7, \mathbf{T}^{13})$	$(\mathbf{T}^8, \mathbf{T}^{12})$	$(\mathbf{T}^8, \mathbf{T}^{13})$	$(\mathbf{T}^8, \mathbf{T}^{14})$	$(\mathbf{T}^8, \mathbf{O})$	$(\mathbf{T}^9, \mathbf{T}^{11})$	$(\mathbf{T}^9, \mathbf{T}^{12})$	$(\mathbf{T}^9, \mathbf{T}^{13})$
	$(\mathbf{T}^{10}, \mathbf{T}^{11})$	$(\mathbf{T}^{10}, \mathbf{T}^{14})$	$(\mathbf{T}^{11}, \mathbf{T}^{13})$	$(\mathbf{T}^{11}, \mathbf{O})$	$(\mathbf{T}^{12}, \mathbf{T}^{13})$	$(\mathbf{T}^{13}, \mathbf{O})$	$(\mathbf{T}^{14}, \mathbf{O})$	
$\mathbf{T}^4$	$(\mathbf{T}^5, \mathbf{T}^7)$	$(\mathbf{T}^5, \mathbf{T}^8)$	$(\mathbf{T}^5, \mathbf{T}^{11})$	$(\mathbf{T}^5, \mathbf{O})$	$(\mathbf{T}^6, \mathbf{T}^8)$	$(\mathbf{T}^6, \mathbf{T}^9)$	$(\mathbf{T}^6, \mathbf{T}^{10})$	$(\mathbf{T}^6, \mathbf{T}^{13})$
	$(\mathbf{T}^7, \mathbf{T}^8)$	$(\mathbf{T}^7, \mathbf{T}^9)$	$(\mathbf{T}^7, \mathbf{T}^{11})$	$(\mathbf{T}^7, \mathbf{T}^{13})$	$(\mathbf{T}^8, \mathbf{T}^9)$	$(\mathbf{T}^8, \mathbf{T}^{10})$	$(\mathbf{T}^8, \mathbf{T}^{13})$	$(\mathbf{T}^8, \mathbf{O})$
	$(\mathbf{T}^9, \mathbf{T}^{14})$	$(\mathbf{T}^9, \mathbf{O})$	$(\mathbf{T}^{10}, \mathbf{O})$	$(\mathbf{T}^{11}, \mathbf{T}^{12})$	$(\mathbf{T}^{11}, \mathbf{T}^{14})$	$(\mathbf{T}^{12}, \mathbf{T}^{13})$	$(\mathbf{T}^{14}, \mathbf{O})$	
$\mathbf{T}^5$	$(\mathbf{T}^6, \mathbf{T}^7)$	$(\mathbf{T}^6, \mathbf{T}^8)$	$(\mathbf{T}^6, \mathbf{T}^{10})$	$(\mathbf{T}^6, \mathbf{T}^{12})$	$(\mathbf{T}^6, \mathbf{T}^{13})$	$(\mathbf{T}^7, \mathbf{T}^9)$	$(\mathbf{T}^7, \mathbf{T}^{10})$	$(\mathbf{T}^7, \mathbf{T}^{11})$
	$(\mathbf{T}^8, \mathbf{T}^{12})$	$(\mathbf{T}^8, \mathbf{O})$	$(\mathbf{T}^9, \mathbf{T}^{10})$	$(\mathbf{T}^9, \mathbf{T}^{13})$	$(\mathbf{T}^{10}, \mathbf{T}^{11})$	$(\mathbf{T}^{10}, \mathbf{T}^{12})$	$(\mathbf{T}^{10}, \mathbf{T}^{13})$	$(\mathbf{T}^{10}, \mathbf{T}^{14})$
	$(\mathbf{T}^{11}, \mathbf{T}^{12})$	$(\mathbf{T}^{11}, \mathbf{O})$	$(\mathbf{T}^{12}, \mathbf{T}^{14})$	$(\mathbf{T}^{12}, \mathbf{O})$	$(\mathbf{T}^{13}, \mathbf{T}^{14})$	$(\mathbf{T}^{14}, \mathbf{O})$		
$\mathbf{T}^6$	$(\mathbf{T}^7, \mathbf{T}^{10})$	$(\mathbf{T}^7, \mathbf{T}^{11})$	$(\mathbf{T}^7, \mathbf{T}^{12})$	$(\mathbf{T}^7, \mathbf{T}^{14})$	$(\mathbf{T}^7, \mathbf{O})$	$(\mathbf{T}^8, \mathbf{T}^9)$	$(\mathbf{T}^8, \mathbf{T}^{12})$	$(\mathbf{T}^8, \mathbf{T}^{13})$
	$(\mathbf{T}^9, \mathbf{T}^{11})$	$(\mathbf{T}^9, \mathbf{T}^{12})$	$(\mathbf{T}^9, \mathbf{T}^{14})$	$(\mathbf{T}^{10}, \mathbf{T}^{13})$	$(\mathbf{T}^{10}, \mathbf{O})$	$(\mathbf{T}^{11}, \mathbf{T}^{14})$	$(\mathbf{T}^{11}, \mathbf{O})$	$(\mathbf{T}^{12}, \mathbf{T}^{14})$
	$(\mathbf{T}^{13}, \mathbf{O})$							
$\mathbf{T}^7$	$(\mathbf{T}^8, \mathbf{T}^9)$	$(\mathbf{T}^8, \mathbf{T}^{10})$	$(\mathbf{T}^8, \mathbf{T}^{13})$	$(\mathbf{T}^8, \mathbf{T}^{14})$	$(\mathbf{T}^9, \mathbf{T}^{10})$	$(\mathbf{T}^9, \mathbf{O})$	$(\mathbf{T}^{10}, \mathbf{T}^{14})$	$(\mathbf{T}^{10}, \mathbf{O})$
	$(\mathbf{T}^{11}, \mathbf{T}^{13})$	$(\mathbf{T}^{11}, \mathbf{T}^{14})$	$(\mathbf{T}^{12}, \mathbf{T}^{13})$	$(\mathbf{T}^{12}, \mathbf{T}^{14})$	$(\mathbf{T}^{12}, \mathbf{O})$	$(\mathbf{T}^{13}, \mathbf{T}^{14})$		
$\mathbf{T}^8$	$(\mathbf{T}^9, \mathbf{T}^{11})$	$(\mathbf{T}^{10}, \mathbf{T}^{14})$	$(\mathbf{T}^{10}, \mathbf{O})$	$(\mathbf{T}^{11}, \mathbf{T}^{12})$	$(\mathbf{T}^{11}, \mathbf{T}^{14})$	$(\mathbf{T}^{13}, \mathbf{O})$		
$\mathbf{T}^9$	$(\mathbf{T}^{10}, \mathbf{T}^{12})$	$(\mathbf{T}^{10}, \mathbf{T}^{13})$	$(\mathbf{T}^{10}, \mathbf{O})$	$(\mathbf{T}^{11}, \mathbf{T}^{13})$	$(\mathbf{T}^{11}, \mathbf{T}^{14})$	$(\mathbf{T}^{12}, \mathbf{T}^{13})$	$(\mathbf{T}^{12}, \mathbf{T}^{14})$	$(\mathbf{T}^{13}, \mathbf{T}^{14})$
	$(\mathbf{T}^{13}, \mathbf{O})$	$(\mathbf{T}^{14}, \mathbf{O})$						
$\mathbf{T}^{10}$	$(\mathbf{T}^{11}, \mathbf{T}^{12})$	$(\mathbf{T}^{11}, \mathbf{T}^{13})$	$(\mathbf{T}^{12}, \mathbf{T}^{14})$	$(\mathbf{T}^{13}, \mathbf{O})$				
$\mathbf{T}^{11}$	$(\mathbf{T}^{12}, \mathbf{O})$	$(\mathbf{T}^{13}, \mathbf{T}^{14})$						
$\mathbf{T}^{12}$	$(\mathbf{T}^{13}, \mathbf{T}^{14})$	$(\mathbf{T}^{14}, \mathbf{O})$						
$\mathbf{T}^{13}$								
$\mathbf{T}^{14}$								
$\mathbf{O}$								

First, we choose $\Omega = \{\mathbf{T}^{13}, \mathbf{T}^{14}\}$. Next, we determine $\mathbf{T}^{\alpha_1} = \mathbf{T}^8$ as having the largest exponent because the row corresponding to the element $\mathbf{T}^{\alpha_1} = \mathbf{T}^8$ includes both elements $\mathbf{T}^{13}$ and $\mathbf{T}^{14}$, and hence we have $\Omega = \{\mathbf{T}^{14}, \mathbf{T}^{13}, \mathbf{T}^8\}$. Further we find the row corresponding to the element $\mathbf{T}^{\alpha_1} = \mathbf{I}$, which includes all elements in Ω. Then we have $\Omega = \{\mathbf{T}^{14}, \mathbf{T}^{13}, \mathbf{T}^8, \mathbf{I}\}$ as a block.

In the next stage, we have $\Omega = \{\mathbf{T}^{12}, \mathbf{O}\}$. We determine $\mathbf{T}^{\alpha_1} = \mathbf{T}^9$ as with the largest exponent, and then $\Omega = \{\mathbf{O}, \mathbf{T}^{12}, \mathbf{T}^9\}$. However, we cannot find the row that includes all these three elements in $\mathbf{T}^{\alpha_2}$ column, so we remove the element $\mathbf{T}^9$. Likewise we determine an element $\mathbf{T}^{\alpha_1} = \mathbf{T}^6$ ($\mathbf{T}^8$ is already marked and we cannot choose $\mathbf{T}^8$), and then $\Omega = \{\mathbf{O}, \mathbf{T}^{12}, \mathbf{T}^6\}$. Further we determine $\mathbf{T}^{\alpha_1} = \mathbf{T}^4$, and then $\Omega = \{\mathbf{O}, \mathbf{T}^{12}, \mathbf{T}^6, \mathbf{T}^4\}$ as a block.

We proceed to choose $\Omega = \{\mathbf{T}^{11}, \mathbf{T}^9\}$ and finally determine $\Omega = \{\mathbf{T}^{11}, \mathbf{T}^9, \mathbf{T}^5, \mathbf{T}\}$ as a block. In the same way we choose $\Omega = \{\mathbf{T}^{10}, \mathbf{T}^7\}$, and determine $\Omega = \{\mathbf{T}^{10}, \mathbf{T}^7, \mathbf{T}^3, \mathbf{T}^2\}$. From the Ω's obtained above, we have the following $\mathbf{H}$ matrix of (128, 116) S4EC-$S_{2\times4/16}$ED code:

$$\mathbf{H} = \left[\begin{array}{cccc:cccc:cccc:cccc} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \mathbf{I} & \mathbf{T}^8 & \mathbf{T}^{13} & \mathbf{T}^{14} & \mathbf{T}^4 & \mathbf{T}^6 & \mathbf{T}^{12} & \mathbf{O} & \mathbf{T} & \mathbf{T}^5 & \mathbf{T}^9 & \mathbf{T}^{11} & \mathbf{T}^2 & \mathbf{T}^3 & \mathbf{T}^7 & \mathbf{T}^{10} \\ \mathbf{I} & \mathbf{T} & \mathbf{T}^{11} & \mathbf{T}^{13} & \mathbf{T}^8 & \mathbf{T}^{12} & \mathbf{T}^9 & \mathbf{O} & \mathbf{T}^2 & \mathbf{T}^{10} & \mathbf{T}^3 & \mathbf{T}^7 & \mathbf{T}^4 & \mathbf{T}^6 & \mathbf{T}^{14} & \mathbf{T}^5 \\ \hdashline \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \mathbf{I} & \mathbf{T}^8 & \mathbf{T}^{13} & \mathbf{T}^{14} & \mathbf{T}^4 & \mathbf{T}^6 & \mathbf{T}^{12} & \mathbf{O} & \mathbf{T} & \mathbf{T}^5 & \mathbf{T}^9 & \mathbf{T}^{11} & \mathbf{T}^2 & \mathbf{T}^3 & \mathbf{T}^7 & \mathbf{T}^{10} \\ \mathbf{O} & \mathbf{T}^4 & \mathbf{T}^{12} & \mathbf{T}^6 & \mathbf{T}^2 & \mathbf{T}^{11} & \mathbf{T}^7 & \mathbf{I} & \mathbf{T}^8 & \mathbf{T}^5 & \mathbf{T}^{14} & \mathbf{T}^9 & \mathbf{T} & \mathbf{T}^{13} & \mathbf{T}^3 & \mathbf{T}^{10} \end{array}\right]. \tag{5.39}$$

Converted Codes As was indicated in the Chen SbEC-DbED codes mentioned in Subsection 5.2.3, the converted method adopted in the parity-check matrix of the byte error control codes can also be applied to the code. For example, for $b = 5$ we can determine the following sets of block elements, 5×5 companion matrices defined by the primitive polynomial $\mathbf{g}(x) = x^5 + x^2 + 1$:

$$\begin{array}{lll} \{\mathbf{O}, \mathbf{T}^{28}, \mathbf{T}^{26}, \mathbf{T}^{18}\} & \{\mathbf{T}^{30}, \mathbf{T}^{27}, \mathbf{T}^{24}, \mathbf{T}^{11}\} & \{\mathbf{T}^{25}, \mathbf{T}^{23}, \mathbf{T}^{22}, \mathbf{T}^{20}\} \\ \{\mathbf{T}^{29}, \mathbf{T}^{21}, \mathbf{T}^{15}, \mathbf{T}^{12}\} & \{\mathbf{T}^{19}, \mathbf{T}^{17}, \mathbf{T}^{6}, \mathbf{T}^{4}\} & \{\mathbf{T}^{16}, \mathbf{T}^{14}, \mathbf{T}^{13}, \mathbf{T}^{9}\} \\ \{\mathbf{T}^{10}, \mathbf{T}^{8}, \mathbf{T}^{7}, \mathbf{T}^{3}\}. & & \end{array} \tag{5.40}$$

By deleting one corresponding column of all 5×5 matrix elements, for example, by deleting the last column of all 5×5 matrices, and hence by deleting one all-zero row in $\mathbf{H}$, we can design (224, 210) S4EC-$S_{2\times4/16}$ED code.

Lengthened Code The following theorem presents the generalized organization of the lengthened code that can apply to the code obtained by Theorem 5.18. This is very similar to the former code defined by Theorem 5.17.

Theorem 5.19 *The null space of the following $\boldsymbol{H}$ matrix is a lengthened SbEC-$S_{p\times b/B}$ED code:*

$$\boldsymbol{H} = \left[\begin{array}{c:c:c:c:c} \begin{matrix} \boldsymbol{I}\,\boldsymbol{I} \cdots \boldsymbol{I} \\ \boldsymbol{H}_0 \end{matrix} & \begin{matrix} \boldsymbol{T}\,\boldsymbol{T} \cdots \boldsymbol{T} \\ \boldsymbol{H}_0 \end{matrix} & \cdots & \begin{matrix} \boldsymbol{T}^{q-2}\,\boldsymbol{T}^{q-2} \cdots \boldsymbol{T}^{q-2} \\ \boldsymbol{H}_0 \end{matrix} & \begin{matrix} \boldsymbol{O}\,\boldsymbol{O} \cdots \boldsymbol{O} \\ \boldsymbol{H}_0 \end{matrix} \end{array}\right],$$

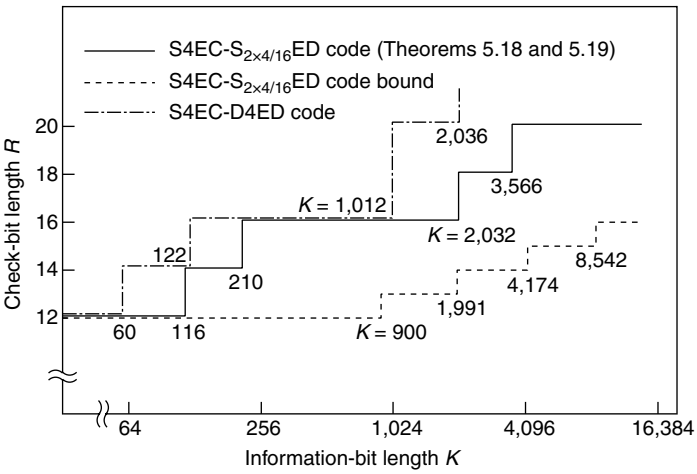


Figure 5.16 *Comparison of check-bit lengths and information-bit lengths of the S4EC-$S_{2\times4/16}$ ED codes.* Source: *[JOHJ97]. © IEICE Japan.*

where $\boldsymbol{H}_0$ is the parity-check matrix of an SbEC-$S_{p\times b/B}$ED code determined, for example, by Theorem 5.18, $q = 2^b$, $\boldsymbol{T}$ is a $b \times b$ companion matrix, and $\boldsymbol{I}$ and $\boldsymbol{O}$ are $b \times b$ identity matrix and $b \times b$ zero matrix, respectively.

This can be easily proved and hence omitted.

5.3.3 Evaluation

Figure 5.16 shows the relationship of the check-bit length to the information-bit length for the S4EC-$S_{2\times4/16}$ED code along with its code bound compared with the case of the S4EC-D4ED code. From the computer simulation we know that the (76, 64) S4EC-$S_{2\times4/16}$ED code shown in Eq. (5.37) has the following error detection capabilities:

- Random 2-bit errors: 98.11%
- Random 2-byte errors: 97.54%
- Random 3-bit errors: 91.99%
- Any burst errors in a block: 93.11%
- One-byte errors in a block and one-bit errors in another block occurred simultaneously: 97.08%

Further the (128, 116) S4EC-$S_{2\times4/16}$ED code shown in Eq. (5.39) has the following error detection capabilities:

- Random 2-bit errors: 92.91%
- Random 2-byte errors: 90.97%
- Random 3-bit errors: 84.03%
- Any burst errors in a block: 88.35%
- One-byte errors in a block and one-bit errors in another block occurred simultaneously: 90.00%

EXERCISES

5.1 For the following (18, 12) S2EC code answer the questions shown below,

$$\mathbf{H} = \begin{bmatrix} \mathbf{T}^2 & \mathbf{0} & \mathbf{T} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{T} & \mathbf{T}^2 & \mathbf{0} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{I} & \mathbf{T}^2 & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}.$$

T: companion matrix defined by the polynomial of $x^2 + x + 1$.
Codeword: $[D \mid C] = [D_0\ D_1\ D_2\ D_3\ D_4\ D_5 \mid C_0\ C_1\ C_2]$,

$$D_i = (d_{i,0}\ d_{i,1}) \quad C_j = (c_{j,0}\ c_{j,1}).$$

(a) Encode the data $[D_0\ D_1\ D_2\ D_3\ D_4\ D_5] = [1\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1]$.
(b) The received data $[D' \mid C'] = [1\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \mid 1\ 0\ 1\ 1\ 1\ 1]$ has a single-byte error. Find the correct data.
(c) Let $[D \mid C]$ be a codeword. Find the codeword for $\overline{D}$, complement of D.

5.2 Using the primitive polynomial of $x^2 + x + 1$ over GF(2), do the following:

(a) Design the **H** matrix of a single-symbol error correcting code over GF(2^2) with $k = 18$ (i.e., $K = 36$ bits).
(b) Choose the column vectors in the **H** matrix obtained in **(a)** and find the **H** matrix of the minimum-weight single-symbol error correcting code over GF(2^2) with $n = 12$ and $k = 9$.
(c) Design the parallel decoding circuit of the (12, 9) code obtained in **(b)** by using only AND, OR, NOT, and exclusive-OR (XOR) gates.

5.3 In the **H** matrix below, assume that the obtained nonzero syndrome does not point to any one byte error, and that two error pointers have already been given in the byte positions 1 and 5, shown by ↓ in **H**.

$$\mathbf{H} = \begin{array}{c} \quad\ \downarrow \qquad\qquad\ \downarrow \\ \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{0} \\ \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^3 & \mathbf{T}^4 & \mathbf{T}^5 & \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{array}{l} \rightarrow S_0 \\ \rightarrow S_1 \end{array} \end{array}$$

T: companion matrix defined by the primitive polynomial $x^4 + x + 1$.
Show that these double-byte errors can be corrected, and express the error patterns E_1 and E_5 by the syndromes S_0 and S_1.

5.4 Prove properties 1 through 3 of the companion matrix **T**.

5.5 Show that the rotational Hamming-type SbEC code has the following maximum code length in bits:

$$N = \frac{b}{2^b - 1} \sum_{d|r} \mu(d)\{(2^b)^{r/d} - 1\}\mathrm{GCD}(d, 2^b - 1),$$

where $\sum_{d|r}$ means the summation of d that divides r, $\mu(d)$ is a Möbius function, and $\mathrm{GCD}(x, y)$ means the greatest common divisor of x and y [IMAI79].

5.6 Prove that the Burton code has the properties 1 to 3 shown in Eqs. (5.7) to (5.9), respectively.

5.7 Design the Fujiwara S4EC code with $K = 56$ bits. Then, using this $\mathbf{H}$ matrix, obtain the rotational Fujiwara S4EC code with $K = 56$ bits. In this case let the companion matrix $\mathbf{T}$ be defined by the primitive polynomial $x^4 + x + 1$.

5.8 Find the generating submatrix $\mathbf{H}_0$ of the rotational Fujiwara S3EC code with $K = 180$ bits.

5.9 Design the Hong-Patel S2EC code with $K = 32$ bits.

5.10 Prove Theorem 5.7.

5.11 Design the Kaneda-Fujiwara S4EC-D4ED code with $K = 128$ bits.

5.12 Design the modularized decoding circuit of the code shown in Figure 5.9.

5.13 Assume that the code expressed by matrix $\mathbf{H}'$, which includes submatrix $\mathbf{H}_0$ having an all-$\mathbf{I}$ (identity element $\mathbf{I} \in \mathrm{GF}(2^b)$) row, is a $((k + r)b,\ kb)$ SbEC-DbED code. Prove that the following code expressed by $\mathbf{H}$ is a 2-modularized $((2k + r + 1)b,\ 2kb)$ SbEC-DbED code:

$$\mathbf{H}' = \left[\begin{array}{c|c} \begin{bmatrix} (\mathbf{I} & \mathbf{I} & \cdots & \mathbf{I}) \\ & & \mathbf{H}_0 & \end{bmatrix} & \begin{matrix} \mathbf{I} & & \\ & \mathbf{I} & \\ & & \ddots \\ & & \mathbf{I} \end{matrix} \end{array}\right]$$

$$\mathbf{H} = \left[\begin{array}{c|c|c} \begin{bmatrix} & \mathbf{H}_0 & \\ & & \end{bmatrix} & \begin{matrix} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \multicolumn{4}{c}{{}^0\mathbf{H}} \end{matrix} & \begin{matrix} \mathbf{I} & & \\ & \mathbf{I} & \\ & & \ddots \\ & & \mathbf{I} \end{matrix} \\ \hline \mathbf{0} \quad \mathbf{0} \quad \cdots \quad \mathbf{0} & & \end{array}\right]$$

Here ${}^0\mathbf{H}$ is a matrix whose order of row vectors in $\mathbf{H}_0$ is turned upside down. Use this code design method to design the 2-modularized S2EC-D2ED code with $N = 76$ bits and $K = 64$ bits.

5.14 Use Theorems 5.11 through 5.13 to design the Chen S3EC-D3ED code with $K = 183$ bits.

5.15 Use the conversion procedure to design the (33, 22) S3EC-D3ED code from the following $\mathbf{H}$ matrix with a normalized form of the (44, 32) S4EC-D4ED code:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^3 & \mathbf{T}^4 & \mathbf{T}^5 & \mathbf{T}^6 & \mathbf{T}^7 \\ \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^4 & \mathbf{T}^6 & \mathbf{T}^8 & \mathbf{T}^{10} & \mathbf{T}^{12} & \mathbf{T}^{14} \end{bmatrix},$$

where

$\mathbf{I}$: identity element $\in$ GF(2^4),
$\mathbf{0}$: zero element $\in$ GF(2^4),
$\mathbf{T}$: companion matrix defined by the primitive polynomial $x^4 + x + 1$.

5.16 Prove Theorems 5.15 and 5.16.

5.17 Verify that Table 5.11 satisfies the relation (5.38).

5.18 Use the conversion procedure shown in Subsection 5.2.3 to design the $(224,\ 210)$ S4EC-$S_{2\times4/16}$ED code.

5.19 Prove Theorem 5.19.

5.20 Using Table 5.11, determine five sets of Ω for $p = 2$ bytes and $B/b = 3$ ($b = 4$ bits, $B = 12$ bits). Design the $(120,\ 108)$ S4EC-$S_{2\times4/12}$ED code.

(Answer : $(\mathbf{T}^{14},\ \mathbf{T}^{13},\ \mathbf{T}^{8})$, $(\mathbf{O},\ \mathbf{T}^{12},\ \mathbf{T}^{9})$, $(\mathbf{T}^{11},\ \mathbf{T}^{7},\ \mathbf{T}^{2})$, $(\mathbf{T}^{6},\ \mathbf{T}^{5},\ \mathbf{T}^{4})$, $(\mathbf{T}^{10},\ \mathbf{T}^{3},\ \mathbf{T})$.)

5.21 Determine the table of elements satisfying the relation (5.38) for $p = 2$ bytes and $b = 3$ bits. Using this table, design the S3EC-$S_{2\times3/B}$ED code with $B = 9$ bits and 12 bits. Here the 3×3 companion matrix $\mathbf{T}$ is defined by the primitive polynomial $\mathbf{g}(x) = x^3 + x + 1$.

(Answer:

$\mathbf{T}^{\alpha_1}$	$\mathbf{T}^{\alpha_2}$			
$\mathbf{I}$	$\mathbf{T}^3$	$\mathbf{T}^5$	$\mathbf{T}^6$	$\mathbf{O}$
$\mathbf{T}$	$\mathbf{T}^2$	$\mathbf{T}^3$	$\mathbf{T}^4$	$\mathbf{O}$
$\mathbf{T}^2$	$\mathbf{T}^4$	$\mathbf{T}^6$	$\mathbf{O}$	
$\mathbf{T}^3$	$\mathbf{T}^5$	$\mathbf{T}^6$		
$\mathbf{T}^4$	$\mathbf{T}^5$	$\mathbf{O}$		
$\mathbf{T}^5$	$\mathbf{T}^6$			
$\mathbf{T}^6$				
$\mathbf{O}$				

for $B/b = 3$, $\Omega = (\mathbf{T}^6,\ \mathbf{T}^5,\ \mathbf{T}^3)$ and $(\mathbf{O},\ \mathbf{T}^4,\ \mathbf{T}^2)$, and for $B/b = 4$, $\Omega = (\mathbf{T}^6,\ \mathbf{T}^5,\ \mathbf{T}^3,\ \mathbf{I})$ and $(\mathbf{O},\ \mathbf{T}^4,\ \mathbf{T}^2,\ \mathbf{T})$.)

5.22 Use Table 5.12 to design the $(128,\ 112)$ S4EC-$S_{3\times4/16}$ED code.

(Hint: The sets of elements are obtained by the algorithm $\{\mathbf{O},\ \mathbf{T}^{14},\ \mathbf{T}^{12},\ \mathbf{T}^5\}$, $\{\mathbf{T}^{13},\ \mathbf{T}^{11},\ \mathbf{T}^{10},\ \mathbf{T}^2\}$, $\{\mathbf{T}^9,\ \mathbf{T}^8,\ \mathbf{T}^6,\ \mathbf{T}^4\}$, $\{\mathbf{T}^7,\ \mathbf{T}^3,\ \mathbf{T},\ \mathbf{I}\}$, where $\mathbf{T}$ is defined by $\mathbf{g}(x) = x^4 + x + 1$.)

5.23 As for the single-byte error correcting and adjacent double-byte error detecting (SbEC-ADbED) codes, do the following:

(a) Prove that the relation below holds for $(N,\ N-R)$ SbEC-ADbED code:

$$N \leq \frac{b(2^R - 2^{2b})}{2^b - 1} + 2b.$$

(b) Show that null space of the following parity-check matrix is an SbEC-ADbED code:

$$\mathbf{H}_n = \left[\begin{array}{c|c|c|c|c|c} \mathbf{I} \cdots \mathbf{I} & \mathbf{T} \cdots \mathbf{T} & & \mathbf{T}^i \cdots \mathbf{T}^i & & \mathbf{T}^{q-2} \cdots \mathbf{T}^{q-2} \quad \mathbf{O} \cdots \mathbf{O} \\ \mathbf{H}_{n-1} & \mathbf{H}_{n-1} & \cdots & \mathbf{H}_{n-1} & \cdots & \mathbf{H}_{n-1} \quad \mathbf{H}_{n-1} \end{array}\right],$$

where $\mathbf{H}_0$ is a nonsingular binary matrix with rank $2b$, $0 \leq i \leq q-2$, $q = 2^b$, $\mathbf{I}$ is a $b \times b$ identity matrix, $\mathbf{O}$ is a $b \times b$ zero matrix, and $\mathbf{T}$ is a $b \times b$ companion matrix defined by the binary primitive polynomial with degree b. Also show that the maximum code length in bits is given by $N = 2b \cdot 2^{b(R/b-2)}$, where $n = R/b - 2$.

(c) Take the challenge to design a more efficient code than the code shown in (**b**).

REFERENCES

[ARLA84] J. Arlet and W. C. Carter, "Implementation and Evaluation of a (b, k)-Adjacent Error Correcting / Detecting Scheme for Supercomputer Systems," *IBM J. Res. Dev.*, 28 (March 1984): 159–168.

[BHAT78] A. K. Bhatt and L. L. Kinney, "A High Speed Parallel Encoder / Decoder for b-Adjacent Error-Checking Codes," *Proc. 3rd USA–Japan Computer Conf.* (1978): 203–207.

[BISH96] J. W. Bishop, M. J. Campion, T. L. Jeremiah, S. J. Mercier, et al., "PowerPC AS A10 64-Bit RISC Microprocessor," *IBM J. Res. Dev.*, 40 (July 1996): 495–505.

[BOSS70] D. C. Bossen, "b-Adjacent Error Correction," *IBM J. Res. Dev.*, 14 (July 1970): 402–408.

[BURT71] H. O. Burton, "Some Asymptotically Optimal Burst-Correcting Codes and Their Relation to Single-Error Correcting Reed-Solomon Codes," *IEEE Trans. Info. Theory*, IT-17 (January 1971): 92–95.

[CART74] W. C. Cater, G. B. Leeman Jr., and A. B. Wadia, "Practical Length Single-Bit Error Correction / Double-Bit Error Detection Codes for small Values of b," *IBM Techn. Dis. Bull.*, 17 (December 1974): 2174–2176.

[CART80] W. C. Carter and A. B. Wadia, "Design and Analysis of Codes and Their Self-Checking Circuit Implementations for Correction and Detection of Multiple-b-Adjacent Errors," *Dig., 10th IEEE Int. Symp. Fault-Tolerant Computing* (October 1980): 35–40.

[CHEN83] C. L. Chen, "Error-Correcting Codes with Byte Error-Detection Capability," *IEEE Trans. Comput.*, C-32 (July 1983): 615–621.

[CHEN86a] C. L. Chen, "Byte-Oriented Error-Correcting Codes for Semiconductor Memory Systems," *IEEE Trans. Computers*, C-35 (July 1986): 646–648.

[CHEN86b] C. L. Chen, "Error-Correcting Codes for Byte-Organized Memory Systems," *IEEE Trans. Info. Theory*, IT-32 (March 1986): 181–185.

[CHEN91] C. L. Chen and L.E. Grosbach, ''Fault-Tolerant Memory Design in the IBM Application System/400™,'' *Dig. 21th IEEE Int. Symp. Fault-Tolerant Computing* (June 1991): 393–400.

[CHEN92] C. L. Chen, ''Symbol Error-Correcting Codes for Computer Memory Systems,'' *IEEE Trans. Comput.*, 41 (February 1992): 252–256.

[DENG87] R. H. Deng and D. J. Costello Jr., ''Decoding of DBEC-TBED Reed-Solomon Codes,'' *IEEE Trans. Comput.*, C-36 (November 1987): 1359–1363.

[FIRE59] P. Fire, ''A Class of Multiple-Error-Correcting Binary Codes for Non-Independent Errors,'' *Sylvania Report RSL-E-2*, Sylvania Electronic Defense Laboratory, Reconnaissance, Systems Division (1959).

[FUJI76] E. Fujiwara, ''Modularized *b*-Adjacent Error Correction'' (in Japanese), *Paper of Technical Group*, *IECE Japan*, EC76–19 (1976).

[FUJI77a] E. Fujiwara, ''A Modularized *b*-Adjacent Error Correction Memory Unit,'' *Trans. IECE Japan*, E60 (February 1977): 69–76.

[FUJI77b] E. Fujiwara and T. Kawakami, ''Modularized *b*-Adjacent Error Correction,'' *Dig., 7th IEEE Int. Symp. Fault-Tolerant Computing* (June 1977): 199.

[FUJI78] E. Fujiwara, ''Odd-Weight-Column *b*-Adjacent Error Correcting Codes,'' *Trans. IECE Japan*, *E61* (October 1978): 781–787.

[FUJI81] E. Fujiwara, ''Error Correcting Code and Its Application to Digital Systems'' (in Japanese), PhD Dissertation, Tokyo Institute of Technology (April 1981).

[FUJI82] E. Fujiwara and S. Kaneda, ''Application of Error Correcting Codes for Increasing Computer System Reliability'' (in Japanese), *J. Info. Process. Soc. (IPS) Japan*, 23 (April 1982): 292–298.

[FUJI90] E. Fujiwara, in H. Imai (ed.), *Essentials of Error-Correcting Coding Techniques*, Academic Press (1990), ch. 4.

[HAMM50] R. W. Hamming, ''Error Detecting and Error Correcting Codes,'' *Bell Syst. Techn. J.*, 26 (April 1950): 147–160.

[HART72] C. R. P. Hartman and K. K. Tzeng, ''Generalization of the BCH Bound,'' *Info. Control*, 18 (1972): 489–498.

[HONG72] S. J. Hong and A. M. Patel, ''A General Class of Maximal Codes for Computer Applications,'' *IEEE Trans. Comput.*, C-21 (December 1972): 1322–1331.

[HORI83] T. Horiguchi and Y. Sato, ''A Decoding Method for Reed-Solomon Codes over $GF(2^m)$'' (in Japanese), *Trans. IECE Japan*, J66-A (January 1983): 97–98.

[IMAI79] H. Imai and Y. Kamiyanagi, ''On Burst-Error-Correcting Codes for Application to Main Memories'' (in Japanese), *Trans. IECE Japan*, J62-D (October 1979): 633–640.

[ITOH83] H. Itoh and M. Nakamichi, ''S*b*EC-D*b*ED Codes Derived from Experiments on a Computer for Semiconductor Memory Systems'' (in Japanese), *Trans. IECE Japan*, J66-A (August 1983): 741–748.

[JOHJ97] Y. Johji and E. Fujiwara, ''A Class of Byte Error Control Codes Based on Hierarchical Error Model,'' *Technical Report of IEICE*, FTS 96-58 (February 1997).

[KANE82] S. Kaneda and E. Fujiwara, ''Single Byte Error Correcting-Double Byte Error Detecting Codes for Memory Systems,'' *IEEE Trans. Comput.*, C-31 (July 1982): 596–602. (Also in *Dig., 10th Ann. Int. Symp. Fault-Tolerant Computing* [October 1980]: 41–46.)

[NARA80] Y. Nara, Y. Sohma, and A. Hattori, ''Error-Correcting and Error-Detecting Systems,'' US Patent 4214228 (July 22, 1980).

[NUMA89] K. Numata, Y. Oowaki, Y. Itoh, et al., ''New Nibbled-Page Architecture for High-Density DRAMs,'' *IEEE J. Solid-State Circ.*, 24 (August 1989): 900–904.

[REED60] I. S. Reed and G. Solomon, ''Polynomial Codes over Certain Finite Fields,'' *SIAM J. Appl. Math.*, 8 (June 1960): 300–304.

[SUNM95] Sun Microsystems, Inc. ''UltraSPARC-I Data Buffer (UDB) DATA SHEET, Revision 0.3,'' *SPARC Technology* (May 1995).

[VARA83] M. R. Varanasi, T. R. N. Rao, and S. Pham, ''Memory Package Error Detection and Correction,'' *IEEE Trans. Comput.*, C-32 (September 1983): 872–874.

[WOLF69] J. K. Wolf, ''Adding Two Information Symbols to Certain Nonbinary BCH Codes and Some Applications,'' *Bell Syst. Techn. J.*, 48 (September 1969): 2405–2424.

CONTENTS

6

Codes for High-Speed Memories III: Bit / Byte Error Control Codes

Byte-organized semiconductor memory chips are widely used in today's digital systems. The usual errors occurred in these systems are soft errors induced by external noise, α particles, etc., which are apt to be manifested as random bit errors in byte-organized systems. Memory cell failure also results in random bit errors. Therefore designers of error control codes for byte-organized memory systems must take into account two types of errors, byte errors and bit errors. Strictly speaking, bit errors are a class of byte errors that corrupt exactly one bit within a byte, but such errors are usually called bit errors, and all others called byte errors.

Based on the background above, this chapter deals with a class of practical codes that controls such errors as mixing byte errors with bit errors. These are abbreviated and designated as follows:

1. *SEC-DED-SbED (or BED) codes*. Single b-bit byte (or burst) error detecting SEC-DED codes.
2. *SbEC-DED codes*. Single b-bit byte error correcting and double-bit error detecting codes.
3. *SbEC-DEC codes*. Single b-bit byte error correcting and double-bit error correcting codes.
4. *SbEC-(Sb+S)ED codes*. Single b-bit byte error correcting and single-byte plus single-bit error detecting codes.

Here, "A plus B error" means A error and B error occurred simultaneously. Hence the above code 4 can detect both single-byte errors and single-bit errors occurred simultaneously.

The SEC-DED-SbED codes have found many applications in computer memory systems, all with byte size $b = 4$ bits. Some of these codes have been applied to recent microprocessor chips as well as to recent server systems.

Code Design for Dependable Systems: Theory and Practical Applications, by Eiji Fujiwara

6.1 SINGLE-BYTE / BURST ERROR DETECTING SEC-DED CODES

We recall here the definitions of *byte errors* and *burst errors*. A word is divided into bytes of length b; a single-byte error is meant to be any number of errors confined to one byte. On the other hand, a burst error of length b is any number of errors confined to b adjacent positions. A burst of length b may begin on any bit position of the word and can spread over portions of two adjacent bytes. Also a code capable of detecting (or correcting) bursts of length b implies that it is capable of detecting (or correcting) all bursts with smaller than or equal to length b. For the purpose of this chapter, a byte always refers to a byte of length b, and a burst refers to a burst of length b or less. Since we do not cover multiple burst error control codes, the term burst here will mean a single burst only.

The SEC-DED codes are not capable of correcting or detecting byte errors or burst errors. On the other hand, SbEC codes and SbEC-DbED codes can correct byte errors but require higher redundancy. From this consideration, codes are required to detect single-byte errors as well as to correct single-bit errors and to detect double-bit errors. The SEC-DED-SbED class of codes is practical from the standpoint of requiring small additional redundancy to the existing SEC-DED codes, and hence has become very popular in recent commercial applications [CHEN84, TSUC86, SUNM95].

6.1.1 Burst Error Detecting SEC-DED Codes (SEC-DED-BED Codes)

Some types of codes that meet the requirements mentioned above have been proposed in [BOSS78, REDD78, FUJI80a, VARA83]. It turns out, however, that all these proposed codes are equivalent to the same type of codes, namely single-burst error detecting SEC-DED code [KANE83]. Generally, the single-burst error detecting SEC-DED codes include single-byte error detecting SEC-DED codes as a special case. Therefore, more efficient codes of single-byte error detecting SEC-DED codes, namely SEC-DED-SbED codes, will be studied in Subsection 6.1.3.

Here we consider the single-byte error detecting SEC-DED codes. Then it will be shown that these codes are the single-burst error detecting SEC-DED codes, called SEC-DED-BED codes. This type of codes is discussed in Subsection 6.1.2. The reader should be careful to note and distinguish the abbreviations BED and SbED. The BED stands for single-burst (of length b) error detection, and SbED stands for single-byte (of length b) error detection.

We begin with the case of two sets of errors, $\mathbf{E}_1$ and $\mathbf{E}_2$, where $\mathbf{E}_1$ is the error set consisting of all single-bit errors and $\mathbf{E}_2$ is the error set consisting of all byte errors, excluding single-bit errors. Hence $\mathbf{E}_1 \bigcap \mathbf{E}_2 = \emptyset$. ($\emptyset$ is the empty set.)

Theorem 6.1 *A linear code, described by the matrix $\boldsymbol{H}$, corrects all errors in $\boldsymbol{E}_1$ and detects all errors in $\boldsymbol{E}_2$, if and only if:*

1. $E \cdot \boldsymbol{H}^T \neq 0$ *for all* $E \in \{\boldsymbol{E}_1 \bigcup \boldsymbol{E}_2\}$,
2. $E_i \cdot \boldsymbol{H}^T \neq E_j \cdot \boldsymbol{H}^T$ *for all* $E_i, E_j \in \boldsymbol{E}_1$,
3. $E_i \cdot \boldsymbol{H}^T \neq E_j \cdot \boldsymbol{H}^T$ *for all* $E_i \in \boldsymbol{E}_1$, *and for all* $E_j \in \boldsymbol{E}_2$.

This theorem can be easily proved so that the conditions 1 and 2 are those for satisfying single-bit error correction, and the conditions 1 and 3 are for single-byte error detection. The codes satisfying all these conditions will be referred to as SEC-SbED codes.

Theorem 6.2 [BOSS78] *Let* $\boldsymbol{H}$ *be the* $R \times Jb$ *matrix:*

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{M}_1 & \boldsymbol{M}_2 & \dots & \boldsymbol{M}_J \\ \boldsymbol{Q} & \boldsymbol{Q} & \dots & \boldsymbol{Q} \end{bmatrix} = \begin{bmatrix} \boldsymbol{H}_1 & \boldsymbol{H}_2 & \dots & \boldsymbol{H}_j \end{bmatrix},$$

where,

$$\boldsymbol{H}_i = \begin{bmatrix} \boldsymbol{M}_i \\ \boldsymbol{Q} \end{bmatrix}, \quad 1 \le i \le J, \quad J = 2^{R-b+1} - 1,$$

$$\boldsymbol{Q} = \begin{bmatrix} 0 & \\ 0 & \\ . & \boldsymbol{I}_{b-1} \\ . & \\ 0 & \end{bmatrix}_{(b-1)\times b}.$$

$\boldsymbol{Q}$ *is a* $(b-1) \times b$ *matrix consisting of an all-0's column and the identity matrix of dimension* $b-1$, *and* $\boldsymbol{M}_i$ *is an* $(R-b+1) \times b$ *matrix whose columns are* b *copies of the binary representation of integer* i. *The code expressed as the foregoing* $\boldsymbol{H}$ *matrix is a single b-bit byte error detecting SEC code (SEC-SbED code) having code length in bits* $N = b \cdot (2^{R-b+1} - 1)$.

In this chapter the code length in bits N is a multiple of b, whereas the check-bit length R is not always a multiple of b. The reader should be careful not to confuse these with the notations of a previous section. Theorem 6.2 can be easily proved such that the code satisfies conditions 1, 2, and 3 in Theorem 6.1.

Example 6.1 [BOSS78]

For $R = 6$, and $b = 4$, we have a (28, 22) SEC-S4ED code with an **H** matrix

$$\mathbf{H} = \begin{bmatrix} 0000 & 0000 & 0000 & 1111 & 1111 & 1111 & 1111 \\ 0000 & 1111 & 1111 & 0000 & 0000 & 1111 & 1111 \\ 1111 & 0000 & 1111 & 0000 & 1111 & 0000 & 1111 \\ \hline 0100 & 0100 & 0100 & 0100 & 0100 & 0100 & 0100 \\ 0010 & 0010 & 0010 & 0010 & 0010 & 0010 & 0010 \\ 0001 & 0001 & 0001 & 0001 & 0001 & 0001 & 0001 \end{bmatrix}.$$

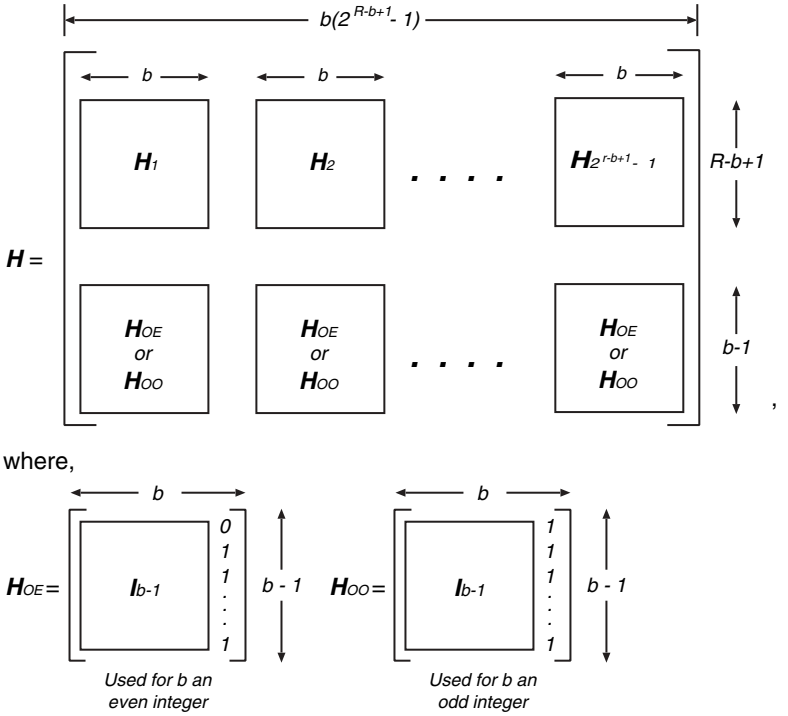
The foregoing can be made an SEC-DED-S4ED code by adding an all-1's row to the matrix. The following matrix expresses a simple example of this type of SED-DED-S4ED code:

$$\mathbf{H} = \begin{bmatrix} 1111 & 1111 & 1111 & 1111 \\ 0000 & 0000 & 1111 & 1111 \\ 0000 & 1111 & 0000 & 1111 \\ \hline 0100 & 0100 & 0100 & 0100 \\ 0010 & 0010 & 0010 & 0010 \\ 0001 & 0001 & 0001 & 0001 \end{bmatrix}.$$

In general, for the SEC-DED-SbED codes, the following condition is necessary in addition to the conditions 1, 2, and 3 of Theorem 6.1:

4. $E_i \cdot \boldsymbol{H}^T + E_j \cdot \boldsymbol{H}^T \neq E_k \cdot \boldsymbol{H}^T$ *for all* $E_i, E_j, E_k \in \boldsymbol{E}_1$, $i \neq j \neq k \neq i$.

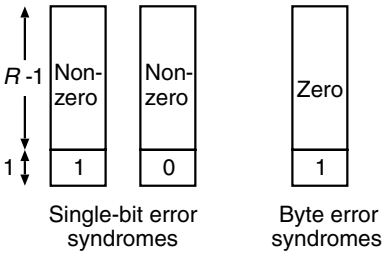
Theorem 6.3 [REDD78] *The codes given by the following* $\boldsymbol{H}$ *matrix are SEC-SbED codes when* $b = 2$, *3, or 4. When* $b \geq 5$, *the codes are SEC-DED-SbED. The code length (in bits) of the codes is* $N = b \cdot (2^{R-b+1} - 1)$.



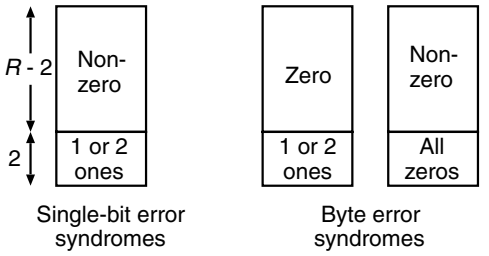
$\boldsymbol{H}_i$: $(R - b + 1) \times b$ matrix whose columns are b copies of the binary representation of integer i, $i = 1, 2, \ldots, 2^{R-b+1} - 1$,

$\boldsymbol{I}_{b-1}$: $(b - 1) \times (b - 1)$ identity matrix.

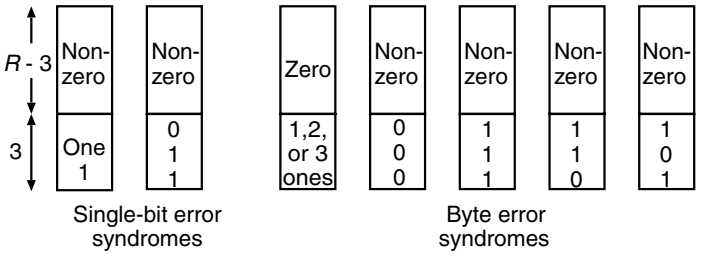
Proof The possible patterns of syndromes corresponding to the single-bit errors and single-byte errors are given in Figure 6.1. The patterns in the figure indicate whether the top $R - b + 1$ positions of the syndrome are zero or nonzero and also the number of ones or the actual bit pattern in the last $b - 1$ positions of the syndrome. In the figure note that the syndromes for the byte errors are nonzero and are different from the syndromes for the single-bit errors. Hence, for $2 \leq b \leq 4$, the codes simultaneously correct all single-bit errors and detect all single-byte errors. Similarly, for $b \geq 5$, the possible pattern of syndromes corresponding to single-bit errors, double-bit errors, and single-byte errors are given in Figure 6.2. From this figure it is apparent that the syndromes for the single-byte errors and double-bit errors are nonzero and are different from the syndromes for the single-bit errors. Hence, for $b \geq 5$,



(a) $b = 2$ bits



(b) $b = 3$ bits



(c) $b = 4$ bits

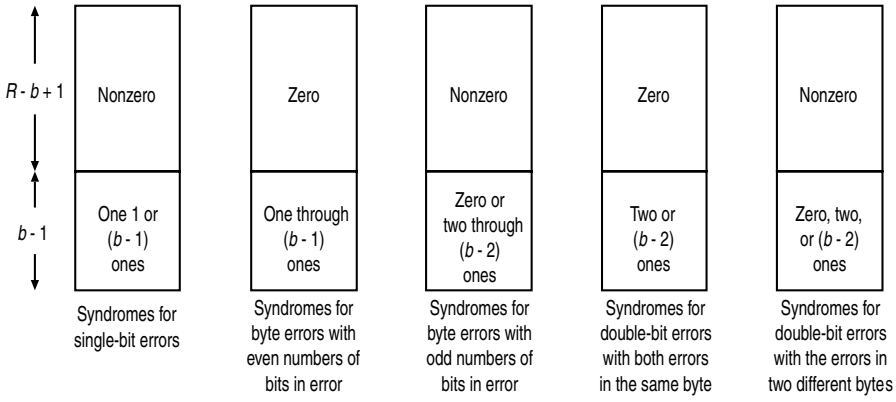
Figure 6.1 *Syndromes for* $b = 2, 3,$ *and* 4 *bits.* Source: *[REDD78]. © 1978 IEEE.*

the codes simultaneously correct all single-bit errors and detect all double-bit errors and all single-byte errors. Q.E.D.

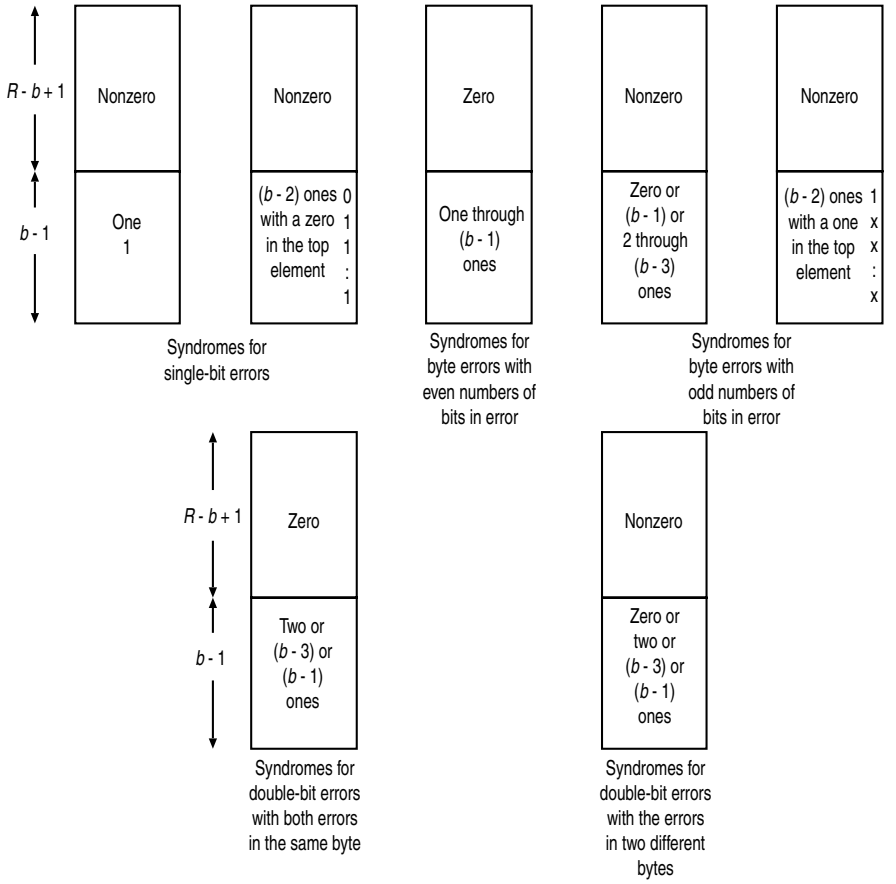
Example 6.2 [REDD78]

For $R = 6$ and $b = 3$, we have a (35,39) SEC-S3ED code with the following **H** matrix:

$$\mathbf{H} = \left[\begin{array}{cccccccccccccccc}
000 & 000 & 000 & 000 & 000 & 000 & 000 & 111 & 111 & 111 & 111 & 111 & 111 & 111 & 111 \\
000 & 000 & 000 & 111 & 111 & 111 & 111 & 000 & 000 & 000 & 000 & 111 & 111 & 111 & 111 \\
000 & 111 & 111 & 000 & 000 & 111 & 111 & 000 & 000 & 111 & 111 & 000 & 000 & 111 & 111 \\
111 & 000 & 111 & 000 & 111 & 000 & 111 & 000 & 111 & 000 & 111 & 000 & 111 & 000 & 111 \\
\hline
101 & 101 & 101 & 101 & 101 & 101 & 101 & 101 & 101 & 101 & 101 & 101 & 101 & 101 & 101 \\
011 & 011 & 011 & 011 & 011 & 011 & 011 & 011 & 011 & 011 & 011 & 011 & 011 & 011 & 011
\end{array}\right].$$



(a) Syndrome patterns for *b* an odd integer



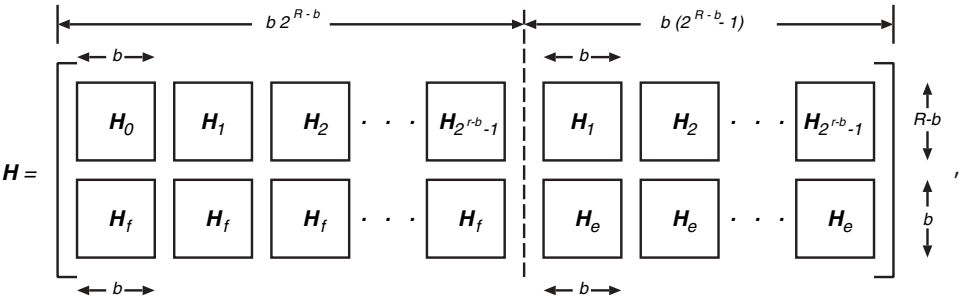
(b) Syndrome patterns for *b* an even integer

Figure 6.2 *Syndrome for $b \geq 5$ bits.* Source: *[REDD78]. © 1978 IEEE.*

If one desires to detect double-bit errors for $2 \leq b \leq 4$, an extra overall parity bit is added. From the **H** matrix above, we can also have the following (24,18) SEC-DED-S3ED code:

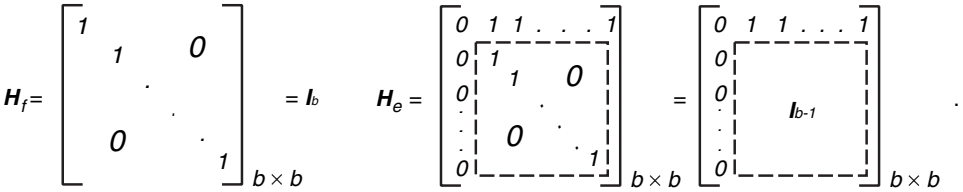
$$\mathbf{H} = \left[\begin{array}{cccccccc} 111 & 111 & 111 & 111 & 111 & 111 & 111 & 111 \\ 000 & 000 & 000 & 000 & 111 & 111 & 111 & 111 \\ 000 & 000 & 111 & 111 & 000 & 000 & 111 & 111 \\ 000 & 111 & 000 & 111 & 000 & 111 & 000 & 111 \\ \hline 101 & 101 & 101 & 101 & 101 & 101 & 101 & 101 \\ 011 & 011 & 011 & 011 & 011 & 011 & 011 & 011 \end{array}\right].$$

Theorem 6.4 [FUJI80a] *The codes given by the following* ***H*** *matrix are SEC-SbED codes. The code length in bits is* $N = b \cdot (2^{R-b+1} - 1)$.



where

$\boldsymbol{H_i}$: *$(R-b) \times b$ matrix whose columns are b copies of the binary representation of integer i, i=0, 1,. . ., $2^{R-b} - 1$*



Proof Let the syndrome pattern from the **H** matrix in this theorem be

$$S = \left[\begin{array}{c} s_0 \\ \vdots \\ s_{R-b-1} \\ \hline s_{R-b} \\ \vdots \\ s_{R-1} \end{array}\right] = \left[\begin{array}{c} S_I \\ \hline S_{II} \end{array}\right]$$

TABLE 6.1 Weight of Syndromes for Errors in E_1 and E_2

Error / Error location	$E_i \in E_1$: Errors in information-bit part	$E_i \in E_1$: Errors in check-bit part	$E_i \in \mathbf{E}_2$	
Errors in the former data part having $b \cdot 2^{R-b}$-bit length	$W_I \neq 0$	$W_I = 0$	$W(E_j) =$ even(≥ 2)	$W_I \neq 0$, $W_{II} = W(E_j) =$ even(≥ 2)
	$W_{II} = 1$	$W_{II} = 1$	$W(E_j) =$ odd(≥ 3)	$W_I \neq 0$, $W_{II} = W(E_j) =$ odd(≥ 3)
Errors in the latter data part having $b \cdot (2^{R-b} - 1)$-bit length	$W_I \neq 0$	$W_I = 1$	$W(E_j) =$ even(≥ 2)	$W_I = 0$, $W_{II} = W(E_j) =$ even(≥ 2)
	$W_{II} = \begin{cases} 0 \\ 2^{\#} \end{cases}$	$W_{II} = 0$	$W(E_j) =$ odd(≥ 3)	$W_I \neq 0$, $W_{II} = \begin{cases} W(E_j)^{\#} - 1 \\ W(E_j) + 1 \end{cases} =$ even(≥ 2)

Source: [FUJI801].© 1980 IECE Japan.

#: For $W(E_j) = 3$ in column 5, we have $s_{R-b} = 1$ for $E_i \in \mathbf{E}_1$, and $s_{R-b} = 0$ for $E_j \in \mathbf{E}_2$.

The weight of each syndrome part is defined as follows:

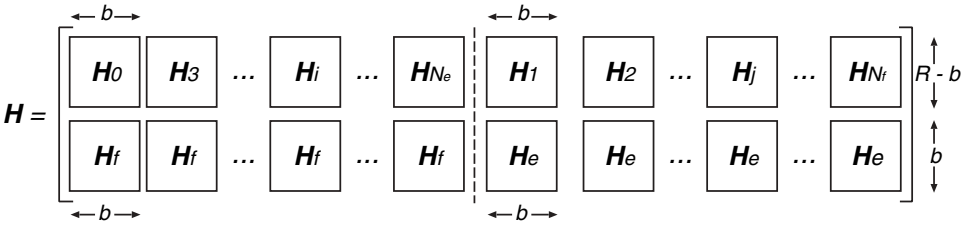
$W_I = W(S_I) =$ weight of S_I,
$W_{II} = W(S_{II}) =$ weight of S_{II}.

Table 6.1 shows the weight of the syndromes for the error sets $\mathbf{E}_1$ and $\mathbf{E}_2$. Note that the syndromes corresponding to byte errors are nonzero and are different from those corresponding to single-bit errors. Q.E.D.

Example 6.3 [FUJI80a]

The **H** matrix shown in Figure 6.3 expresses the (60, 53) SEC-S4ED code.

Theorem 6.5 [FUJI80a] *The codes given by the following* ***H*** *matrix are SEC-DED-SbED codes. The code length in bits is* $N = b \cdot 2^{R-b}$.



where

$\boldsymbol{H}_0, \boldsymbol{H}_1, \boldsymbol{H}_2, \ldots, \boldsymbol{H}_f, \boldsymbol{H}_e$: *same as matrices in Theorem 6.4,*

i: integer whose binary representation has even weight,

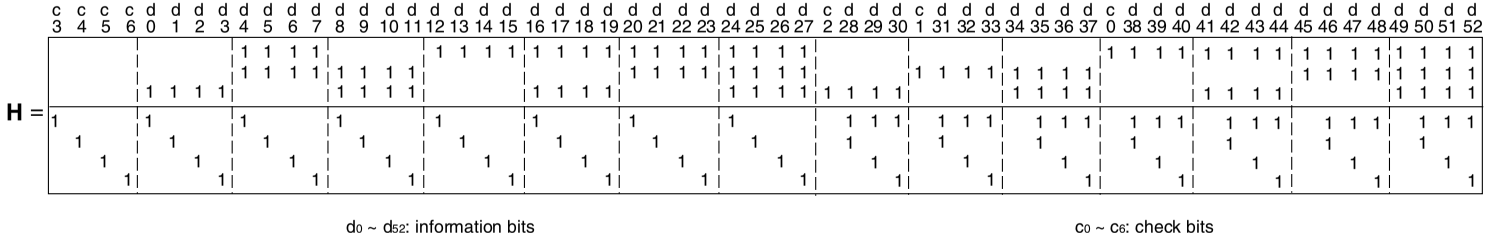


Figure 6.3 *(60, 53) SEC-S4ED code.* Source: *[FUJI80a]. © 1980 IECE Japan.*

N_e: maximal integer i no greater than $(2^{R-b} - 1)$,
j: integer whose binary representation has odd weight,
N_f: maximal integer j no greater than $(2^{R-b} - 1)$.

Proof The **H** matrix shown in Theorem 6.5 is clearly a part of the **H** matrix of the SEC-S*b*ED codes shown in Theorem 6.4. Also, since every column vector in this matrix is odd weight, it has the DED property. Therefore it is an SEC-DED-S*b*ED code. Q.E.D.

Example 6.4 [FUJI80a]

The **H** matrix of the (32,25) SEC-DED-S4ED code with $b = 4$, and $r = 7$, is designed as follows:

H =

c 3	c 4	c 5	c 6	c 0	c 1	c 2	c 3	d 4	d 5	d 6	d 7	d 8	d 9	d 10	d 11	c d 2	d 12	d 13	d 14	c 1	d 15	d 16	d 17	c 0	d 18	d 19	d 10	d 21	d 22	d 23	d 24
								1	1	1	1	1	1	1	1									1	1	1	1	1	1	1	1
				1	1	1	1					1	1	1	1					1	1	1	1					1	1	1	1
				1	1	1	1	1	1	1	1					1	1	1	1									1	1	1	1
1				1				1				1					1	1	1		1	1	1		1	1	1		1	1	1
	1				1				1				1				1				1				1				1		
		1				1				1				1				1				1				1				1	
			1				1				1				1				1				1				1				1

$c_0 \sim c_6$: check bits
$d_0 \sim d_{24}$: information bits

(6.1)

The **H** matrices of the codes defined in Theorems 6.4 and 6.5 have a nice characteristic that the check-bit position can be directly determined without any row operations.

Theorem 6.6 [VARA83] *Let $\mathbf{p}(x)$ be a polynomial of degree $l < b$ and of exponent e such that $e > b$ and $GCD(e, b) = 1$, where $GCD(e, b)$ expresses the greatest common divisor of e and b. Then the $(N, N - b - l)$ cyclic code generated by $\mathbf{g}(x) = (x^b - 1)\mathbf{p}(x)$ with $N = e \cdot b$ bits is an SEC-DED-SbED code.*

Proof Let α be a primitive root of $\mathbf{p}(x)$, $\mathbf{I}_b$ be the identity matrix of size b, and $E(x)$ be the error pattern. Then the **H** matrix can be constructed as

$$\mathbf{H} = \left[\begin{array}{cccccccc} & \mathbf{I}_b & \mathbf{I}_b & \cdots & \mathbf{I}_b & \cdots & \mathbf{I}_b & \mathbf{I}_b \\ \hline 1\alpha\alpha^2 \cdots \alpha^{e-1} & & 1\alpha\alpha^2 \cdots \alpha^{e-1} & & \cdots & & 1\alpha\alpha^2 \cdots \alpha^{e-1} & \end{array}\right],$$

and the syndrome $S(x)$ can be generated as two parts, $S_1(x)$ and $S_2(x)$, due to the factors $x^b - 1$ and $\mathbf{p}(x)$, respectively.

$$\begin{aligned} S(x) &= (S_1(x), S_2(x)), \\ S_1(x) &\equiv E(x) \bmod (x^b - 1), \\ S_2(x) &\equiv E(x) \bmod \mathbf{p}(x). \end{aligned}$$

Since $\mathrm{GCD}(e, b) = 1$, that is, e and b are relatively prime, there are no two identical columns in the matrix $\mathbf{H}$, and hence the single errors are correctable. Also $S_1(x)$ provides the byte error pattern, and it is not equal to that of single-bit errors. Next we prove that together the two components $S_1(x)$ and $S_2(x)$ detect random double-bit errors as follows: if random double-bit errors occur, we need to consider the possibility that the erroneous bits may be in the same byte or in different bytes. In the first case, it is a byte error and it is detectable by $S_1(x)$. In the latter case, the assumption $\mathrm{GCD}(e, b) = 1$ guarantees double-bit errors having different values at least for S_1 or S_2. Therefore double-bit errors have (1) different values for S_2 and also S_1, (2) different values for S_1 and the same values for S_2, or (3) the same values for S_1 and different values for S_2. The random double-bit errors are detectable, and the code is the SEC-DED-SbED code.

Since the exponent of the primitive polynomial $\mathbf{p}(x)$ of degree $l = R - b$ is $e = 2^{R-b} - 1$, the code length (in bits) of this code can be expressed as

$$N = b \cdot (2^{R-b} - 1).$$

Q.E.D.

Example 6.5

Let the primitive polynomial of degree 3 be $\mathbf{p}(x) = x^3 + x + 1$ and $b = 4$, $l = 3$. Then $\mathrm{GCD}(e, b) = \mathrm{GCD}(2^3 - 1, 4) = 1$, and the following (28, 21) cyclic SEC-DED-S4ED code can be designed:

$$\mathbf{H} = \left[\begin{array}{cccc:cccc:cccc:cccc:cccc:cccc:cccc}
1 & & & & 1 & & & & 1 & & & & 1 & & & & 1 & & & & 1 & & & & 1 & & & \\
 & 1 & & & & 1 & & & & 1 & & & & 1 & & & & 1 & & & & 1 & & & & 1 & & \\
 & & 1 & & & & 1 & & & & 1 & & & & 1 & & & & 1 & & & & 1 & & & & 1 & \\
 & & & 1 & & & & 1 & & & & 1 & & & & 1 & & & & 1 & & & & 1 & & & & 1 \\
\hline
1 & & & 1 & & 1 & 1 & 1 & & & 1 & & 1 & 1 & 1 & & & 1 & & 1 & 1 & 1 & & & 1 & & 1 & 1 \\
 & 1 & & 1 & 1 & 1 & & & 1 & & 1 & 1 & 1 & & & 1 & & 1 & 1 & 1 & & & 1 & & 1 & 1 & 1 & \\
 & & 1 & & 1 & 1 & 1 & & & 1 & & 1 & 1 & 1 & & & 1 & & 1 & 1 & 1 & & & 1 & & 1 & 1 & 1
\end{array}\right] \qquad (6.2)$$

6.1.2 Generalized Burst Error Detecting SEC-DED Codes

Consider a class of generalized burst error detecting SEC-DED codes (i.e., generalized SEC-DED-BED codes)[KANE83] that include the codes previously mentioned. Here it will be shown that the foregoing codes are the burst error detecting SEC-DED codes.

Definition 6.1 A code of length N, where N is an integer multiple of b, is said to be *b-grouped parity checkable* if it can be divided equally into b groups in an interlaced form and the mod-2 sum of the data included in each group has constant value regardless of the input codewords. □

The concept of this b-grouped parity checking is shown in Figure 6.4.

Definition 6.2 Let b be the byte length in bits and R be a check-bit length of the code defined by the matrix $\mathbf{H}$. A $b \times R$ binary matrix $\mathbf{A}$ whose any b columns are linearly independent, that is, have *rank b*, is referred to as a *grouping matrix*, whereby R row vectors of the $\mathbf{H}$ matrix are collected into b groups in which vectors are modulo 2 added. □

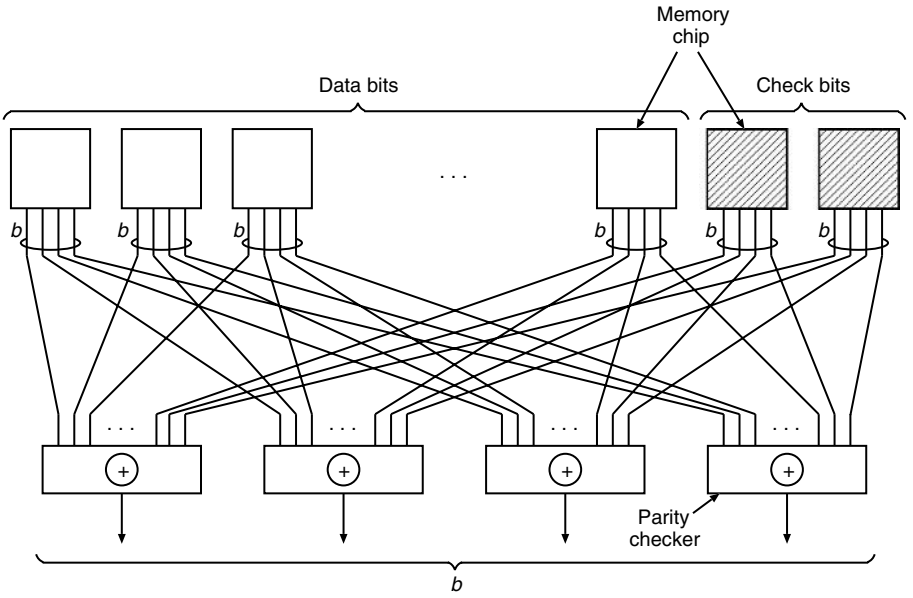


Figure 6.4 *b-Grouped parity checking.* Source: *[KANE83]. © 1983 IECE Japan.*

Let the **H** matrix of a code be denoted by its R row vectors $P_0, P_1, \ldots, P_{R-1}$. The product $\mathbf{A} \cdot \mathbf{H}$ is a matrix **F**, and its row vectors are denoted by $F_0, F_1, \ldots, F_{b-1}$. Then F_i is the result of a bit-by-bit (mod 2) sum of all P_j's for which $a_{ij} = 1$. Here $a_{i,j} \in \{0, 1\}$ is an element of **A**:

$$\begin{bmatrix} F_0 \\ F_1 \\ \vdots \\ F_{b-1} \end{bmatrix} = [\mathbf{A}] \cdot \begin{bmatrix} P_0 \\ P_1 \\ \vdots \\ P_{R-1} \end{bmatrix}.$$

Figure 6.5 shows the multiplication of **A** to **H** where byte length $b = 4$ and check-bit length $R = 8$. In this example the **H** matrix defines the type of code stated in Example 6.4.

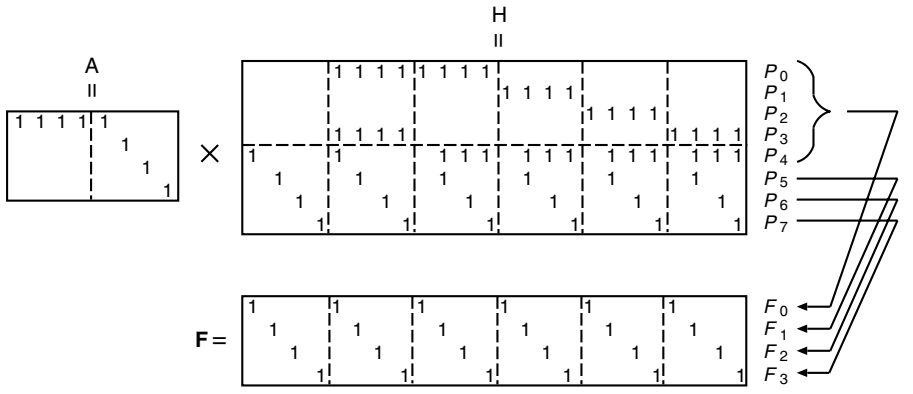


Figure 6.5 *Example of grouping matrix* ***A****.* Source: *[KANE83]. © 1983 IECE Japan.*

The first five rows of **H** are added to obtain F_0. The sixth, seventh, and eighth rows of **H** are the same as F_1, F_2, and F_3, respectively.

The following theorem is relevant to the design of b-grouped parity checkable codes using the grouping matrix **A**.

Theorem 6.7 *A code is b-grouped parity checkable, if and only if there exists a matrix **A** such that the product of **A** and **H** is identical to the concatenation of the $b \times b$ identity matrices.*

Proof Let the concatenation of the $b \times b$ identity matrix be **F**, and let the codeword be W. Then from $\mathbf{F} = \mathbf{A} \cdot \mathbf{H}$ we have

$$\begin{aligned} W \cdot \mathbf{F}^T &= W \cdot (\mathbf{A} \cdot \mathbf{H})^T = (W \cdot \mathbf{H}^T) \cdot \mathbf{A}^T \\ &= 0 \qquad (\because W \cdot \mathbf{H}^T = 0). \end{aligned}$$

Note that the mod-2 sum of the data included in each group has a constant value (zero) regardless of the codeword W. Since each column vector of **F** has weight 1, and each row vector of **F** has equal constant weight, the code derived from the **H** matrix is b-grouped parity checkable. This proves the necessity of the condition, and its sufficiency can be proven in a similar manner. Q.E.D.

Figure 6.6 shows the b-grouped parity checking of the arbitrary codewords, W_1 and W_2.

Theorem 6.7 states that if the addition of subsets of row vectors of **H** as prescribed by **A** yields a concatenation of $b \times b$ identity matrices, then the code represented by **H** is b-grouped parity checkable.

Theorem 6.8 *Assume that the **H** matrix of a code **C** has distinct columns, and the code is b-grouped parity checkable. Then **C** is an SEC-DED-BED code with code length in bits*

$$N = b \cdot 2^{R-b}.$$

Proof Let $P_0, P_1, \ldots, P_{r-1}$, be row vectors of **H**. Since the vector sum of some P_i's yields F_i's, $0 \leq i \leq b-1$, there cannot be an all-0 column vector in **H**. Also, by the assumption, these column vectors in **H** are distinct. Thus the code can correct all single-bit errors.

According to Theorem 6.7, the mod-2 sum of $F_0, \ldots, F_{b-1}$, yields an all-1 row vector. This means that every column in **H** has odd weight. Therefore this code can also detect all double-bit errors.

Next let us examine a vector obtained by multiplying the syndrome to the grouping matrix **A**. It is easy to see that the result is identical to the single-byte error pattern. This pattern can indicate whether the error is a correctable single-bit error or an uncorrectable byte error. On the other hand, since the product of **A** and **H** yields the concatenation of $b \times b$ identity matrices, this byte error pattern will always indicate any b-adjacent errors, that is, burst errors. This means the code can detect single b-bit burst errors (see Exercise 2.29).

Let us consider the bit length of this code. Since the product of **A** and **H**, which is **F**, is the concatenation of identity matrices, the bit having 1 in the vector F_i shows that the

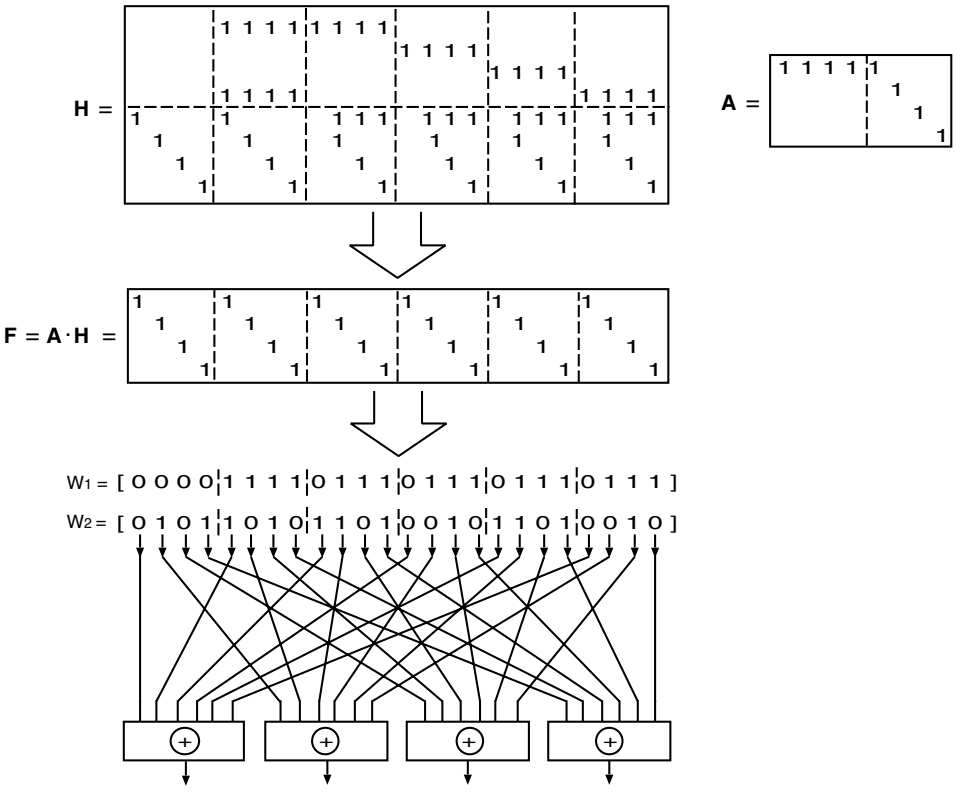


Figure 6.6 *b-Grouped parity checking of codewords W_1 and W_2.*

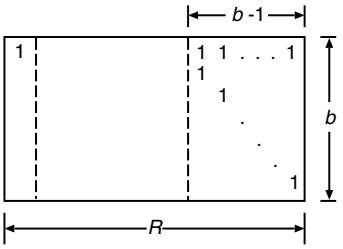
corresponding **H** matrix column vectors are included uniquely in the i-th group. Let $(x_0, x_1, \ldots, x_{R-1})^T$ represent the **H** matrix column vector included in the i-th group. These column vectors should satisfy the following relation. Note that only the i-th element of the vector in the left-hand side is 1, and the other elements are all 0's.

$$i)\begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}_b = [\mathbf{A}]_{b\times R} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{R-1} \end{bmatrix}. \tag{6.3}$$

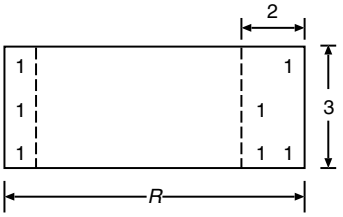
According to Definition 6.2, the rank of **A** is b. From linear algebra we know that the first-order equation of (6.3) has 2^{R-b} solutions of $(x_0, x_1, \ldots, x_{R-1})^T$. Therefore the bit length N of this code is $b \cdot 2^{R-b}$. Q.E.D.

Figure 6.7 shows some grouping matrices for SEC-DED-BED codes of Theorems 6.2 through 6.6 [KANE83]. As the figure shows, we can prove that the codes of Theorems 6.2,

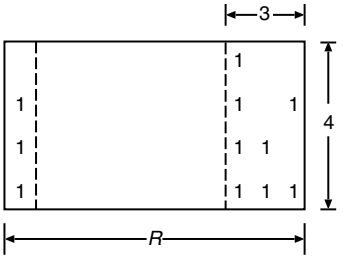
Code of Theorem 6.2 : **A** =



Code of Theorem 6.3 : $b = 3$: **A** =

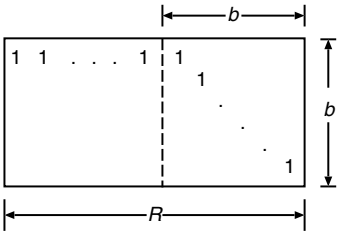


$b = 4$: **A** =



The codes for $b \geq 5$ are not b-grouped parity checkable.

Code of Theorem 6.5 : **A** =



Code of Theorem 6.6 : **A** =

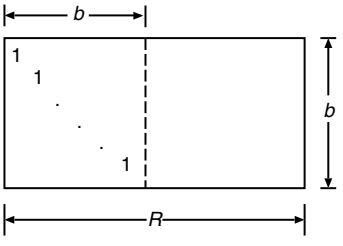


Figure 6.7 *Grouping matrices for existing SEC-DED-BED codes. (In order to have double-bit error detection ability in the codes of Theorems 6.2 and 6.3, an all-1 row vector is added to H.)*

6.3 (for $b \leq 4$), and 6.5 are equivalent to each other. The codes of Theorem 6.6 have a cyclic structure and have smaller code length than the other codes. However, these are also b-grouped parity checkable codes.

The grouping matrix **A** has the following properties:

Property 1 If every column in **A** has odd weight, then every column in **H** of the SEC-DED-BED code has also odd weight. For examples, see Figure 6.5 and Eq. (6.4).

Property 2 If every column in **A** has weight 1, then the **H** matrix of the SEC-DED-BED code has r weight-1 check columns. For an example, see Figure 6.5 and Eq. (6.4).

Property 3 If every column in **A** has weight 1 and every row in **A** has constant weight, then the r columns for check bits in **H** of the SEC-DED-BED code are clustered together in R/b places. For an example, see Eq. (6.4) and Theorem 6.9.

Now consider another b-grouped parity checkable code, as shown in Eq. (6.4), whose matrix **A** is different from those shown in Figure 6.7. The matrix **A** of this code satisfies properties 1, 2, and 3. The **H** matrix has all odd-weight columns and satisfies the conditions of the equal-weight-row code, and this also satisfies the rotational code with degree two, or 2-modularized code.

$$
\mathbf{H} = \left[\begin{array}{cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc}
1&&& & 1&&& & 1&1&1&1 & 1&1&1& & 1&&& & &&& & &&& & &1&1&1 & &1&1& & &&& \\
&1&& & &1&1&1 & 1&1&& & 1&&&1 & &1&& & &&& & &&1&1 & 1&&& & 1&1&&1 & &&& \\
&&1&1 & 1&1&1& & &&1& & 1&&&1 & &&1& & &&&1 & 1&1&& & &&& & 1&&1&1 & &&& \\
1&1&1&1 & &&&1 & &&&1 & &1&1& & &&&1 & 1&1&1& & &&& & &&& & &1&1&1 & &&& \\
\hline
&&& & &&& & &1&1&1 & &1&1& & &&& & 1&&& & 1&&& & 1&1&1&1 & 1&1&1& & 1&&& \\
&&& & &&1&1 & 1&&& & 1&1&&1 & &&& & &1&& & &1&1&1 & 1&1&& & 1&&&1 & &1&& \\
&&&1 & 1&1&& & &&& & 1&&1&1 & &&& & &&1&1 & 1&1&1& & &&1& & 1&&&1 & &&1& \\
1&1&1& & &&& & &&& & &1&1&1 & &&& & 1&1&1&1 & &&&1 & &&&1 & &1&1& & &&&1
\end{array}\right],
$$

$$
\mathbf{A} = \left[\begin{array}{cccc|cccc}
1&&& & 1&&& \\
&1&& & &1&& \\
&&1& & &&1& \\
&&&1 & &&&1
\end{array}\right].
\tag{6.4}
$$

The error detection circuit of the b-grouped parity checkable codes can be systematically obtained by multiplying the syndrome to matrix **A**. Figure 6.8 shows this circuit of the code shown in Eq. (6.4). Figure 6.9 shows the relation between the information-bit length K and the check-bit length R of the SEC-DED-BED codes for $b = 3,\ 4,\ 8$, and 12 bits.

Rotational Burst Error Detecting SEC-DED Codes Let us design the rotational generalized SEC-DED-BED codes [KANE83].

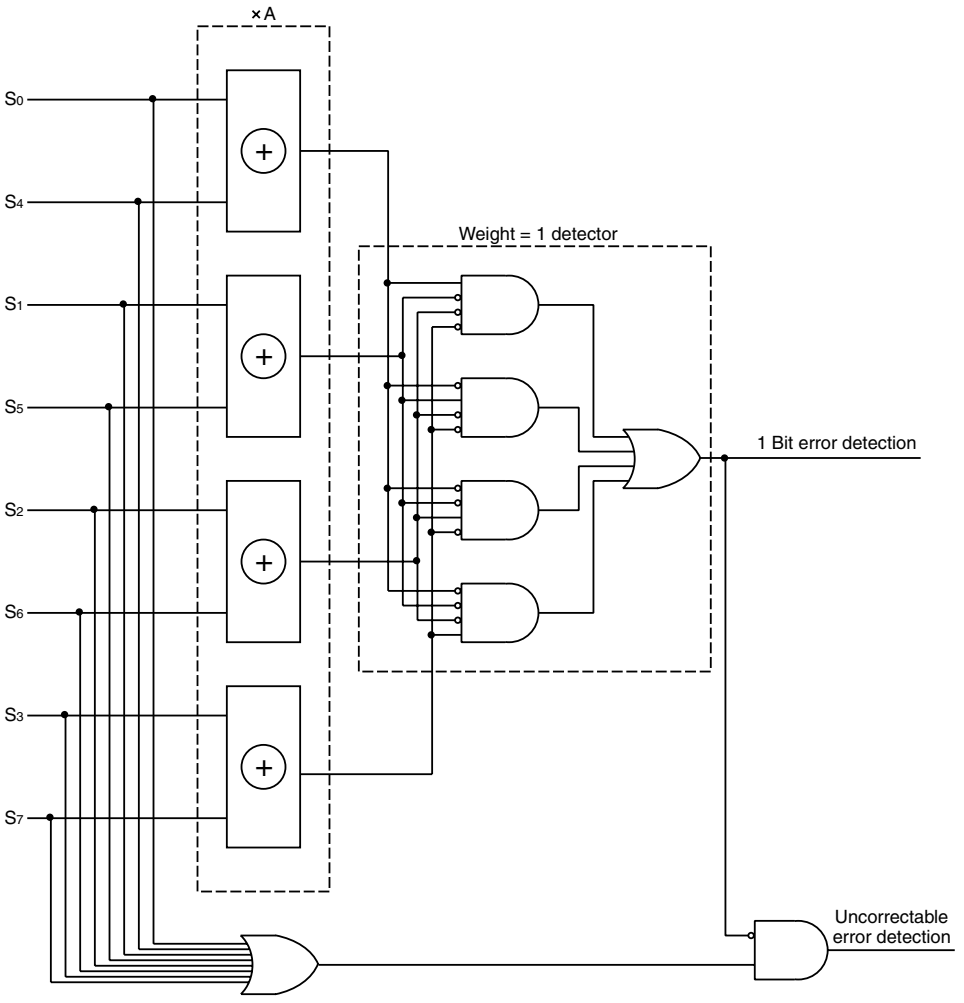


Figure 6.8 *Error detection circuit for the (40, 32) SEC-DED-BED code shown in Eq. (6.4).* Source: *[KANE83]. © 1983 IECE Japan.*

Theorem 6.9 *If the grouping matrix* $\boldsymbol{A}$ *is given as a concatenation of (R/b) b-th degree identity matrices* $\boldsymbol{I}_b$,

$$\boldsymbol{A} = [\boldsymbol{I}_b\ \boldsymbol{I}_b\ \cdots \boldsymbol{I}_b]_{R/b},$$

then the code generated by $\boldsymbol{A}$ *is a rotational SEC-DED-BED code whose* $\boldsymbol{H}$ *matrix has* R/b *submatrices. The code length (in bits) of this code is given as follows:*

$$N = b \cdot \sum_{\substack{m|(R/b) \\ m\,:\,odd}} \mu(m) \cdot 2^{R/m-b}. \tag{6.5}$$

Here the summation is performed over the odd divisors of R/b, *and* $\mu(m)$ *is the Möbius function defined in Section 3.5.*

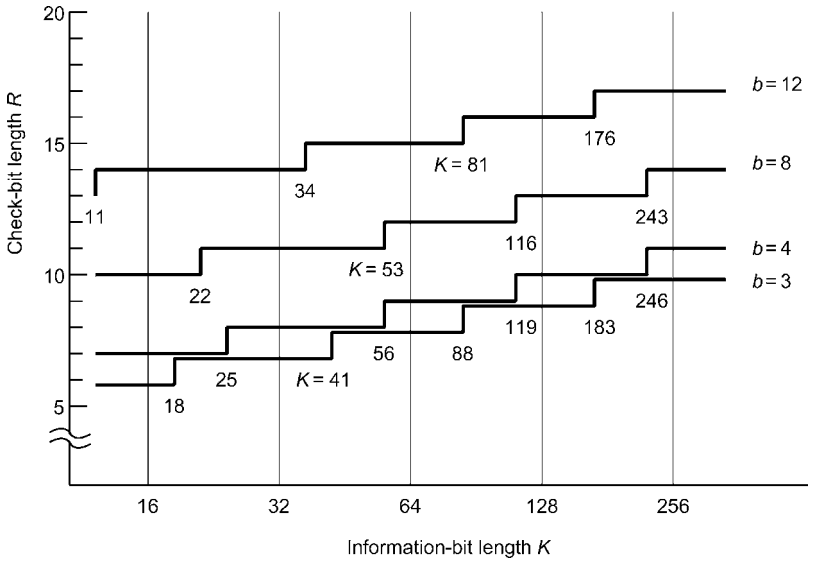


Figure 6.9 *Relationship between check-bit lengths and information-bit lengths of the SEC-DED-BED codes.*

Theorem 6.9 can be proved by using the definitions of the grouping matrix **A**, the b-grouped parity checkable code, and the rotational code. The code length can be obtained by using the modified Möbius inversion formula given in Theorem 5.4. From property 1 of the grouping matrix **A**, this code is an odd-weight-column code. The code length N bits of this code is shown in Table 6.2.

An example of the rotational (45, 36) SEC-DED-BED code (with $b = 3$ bits) whose rotational degree is three is given in Eq. (6.6). Earlier another example of a code with rotational degree two was presented in Eq. (6.4).

TABLE 6.2 Code Length (in Bits) of the Rotational SEC-DED-BED Code

R/*b* \ *b* (Bits)	1	2	3	4	5	6
2	2/2	8/8	24/24	64/64	160/160	384/384
3	3/4	30/32	189/192	1,020/1,024	5,115/5,120	24,570/24,576
4	8/8	128/128	1,536/1,536	16,384/16,384	163,840/163,840	1,572,864/1,572,864
5	15/16	510/512	12,285/12,288	.	.	.

Source: [KANE83]. © 1983 IECE Japan.

Note: Column 1 gives the rotational odd-weight-column SEC-DED code. The numerator gives the rotational code length in bits and the denominator the nonrotational code length in bits.

$$
\mathbf{H} = \left[\begin{array}{ccc|ccc|ccc|ccc|ccc||ccc|ccc|ccc|ccc|ccc||ccc|ccc|ccc|ccc|ccc}
1&& & &1& & 1&&1 & 1&1&1 & 1&& & && & && & && & &&1 & && & && & 1&1& & &&1 & &1& & && \\
1&1&1 & 1&1&1 & && & && & &1& & && & && & && & &1& & && & 1&&1 & 1&&1 & &1& & && & && \\
&1&1 & && & 1&1&1 & && & &&1 & && & && & && & 1&& & && & &1& & &&1 & 1&1& & 1&&1 & && \\
\hline
&& & 1&1& & &&1 & &1& & && & 1&& & &1& & 1&&1 & 1&1&1 & 1&& & && & && & && & &&1 & && \\
1&&1 & 1&&1 & &1& & && & && & 1&1&1 & 1&1&1 & && & && & &1& & && & && & && & &1& & && \\
&1& & &&1 & 1&1& & 1&&1 & && & &1&1 & && & 1&1&1 & && & &&1 & && & && & && & 1&& & && \\
\hline
&& & && & && & &&1 & && & && & 1&1& & &&1 & &1& & && & 1&& & &1& & 1&&1 & 1&1&1 & 1&& \\
&& & && & && & &1& & && & 1&&1 & 1&&1 & &1& & && & && & 1&1&1 & 1&1&1 & && & && & &1& \\
&& & && & && & 1&& & && & &1& & &&1 & 1&1& & 1&&1 & && & &1&1 & && & 1&1&1 & && & &&1
\end{array}\right]
$$

$$
\mathbf{A} = \left[\begin{array}{ccc|ccc|ccc}
1&& & 1&& & 1&& \\
&1& & &1& & &1& \\
&&1 & &&1 & &&1
\end{array}\right]. \tag{6.6}
$$

6.1.3 Byte Error Detecting SEC-DED Codes (SEC-DED-S*b*ED Codes)

We can say that the SEC-DED-SbED code is a special class of the SEC-DED-BED code in the sense that the burst error is confined to the bounds of one byte. A BED code is also an SbED code, but not necessarily the other way around. Therefore the SbED codes are likely to have fewer check bits than the BED codes. For one of the practical code parameters of information-bit length $K = 64$ and byte length $b = 4$ bits, an SEC-DED-SbED code could have a check-bit length of $R = 8$, which is the same check-bit length of the SEC-DED code for $K = 64$. On the other hand, from Figure 6.9, the SEC-DED-BED code requires a check-bit length of $R = 9$ for $K = 64$.

The SEC-DED-SbED codes are important from the practical point of view, and hence they have been actively studied. Some excellent codes are presented in [CHEN83, KANE84, KANE85, DUNN83, DUNN85]. Chen [CHEN83] uses the structure of *orthogonal flats* in finite Euclidean geometry to design these codes. Codes with $b = 3$ bits precisely meet the upper bound. For $b = 4$ bits, Kaneda, Chen, Tsuchimoto, Boyarinov, Davydov, and Holman [KANE84, CHEN84, TSUC86, BOYA87, DAVY91, HOLM99] provide some practical codes especially for $K = 64$ bits. Dunning [DUNN85] develops some excellent codes with $R = b + 2$. However, the "best" code that meets the upper bound of the code length for an arbitrary byte length b has not yet been found. Here we introduce the codes with $b = 4$ and $R = b + 2$.

Codes for b* $=$ *4 The SEC-DED-SbED codes with 4-bit byte (i.e., $b = 4$) [KANE84] are important from the practical stand point because 4-bit byte organized RAM chips have been commonly used in high-speed memories since the mid-1980s. Compared to the SEC-DED-BED code, the check-bit length can be decreased by one bit for an information-bit length of $K = 64$. This is the same check-bit length as that of for the SEC-DED code with $K = 64$. We can design the (72, 64) SEC-DED-S4ED code by the following construction methods: The code indicated here is not restricted to the length of $K = 64$ bits, but the code can take any value of K and N.

Construction 1

Step 1. Let R be an arbitrary but positive even integer ($R > 3$). Let G denote a column vector having $R/2$ binary elements. Vector G is arbitrary, but $G =$ all 1's vector is recommended to use in order to reduce the **H** matrix weight.

Step 2. Let F_q denote a column vector having $R/2$ binary elements. The weight of F_q must be even for an odd weight of G and must be odd for an even weight of G. The number of F_q's is $2^{R/2-1}$, where $q = 0,\ 1,\ \ldots,\ 2^{R/2-1} - 1$.

Step 3. Two column vectors F_i and F_j are selected arbitrarily from the F_q's, where $0 \le i \ne j \le 2^{R/2-1} - 1$. Using F_i and F_j, define the following $R \times 4$ matrix:

$$\mathbf{H}_{i,j} = \begin{bmatrix} G + F_i + F_j & G + F_i + F_j & F_i & F_j \\ F_i & F_j & G + F_i + F_j & G + F_i + F_j \end{bmatrix}.$$

Here the plus sign represents the mod-2 sum of the column vectors.

Step 4. Construct the **H** matrix as follows:

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{0,1} & \mathbf{H}_{0,2} & \mathbf{H}_{0,3} & \ldots & \mathbf{H}_{i,j} & \ldots & \mathbf{H}_{p-1,p} \end{bmatrix}, \qquad p \le 2^{R/2-1} - 1 \tag{6.7}$$

Construction 2 Let W be a row vector of N 0's followed by N 1's, where $N = n \cdot b$. Define

$$\mathbf{H}_{R+1} = \left[\begin{array}{c} W \\ \hline \mathbf{H}_R\ \mathbf{H}_R \end{array}\right],$$

where $\mathbf{H}_R$ is the parity-check matrix of the code $\mathbf{C}(R,\ b)$ with code length n bytes and check-bit length R. Then the code $\mathbf{C}(R + 1,\ b)$ defined by the parity-check matrix $\mathbf{H}_{R+1}$ is an SEC-DED-SbED code with a code length of $2n$ bytes. Here $\mathbf{C}(R,\ b)$ denotes an SEC-DED-SbED code with R check bits and b bits in a byte. This is the same code extension method as that shown in Theorem 5.8.

Theorem 6.10 *The code designed by construction 1 is an odd-weight-column SEC-DED-S4ED code with an even number of check bits larger than 3 and a code length in bytes* $n = 2^{R-3} - 2^{R/2-2}$.

Proof Obviously, the column vectors in **H** shown in Eq. (6.7) have odd weight. Now let us assume that F_i has odd weight. In this case G must have even weight. When F_i has even weight, the proof is the same, as will be shown.

1. *Proof of single-error correction capability.* The weight of F_i is odd and the weight of $G + F_i + F_j$ is even. Then we assume that the following equations are satisfied for the column vectors:

$$F_i = F_i',$$
$$G + F_i + F_j = G + F_i' + F_j'.$$

If $F_i = F_i'$, then by construction 1, F_j should not be equal to F_j'. However, we can derive $F_i = F_i'$ and $F_j = F_j'$ from the equations above, and this is a contradiction. Therefore the column vectors in **H** shown in Eq. (6.7) are distinct. Consequently the codes are SEC-DED codes.

2. *Proof of single 4-bit byte error detection capability.* Double-bit errors within each byte can be detected through the DED capability. We should verify the error detection capability of triple-bit errors and quadruple-bit errors within a byte.

a. *Triple-bit errors*. In the **H** shown in Eq. (6.7), replace arbitrarily three column vectors in the byte corresponding to the erroneous triple-bit errors in this byte, and then do a mod-2 sum of these vectors. The resultant column vector should be equal to the syndrome caused by the triple-bit errors. It can be easily shown that either the lower-half $R/2$ bits or the upper-half $R/2$ bits in the syndrome are equal to G. From construction 1, F_i is not equal to F_j. Therefore $G + F_i + F_j$ cannot be equal to G. So triple-bit errors within a byte can be detected.

b. *Quadruple-bit errors*. In the **H** shown in Eq. (6.7), take a mod-2 sum of the four column vectors in a byte corresponding to the erroneous quadruple-bit errors in this byte. Obviously, the resultant column vector has even weight and is not equal to the all-0 vector. Thus the quadruple-bit errors within a byte can be detected.

Note that the number of F_q's is $2^{R/2-1}$. In addition the number of $\mathbf{H}_{i,j}$'s is the combination number of two elements from $2^{R/2-1}$ elements. Hence the code length in bytes is equal to $n = 2^{R/2-2}(2^{R/2-1} - 1) = 2^{R-3} - 2^{R/2-2}$. Q.E.D.

If an odd number of check bits is required, construction 2 can be applied to the code in Theorem 6.10. The codes obtained have the code length of $2^{R-2} - 2^{R/2-1}$ bytes for R ($=$ odd) check bits.

Example 6.6 [KANE84]

For $b = 4$ and $R = 8$, let G be an all-1 column vector:

$$G = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

This G is even weight; therefore, F_q must be odd weight. There are eight F_q's:

$$\begin{array}{cccccccc} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}$$

and among these F_q's, two columns are chosen. Then the $\mathbf{H}_{ij}$ can be constructed. There are $\binom{8}{2} = 28\,\mathbf{H}_{ij}$'s. Examples for $\mathbf{H}_{01}$ and $\mathbf{H}_{02}$ are indicated as follows:

$$\mathbf{H}_{01} = \left[\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{array}\right] \qquad \mathbf{H}_{02} = \left[\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{array}\right].$$

The complete **H** matrix for $b = 4$ and $R = 8$ is given in Eq. (6.8).[a] This is a (112, 104) SEC-DED-S4ED code.

$$\mathbf{H} = \left[\begin{array}{cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc}
1&1&&& 1&1&&& &&&1& &&&1& &&&1& &&&1& 1&1&&& 1&1&&& &&&1& &&&1& &&&1& &&&1 \\
1&1&&& &&&1& 1&1&&& &&&1& &&&1& 1&1&&& &&&1& &&&1& 1&1&&& &&&1& &&&1& 1&1&& \\
&&&1& 1&1&&& 1&1&&& &&&1& 1&1&&& &&&1& &&&1& &&1&& &&1&& 1&1&1&1& &&1&& 1&1&1&1 \\
&&1&& &&1&& &&1&& &&1&& 1&1&1&1& 1&1&1&1& 1&1&1&1& 1&1&&& 1&1&&& 1&1&&& &&&1& &&&1 \\
\hline
&&1&1& &&1&1& &1&&& &1&&& &1&&& &1&&& &&1&1& &&1&1& &1&&& &1&&& &1&&& &1&& \\
&&1&1& &1&&& &&1&1& &1&&& &1&&& &&1&1& &1&&& &1&&& &&1&1& &1&&& &1&&& &&1&1 \\
&1&&& &&1&1& &&1&1& &1&&& &&1&1& &1&&& &1&&& 1&&&& 1&&&& 1&1&1&1& 1&&&& 1&1&1&1 \\
1&&&& 1&&&& 1&&&& 1&&&& 1&1&1&1& 1&1&1&1& 1&1&1&1& &&1&1& &&1&1& &&1&1& &1&&& &1&&
\end{array}\right.$$

$$\left.\begin{array}{cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc|cccc}
1&1&&& &&&1& &&&1& &&&1& &&&1& 1&1&&& 1&1&1&1& 1&1&1&1& 1&1&1&1& &&1&& 1&1&1&1& 1&1&1&1 \\
&&&1& &&1&& 1&1&1&1& 1&1&1&1& &&1&& 1&1&1&1& &&&1& &&&1& 1&1&&& &&&1& 1&1&1&1& &&1& \\
1&1&1&1& 1&1&&& &&&1& 1&1&&& &&&1& &&&1& &&&1& 1&1&&& &&&1& &&&1& &&1&& 1&1&1&1 \\
&&&1& 1&1&&& 1&1&&& &&&1& &&&1& &&&1& 1&1&&& &&&1& &&&1& &&&1& &&&1& &&&1 \\
\hline
&&1&1& &1&&& &1&&& &1&&& &1&&& &&1&1& 1&1&1&1& 1&1&1&1& 1&1&1&1& 1&&&& 1&1&1&1& 1&1&1&1 \\
&1&&& 1&&&& 1&1&1&1& 1&1&1&1& 1&&&& 1&1&1&1& &1&&& &1&&& &&1&1& &1&&& 1&1&1&1& 1&&& \\
1&1&1&1& &&1&1& &1&&& &&1&1& &1&&& &1&&& &1&&& &&1&1& &1&&& &1&&& 1&&&& 1&1&1&1 \\
&1&&& &&1&1& &&1&1& &1&&& &1&&& &1&&& &&1&1& &1&&& &1&&& &1&&& &1&&& &1&&
\end{array}\right.$$

$$\left.\begin{array}{cccc|cccc|cccc|cccc}
&&1&& 1&1&1&1& &&1&& &&1& \\
1&1&1&1& &&1&& 1&1&1&1& &&&1 \\
1&1&1&1& &&&1& &&&1& 1&1&1&1 \\
&&&1& 1&1&1&1& 1&1&1&1& 1&1&1&1 \\
\hline
1&&&& 1&1&1&1& 1&&&& 1&&& \\
1&1&1&1& 1&&&& 1&1&1&1& &1&& \\
1&1&1&1& &1&&& &1&&& 1&1&1&1 \\
&1&&& 1&1&1&1& 1&1&1&1& 1&1&1&1
\end{array}\right] \tag{6.8}$$

The code shown in Figure 6.10 is a 2-modularized odd-weight-column (72, 64) SEC-DED-S4ED code shortened and converted from the original code of Eq. (6.8). Check columns are concentrated in two bytes using row operations (i.e., successive addition of one row vector to another).

As a consequence we can design the SEC-DED-S4ED code having the same code parameters of check-bit length $R = 8$ and information-bit length $K = 64$ as the SEC-DED code. Furthermore the parallel encoding / decoding circuit of this (72, 64) SEC-DED-S4ED code has almost the same hardware gate amount as that of the (72, 64) SEC-DED code. Figure 6.11 shows the error detection circuit of the code shown in Eq. (6.8).

The codes with $b = 4$ bits are important from a practical stand point. The code shown in Figure 6.10 has been recently applied to a computer system [SUNM95]. In this system, 64K-word $\times$ 4-bit byte ($b = 4$) organized high-speed static RAMs are used in the main storage units.

The computer-generated code with $K = 64$ bits, and $R = 8$ bits shown in Figure 6.12 has also been applied to a computer system [TSUC86]. This code has the following properties:

1. Odd-weight-column code (see Section 3.2).
2. Minimum-weight & equal-weight-row code (see Section 3.1).
3. Eight bits error detection over any two bytes.

The codes shown in Figures 6.13 and 6.14 are also 2-modularized or nearly 4-modularized odd-weight-column (72, 64) SEC-DED-S4ED codes, respectively [HOLM99] [BOYA87].

---

[a]Source of Eq. (6.8): [*KANE84*]. © 1984 IEEE.

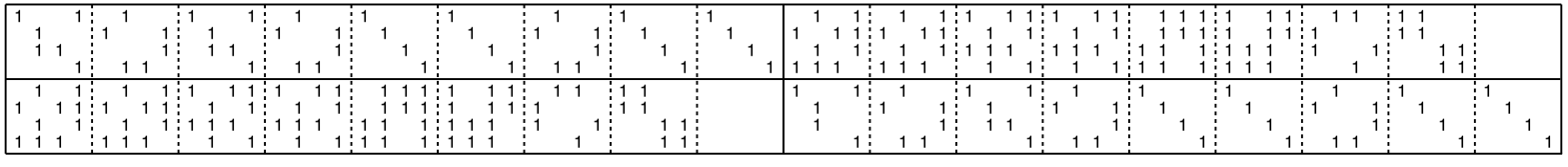


Figure 6.10 *2-Modularized odd-weight-column (72, 64) SEC-DED-S4ED code.* Source: *[KANE84]. © 1984 IEEE.*

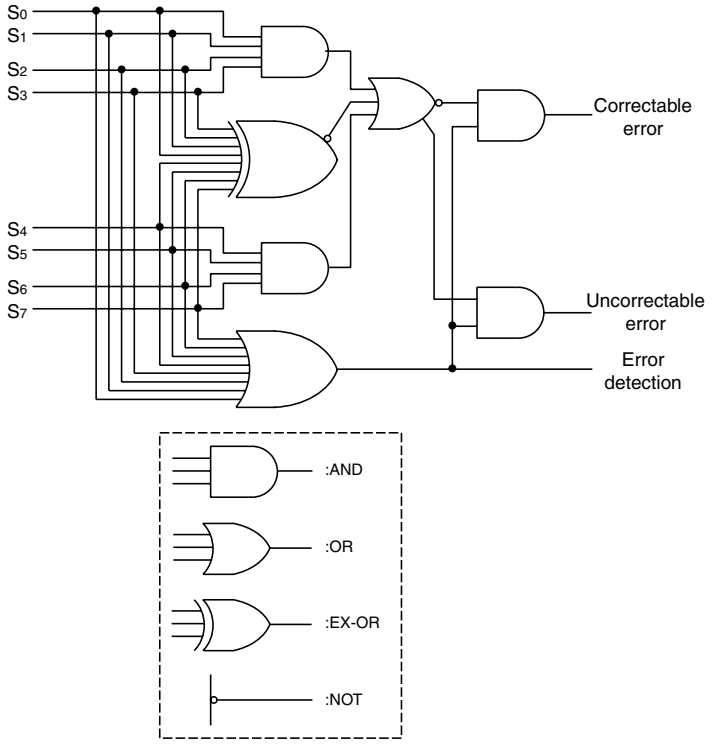


Figure 6.11 *Error detection circuit of the code shown in Eq. (6.8).* Source: *[KANE84]. © 1984 IEEE.*

The three codes of Figures 6.12 to 6.14 are optimal in the sense that the number of 1's in each **H** matrix is minimum (i.e., 216), which is equal to that of the optimal (72, 64) odd-weight-column SEC-DED code shown in Figure 4.2 in Subsection 4.1.1. These codes are also equal-weight-row codes, and hence are minimum-weight & equal-weight-row codes. Other codes are presented in Figures 6.15 and 6.16 [DAVY91] [CHEN84]. Table 6.3 shows an evaluation of these six codes.

Codes for R = b + 2 and Others Dunning suggests an excellent design method for a rotational SEC-DED-SbED code with $R = b + 2$ bits [DUNN85]. The method deals with the codes with other than $b = 4$ bits and the code with $R = b + 2$ bits, as we explain here.

Theorem 6.11 [DUNN85] *Let $b > 2$ be given and $(b - 1) \times (b - 2)$ matrix $\boldsymbol{T} = [T_{ij}]$ be defined by $T_{ij} = 1$ if $i = j$ or $i = j + 1$ and $T_{ij} = 0$ otherwise. The basic $\boldsymbol{H}$ matrix of an SEC-DED-SbED code with byte length b bits, check-bit length $R = b + 2$, and code length of $b + 2$ bytes is given by*

$$\boldsymbol{H}_0 = \begin{bmatrix} 0 & 1 & 0_{b-2} \\ 1 & 1 & 1_{b-2} \\ 0 & 1 & 0_{b-2} \\ \boldsymbol{0}_{b-1,2} & & \boldsymbol{T} \end{bmatrix}_{R \times b},$$

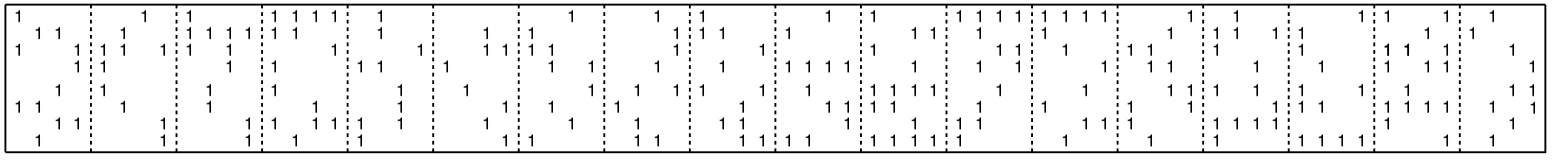
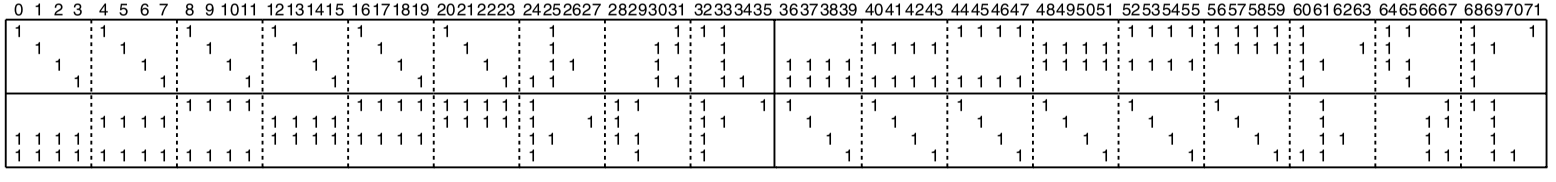


Figure 6.12 *Optimal odd-weight-column (72, 64) SEC-DED-S4ED code.* Source: *[TSUC86]. © 1986 Nikkei Business Publishers Inc.; republished by permission.*



Bits numbered by 26, 27, 34, 35, 62, 63, 70, and 71 are check bits.

Figure 6.13 *Optimal 2-modularized odd-weight-column (72, 64) SEC-DED-S4ED code.* Source: *[HOLM99].*

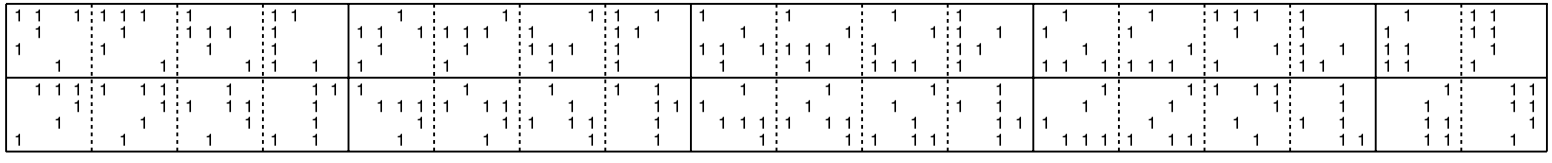


Figure 6.14 *Optimal odd-weight-column (72, 64) SEC-DED-S4ED code (nearly 4-modularized).* Source: *[BOYA87].*

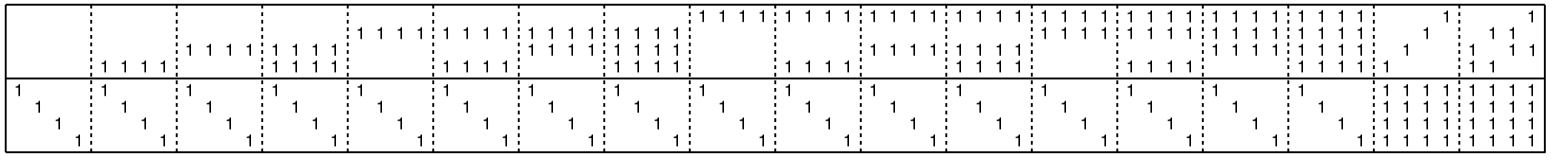


Figure 6.15 *(72, 64) SEC-DED-S4ED code.* Source: *[DAVY91]. © 1991IEEE.*

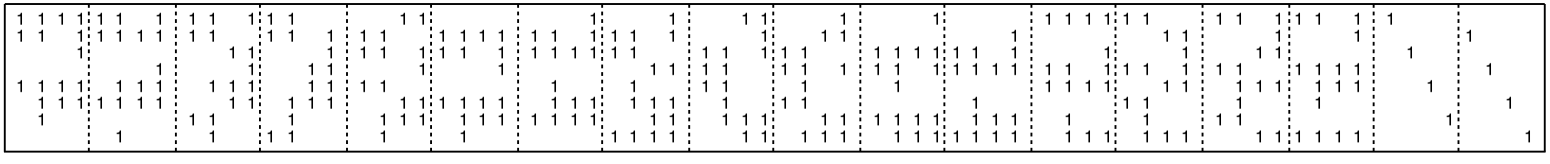


Figure 6.16 *2-Modularized (72, 64) SEC-DED-S4ED code.* Source: *[CHEN84]. © Copyright 1984 by International Business Machines Corporation; republished by permission.*

TABLE 6.3 Evaluation of the (72, 64) SEC-DED-S4ED Codes

Codes	Error Detection Capabilities (%)			Number of 1's in **H**	Modularity
	Triple-bit errors	Double-byte errors	8-Bit errors over any two bytes		
Figure 6.12 code [TSUC86]	43.72	71.44	100	216	No
Figure 6.13 code [HOLM99]	43.61	71.86	80.39	216	2-Modularized
Figure 6.14 code [BOYA87]	43.61	70.87	55.56	216	Nearly 4-modularized
Figure 6.15 code [DAVY91]	51.57	70.37	11.11	236	No
Figure 6.10 code [KANE84]	45.00	72.35	89.54	248	2-Modularized
Figure 6.16 code [CHEN84]	40.23	71.07	84.31	264	2-Modularized

where 0_{b-2} means a row vector of $b-2$ copies of 0, 1_{b-2} means a row vector of $b-2$ copies of 1, and $\mathbf{0}_{b-1,2}$ means $(b-1)\times 2$ all-0 matrix. The resulting $\boldsymbol{H}$ matrix is given by a concatenation of r $\boldsymbol{H}_i$'s whose row vectors are cyclically rotated by one row compared to the adjacent $\boldsymbol{H}_i$'s (see Section 3.5):

$$\boldsymbol{H} = \left[\boldsymbol{H}_0|\boldsymbol{H}_1|\cdots|\boldsymbol{H}_i|\cdots|\boldsymbol{H}_{R-1}\right],$$

$$\boldsymbol{H}_i = \boldsymbol{R}^i \cdot \boldsymbol{H}_0, \quad \boldsymbol{R} = \begin{bmatrix} 0 & 0 & & 0 & 1 \\ 1 & 0 & & 0 & 0 \\ 0 & 1 & & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & & 1 & 0 \end{bmatrix}_{R\times R}.$$

Example 6.7

For $b = 3$, the (15, 10) SEC-DED-S3ED code is shown as

$$\mathbf{H} = \begin{bmatrix} 010 & 001 & 001 & 010 & 111 \\ 111 & 010 & 001 & 001 & 010 \\ 010 & 111 & 010 & 001 & 001 \\ 001 & 010 & 111 & 010 & 001 \\ 001 & 001 & 010 & 111 & 010 \end{bmatrix}.$$

For $b = 4$, the (24, 18) SEC-DED-S4ED code is shown as

$$\mathbf{H} = \begin{bmatrix} 0100 & 0001 & 0011 & 0010 & 0100 & 1111 \\ 1111 & 0100 & 0001 & 0011 & 0010 & 0100 \\ 0100 & 1111 & 0100 & 0001 & 0011 & 0010 \\ 0010 & 0100 & 1111 & 0100 & 0001 & 0011 \\ 0011 & 0010 & 0100 & 1111 & 0100 & 0001 \\ 0001 & 0011 & 0010 & 0100 & 1111 & 0100 \end{bmatrix}.$$

The codes defined in Theorem 6.11 are optimal in the sense that these are odd-weight-column rotational codes and also minimum-weight codes; that is, the number of 1's in $\mathbf{H}$ is minimum. As for the SEC-DED-SbED codes with other code parameters [GILS86], the codes defined in Theorem 6.3 [DUNN85] are shown to be best for $b = 5$, and $R = b + 1$ for $b \geq 5$.

Table 6.4 shows the best codes so far obtained for byte length b bits and check-bit length R. Figure 6.17 shows the relation between the information-bit length K and the check-bit length R of the SEC-DED-SbED codes shown in this table for $b = 3$, 4, 8, and 12 bits.

6.2 SINGLE-BYTE ERROR CORRECTING AND DOUBLE-BIT ERROR DETECTING (SbEC-DED) CODES

The SbEC code is capable of correcting single b-bit byte errors, but it is not guaranteed to detect all random double-bit errors spanning over two bytes. Therefore an SbEC code having random double-bit error detection capability, called SbEC-DED code, is required as a small additional redundancy. The SbEC-DbED codes, of course, have an SbEC-DED function but they require much more redundancy.

This section presents some code design methods of the SbEC-DED code that are applicable to the high-speed memories using byte-organized RAM chips. In particular, we will show that the code using elements in a coset of a subfield under addition gives a practical (76, 64) S4EC-DED code with the same check-bit length of $R = 12$ bits as the Hamming S4EC code with byte length $b = 4$ bits and $K = 64$ bits.

6.2.1 Subfield

The definition of a subfield is given in Subsection 2.1.2. The following definition gives more precise description of a subfield.

Definition 6.3 The set

$$\mathbf{0},\ \mathbf{T},\ \mathbf{T}^{\lambda},\ \mathbf{T}^{2\lambda}, \ldots,\ \mathbf{T}^{(p-1)\lambda} = \mathbf{I},$$

where $\mathbf{0}$, $\mathbf{I}$, $\mathbf{T}^{\lambda}$, etc., are elements of $GF(q = 2^b)$, makes up a $GF(p = 2^A)$ *subfield* of $GF(q = 2^b)$ if and only if A is a divisor of b and λ is equal to $(q - 1)/(p - 1)$. □

Example 6.8

Consider the following subfield of $GF(2^4)$. The companion matrix $\mathbf{T}$ corresponding to the fourth degree binary primitive polynomial $\mathbf{g}(x) = x^4 + x + 1$ is written as

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

The set of powered elements, namely $\mathbf{0}$, $\mathbf{T}$, $\mathbf{T}^2$, $\ldots$, $\mathbf{T}^{15} = \mathbf{I}$, forms $GF(2^4)$. Then the subset $\{\mathbf{0},\ \mathbf{T}^5,\ \mathbf{T}^{10},\ \mathbf{T}^{15} = \mathbf{I}\}$ forms a $GF(2^2)$ subfield of $GF(2^4)$ under $A = 2$ and

TABLE 6.4 Code Lengths n (Bytes) of SEC-DED-SbED Codes

Check-bit length R \ Byte length b (bits)	3	4	5	6	7	8	9	$b > 9$
$b+1$	2 (a)	2 (b)	(d) 3	3	3	3	3	3
$b+2$	5	6	7	8 (c)	9	10	11	$b+2$
$b+3$	10	12	15	18 (e)	24	27 (g)	33	$3(b+2)$ (h)
$b+4$	21	28	31	36	56	63	77	$7(b+2)$
$b+5$	42	56	63	75 (f)	120	135	165	$15(b+2)$
$b>5$	$\lfloor(2^{r-1}-1)/3\rfloor$	$2^{r-3}-2^{\lceil r/2-2\rceil}$	$2^{r-b+1}-1$	$5(2^{r-7}-1)$	$8(2^{r-8}-1)$	$9(2^{r-9}-1)$	$11(2^{r-10}-1)$	$(2^{r-b-1}-1)(b+2)$

(a): [CHEN83]; (b): [KANE84]; (c): [DUNN85]; (d): [REDD78]; (e): [CHEN 83]; (f–h): [DUNN85].
Source: *[DUNN85]. © 1985 IEEE.*
Note: $\lfloor x \rfloor$: Largest integer smaller than or equal to x; $\lceil x \rceil$: Smallest integer larger than or equal to x; ( ) : Same type of codes.

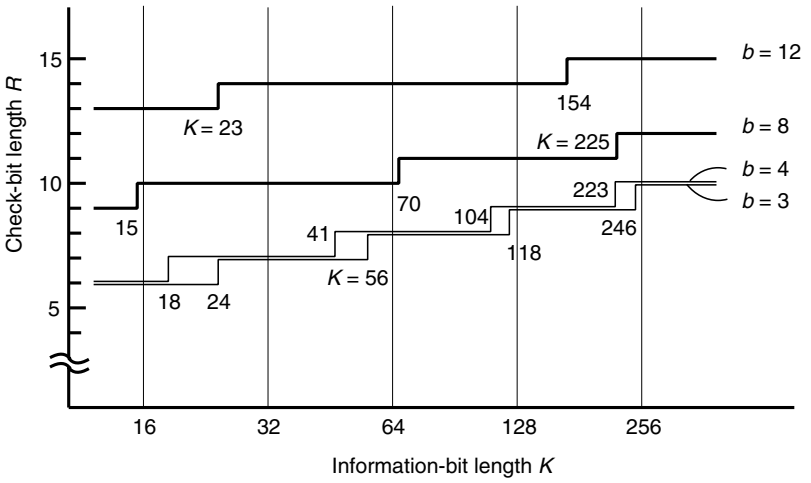


Figure 6.17 *Relationship between check-bit lengths and information-bit lengths of the SEC-DED-SbED codes.*

$\lambda = (2^4 - 1)/(2^2 - 1) = 5$. It can be easily understood that the subset satisfies the axioms of the field.

The following lemmas pertaining to the subfield are demonstrated without proof [FUJI91b].

Lemma 6.1 *Let A be a divisor of b, then* $\lambda = (2^b - 1)/(2^A - 1) > b$.

Lemma 6.2 *Every binary column vector of a* $b \times b$ *matrix element* $\boldsymbol{T}^{i\lambda}$ *in the subfield* $GF(2^A)$ *is different from that of the another* $b \times b$ *matrix element* $\boldsymbol{T}^{j\lambda}$ *in the same subfield, where i and j can take any value in the range of 1,* $2, \ldots, 2^A - 1$, *under* $i \neq j$.

Lemma 6.3 *No other elements than the identity element* $\boldsymbol{I}$ *in the subfield* $GF(2^A)$ *have weight one binary column vectors.*

6.2.2 Design for S*b*EC-DED Codes

Design by Using Elements of Subfield The elements from the subfield $GF(2^A)$ can be used to design the **H** matrix of the S*b*EC-DED code with two rows in the parity-check matrix [FUJI91a, 91b]. In terms of code length, A should be the largest divisor of b. This always holds for $b = 2A$ as the best case.

Theorem 6.12 [PATE80] *The code expressed by the following* $\boldsymbol{H}$ *matrix is an S*b*EC-DED code having maximum code length in bits* $N = b \cdot (2^A - 1)$ *and check-bit length* $R = 2b$:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{I} & \cdots & \boldsymbol{I} & \cdots & \boldsymbol{I} \\ \boldsymbol{T}^{\lambda} & \boldsymbol{T}^{2\lambda} & \boldsymbol{T}^{3\lambda} & \cdots & \boldsymbol{T}^{i\lambda} & \cdots & \boldsymbol{T}^{(2^A-1)\lambda} = \boldsymbol{I} \end{bmatrix},$$

where $\boldsymbol{T}^{\lambda}, \boldsymbol{T}^{2\lambda}, \ldots, \boldsymbol{T}^{(2^A-1)\lambda} = \boldsymbol{I}$, *are elements in* $GF(2^A)$ *subfield of* $GF(2^b)$.

Proof Let two single-bit errors be E_1 and E_2, located in the i-th byte and j-th byte, respectively, and single-byte error be E_3, located in the k-th byte. Suppose that the errors E_1 and E_2 are miscorrected to the error E_3. Then the following relations hold:

$$E_1 + E_2 = E_3,$$
$$E_1 \cdot \mathbf{T}^{i\lambda} + E_2 \cdot \mathbf{T}^{j\lambda} = E_3 \cdot \mathbf{T}^{k\lambda}.$$

From these, we finally obtain the following relation:

$$E_1 \cdot (\mathbf{T}^{i\lambda} + \mathbf{T}^{k\lambda})(\mathbf{T}^{j\lambda} + \mathbf{T}^{k\lambda})^{-1} = E_2.$$

Let $(\mathbf{T}^{i\lambda} + \mathbf{T}^{k\lambda})(\mathbf{T}^{j\lambda} + \mathbf{T}^{k\lambda})^{-1}$ be $\mathbf{T}^{l\lambda}$. It is apparent that $\mathbf{T}^{l\lambda} \neq \mathbf{I}$. Since, by Lemma 6.3, $\mathbf{T}^{l\lambda}$ does not have weight-one columns, we have $E_1 \cdot \mathbf{T}^{l\lambda} \neq E_2$ for any weight-one vectors E_1 and E_2. This is a contradiction. So the code in this theorem is an SbEC-DED code.

Q.E.D.

The SbED-DED code can be extended by adding a column vector as shown in the next theorem.

Theorem 6.13 *The SbEC-DED code shown in Theorem 6.12 is extended by adding one column vector* $\begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{0} \end{bmatrix}$ *as follows*:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{I} & \boldsymbol{I} & \cdots & \boldsymbol{I} & \cdots & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{T}^{\lambda} & \boldsymbol{T}^{2\lambda} & \boldsymbol{T}^{3\lambda} & \cdots & \boldsymbol{T}^{i\lambda} & \cdots & \boldsymbol{T}^{(2^A-1)\lambda} = \boldsymbol{I} \end{bmatrix}.$$

The maximum code length in bits is $N = b \cdot 2^A$, *and check-bit length is* $R = 2b$.

The preceding codes are problematic in that the code lengths are determined only by byte length b.

Definition 6.4 The $s \times b$ *slimmed matrix*, where $s \geq b$, is a matrix whose $s - b$ columns are deleted from the original $s \times s$ matrix $\mathbf{T}_s^{i\lambda}$ and is written as $|\mathbf{T}_s^{i\lambda}|_b$. □

Theorem 6.14 *Let* $\boldsymbol{T}_s$ *be a primitive element of* $GF(2^s)$, *where* $s = R - b$ *and* $R \geq 2b$, *and let the set of* 2^A *binary expressed* $s \times s$ *elements, namely* $\boldsymbol{0}$, $\boldsymbol{T}_s^{\lambda}$, $\boldsymbol{T}_s^{2\lambda}$, …, $\boldsymbol{T}_s^{i\lambda}$, …, $\boldsymbol{T}_s^{(p-1)\lambda} = \boldsymbol{I}_s$, *be a* $GF(p = 2^A)$ *subfield of* $GF(2^s)$. *Also let* $|\boldsymbol{T}_s^{i\lambda}|_b$ *be an* $s \times b$ *slimmed element of* $\boldsymbol{T}_s^{i\lambda}$, *where* $i = 1, 2, \cdots, 2^A - 1$, *and let* $\boldsymbol{I}_b$ *be an identity element of* $GF(2^b)$. *Then the following* $R \times N$ $\boldsymbol{H}$ *matrix shows the SbEC-DED code with code parameters of* $N = b \cdot 2^A$ *(bits) for any* R *and* b *in bits*:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I}_b & \boldsymbol{I}_b & \cdots & \boldsymbol{I}_b & \cdots & \boldsymbol{I}_b \\ |\boldsymbol{0}_s|_b & |\boldsymbol{T}_s^{\lambda}|_b & \cdots & |\boldsymbol{T}_s^{i\lambda}|_b & \cdots & |\boldsymbol{T}_s^{(2^A-1)\lambda}|_b = |\boldsymbol{I}_s|_b \end{bmatrix} \begin{matrix} \uparrow \\ b \\ \downarrow \\ \uparrow \\ s \\ \downarrow \end{matrix}.$$

This theorem can be easily proved, and therefore the proof is omitted.

Based on the **H** matrix of the Theorem 6.14 code with two rows, a more general type of S*b*EC-DED code having multiple rows in **H** can be designed as described in the next theorem.

Theorem 6.15 *The following **H** matrix having a linearly independent pair of columns shows the SbEC-DED code with multiple rows*:

$$\boldsymbol{H} = \left[\begin{array}{c} \boldsymbol{I}_b \quad \boldsymbol{I}_b \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \cdots \quad \boldsymbol{I}_b \\ [\textit{elements from } GF(2^{A_1}) \textit{ subfield of } GF(2^{s_1})] \\ [\textit{elements from } GF(2^{A_2}) \textit{ subfield of } GF(2^{s_2})] \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ [\textit{elements from } GF(2^{A_m}) \textit{ subfield of } GF(2^{s_m})] \end{array}\right] \begin{array}{c} \uparrow \\ b \\ \downarrow \\ \uparrow \\ s_1 \\ \downarrow \\ \uparrow \\ s_2 \\ \downarrow \\ \vdots \\ \uparrow \\ s_m \\ \downarrow \end{array},$$

where $s = R - b = s_1 + s_2 + \ldots + s_m$ for $s_j \geq b$, $j = 1, 2, \ldots, m$, $\boldsymbol{I}_b$, is an identity element of $GF(2^b)$, and each element of $GF(2^{A_j})$ subfield of $GF(2^{s_j})$, $j = 1, 2, \ldots, m$, is a binary expressed $s_j \times b$ slimmed matrix. The maximum code length in bits is $N = b \cdot 2^{A_1 + A_2 + \cdots + A_m}$.

This theorem can be easily proved, and therefore the proof is omitted.

Example 6.9 (256, 240) S4EC-DED code

Let $b = 4$ and $R = 16$, then $R - b = 8 + 4$. The elements of $GF(2^4)$ subfield of $GF(2^8)$ and those of $GF(2^2)$ subfield of $GF(2^4)$ are used in the second and the third rows of the **H** matrix, respectively:

$GF(2^4)$ subfield of $GF(2^8) = \{\mathbf{0}_8,\ \mathbf{T}_8^{17},\ \mathbf{T}_8^{34},\ \mathbf{T}_8^{51},\ \ldots,\ \mathbf{T}_8^{255} = \mathbf{I}_8\}$,
$GF(2^2)$ subfield of $GF(2^4) = \{\mathbf{0}_4,\ \mathbf{T}_4^{5},\ \mathbf{T}_4^{10},\ \mathbf{T}_4^{15} = \mathbf{I}_4\}$,

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_4 & \mathbf{I}_4 & \mathbf{I}_4 & \cdots & \mathbf{I}_4 & \cdots & \mathbf{I}_4 \\ |\mathbf{0}_8|_4 & |\mathbf{T}_8^{17}|_4 & |\mathbf{T}_8^{34}|_4 & \cdots & |\mathbf{T}_8^{17i}|_4 & \cdots & |\mathbf{T}_8^{255}|_4 = |\mathbf{I}_8|_4 \\ \mathbf{0}_4 & \mathbf{0}_4 & \mathbf{0}_4 & \cdots & \mathbf{T}_4^{5j} & \cdots & \mathbf{T}_4^{15} \end{bmatrix}.$$

Design by Using Elements of Multiplicative Coset The finite field $GF(2^b)$ can be factored into $\lambda = (2^b - 1)/(2^A - 1)$ multiplicative cosets by the subfield $GF(2^A)$.

Example 6.10

For $b = 4$ and $A = 2$, the subfield $GF(2^2)$ can be used to factor $GF(2^4)$ into $\lambda = 5$ multiplicative cosets as follows:

$$\begin{aligned}
\mathbf{Q}_0 &= \{\mathbf{0},\ \mathbf{T^5},\ \mathbf{T^{10}},\ \mathbf{T^{15}} = \mathbf{I}\} = GF(2^2) \text{ subfield of } GF(2^4)\\
\mathbf{Q}_1 &= \{\mathbf{0},\ \mathbf{T^6},\ \mathbf{T^{11}},\ \mathbf{T}\} = \mathbf{T}\ \cdot \mathbf{Q}_0\\
\mathbf{Q}_2 &= \{\mathbf{0},\ \mathbf{T^7},\ \mathbf{T^{12}},\ \mathbf{T^2}\} = \mathbf{T}^2 \cdot \mathbf{Q}_0\\
\mathbf{Q}_3 &= \{\mathbf{0},\ \mathbf{T^8},\ \mathbf{T^{13}},\ \mathbf{T^3}\} = \mathbf{T}^3 \cdot \mathbf{Q}_0\\
\mathbf{Q}_4 &= \{\mathbf{0},\ \mathbf{T^9},\ \mathbf{T^{14}},\ \mathbf{T^4}\} = \mathbf{T}^4 \cdot \mathbf{Q}_0.
\end{aligned}$$

Lemma 6.4 *Every element in a multiplicative coset of the $GF(2^A)$ subfield of $GF(2^b)$ is closed under addition in $GF(2^b)$.*

This lemma can be easily proved, and therefore the proof is omitted. The lemma leads to the next theorem [FUJI91a, 91b].

Theorem 6.16 *With elements in a multiplicative coset of the $GF(2^A)$ subfield of $GF(2^b)$, namely $\boldsymbol{Q}_d = \{\boldsymbol{0},\ \boldsymbol{T}^{\lambda+d},\ \boldsymbol{T}^{2\lambda+d},\ \ldots,\ \boldsymbol{T}^{(2^A-1)\lambda+d}\}$, where $d = 0, 1, \ldots, \lambda - 1$, the following $\boldsymbol{H}$ matrix shows the SbEC-DED code with maximum code length in bits $N = b{\cdot}2^A$:*

$$\boldsymbol{H} = \begin{bmatrix}
\boldsymbol{I}_b & \boldsymbol{I}_b & \boldsymbol{I}_b & \cdots & \boldsymbol{I}_b & \cdots & \boldsymbol{I}_b \\
|\boldsymbol{0}_{R-b}|_b & \left|\boldsymbol{T}^{\lambda+d}_{R-b}\right|_b & \left|\boldsymbol{T}^{2\lambda+d}_{R-b}\right|_b & \cdots & \left|\boldsymbol{T}^{i\lambda+d}_{R-b}\right|_b & \cdots & \left|\boldsymbol{T}^{(2^A-1)\lambda+d}_{R-b}\right|_b
\end{bmatrix} \begin{matrix} \updownarrow b \\ \updownarrow R-b \end{matrix},$$

where $|\boldsymbol{0}_{R-b}|_b$ and $|\boldsymbol{T}^{i\lambda+d}_{R-b}|_b$ are $(R-b) \times b$ binary slimmed elements of the original ones in $\boldsymbol{Q}_d$, and $\boldsymbol{I}_b$ is an identity element in $GF(2^b)$.

It is apparent that the SbEC-DED code having multiple rows in **H** can also be designed by using the elements of a multiplicative coset as shown in Theorem 6.15.

Design by Using Elements of Additive Coset The finite field $GF(2^b)$ can be factored into 2^{b-A} additive cosets by the subfield $GF(2^A)$. Every element of the $GF(2^b)$ field is in one and only one coset of a subfield $GF(2^A)$.

Example 6.11

For $b = 4$ and $A = 2$, there exist four additive cosets of the $GF(2^2)$ subfield of $GF(2^4)$ as shown below. In this case the companion matrix **T** is determined by the primitive polynomial $\mathbf{g}(x) = x^4 + x + 1$:

$$\begin{aligned}
\mathbf{P}_0 &= \{\mathbf{0},\ \mathbf{T}^5,\ \mathbf{T}^{10},\ \mathbf{T}^{15} = \mathbf{I}\} = GF(2^2) \text{ subfield of } GF(2^4)\\
\mathbf{P}_1 &= \{\mathbf{T},\ \mathbf{T}^2,\ \mathbf{T}^4,\ \mathbf{T}^8\} = \mathbf{T}\ + \mathbf{P}_0\\
\mathbf{P}_2 &= \{\mathbf{T}^3,\ \mathbf{T}^{11},\ \mathbf{T}^{12},\ \mathbf{T}^{14}\} = \mathbf{T}^3 + \mathbf{P}_0\\
\mathbf{P}_3 &= \{\mathbf{T}^6,\ \mathbf{T}^7,\ \mathbf{T}^9,\ \mathbf{T}^{13}\} = \mathbf{T}^6 + \mathbf{P}_0.
\end{aligned}$$

Lemma 6.5 *The addition with two elements in an additive coset results in an element of the subfield $GF(2^A)$.*

This lemma can be easily proved by using the property of the additive coset. So we have the following S*b*EC-DED code [HAMA91, FUJI93].

Theorem 6.17 *For using elements in an additive coset of the $GF(2^A)$ subfield of $GF(2^b)$, the following **H** matrix shows the SbEC-DED code with maximum code length in bits $N = b \cdot 2^A$:*

$$\boldsymbol{H}_2 = \begin{bmatrix} \boldsymbol{I}_b & \boldsymbol{I}_b & \boldsymbol{I}_b & \cdots & \boldsymbol{I}_b & \cdots & \boldsymbol{I}_b \\ \boldsymbol{T}^{p_0} & \boldsymbol{T}^{p_1} & \boldsymbol{T}^{p_2} & \cdots & \boldsymbol{T}^{p_i} & \cdots & \boldsymbol{T}^{p_{2^A-1}} \end{bmatrix},$$

where $\boldsymbol{I}_b$ is an identity element in $GF(2^b)$ and $\boldsymbol{T}^{p_i}$'s, $i = 0, 1, 2, \cdots, 2^A - 1$, are elements in an additive coset.

From this theorem, it is apparent that the **H** matrix with multiple rows that satisfies linear independence between any pair of columns gives the S*b*EC-DED code with using elements in an additive coset. That is, the **H** matrix has the following structure: all **I** elements are included in the first row and the remaining part of **H** with $r - 1$ rows has different column vectors from the elements of an additive coset, as shown below:

$$\mathbf{H}_r = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} & \cdots & \mathbf{I} & \cdots & \mathbf{I} \\ \cdot & \cdot & \cdots & \mathbf{T}^{i_1} & \cdots & \mathbf{T}^{j_1} & \cdots & \cdot \\ \cdot & \cdot & \cdots & \mathbf{T}^{i_2} & \cdots & \mathbf{T}^{j_2} & \cdots & \cdot \\ & & & \vdots & & \vdots & & \\ \cdot & \cdot & \cdots & \mathbf{T}^{i_{r-1}} & \cdots & \mathbf{T}^{j_{r-1}} & \cdots & \cdot \end{bmatrix}. \tag{6.9}$$

In this case, $\mathbf{T}^{i_1}$, $\mathbf{T}^{i_2}$, ..., $\mathbf{T}^{i_{r-1}}$, $\mathbf{T}^{j_1}$, $\mathbf{T}^{j_2}$, ... , $\mathbf{T}^{j_{r-1}}$, are elements included in the additive coset of $GF(2^A)$ and **I** is an identity element in $GF(2^b)$. The maximum number of column vectors, meaning the maximum code length n (bytes), can be obtained as

$$n = (2^A)^{r-1} = 2^{A(r-1)}. \tag{6.10}$$

By using the $\mathbf{H}_r$ with r rows above defined, we obtain the new S*b*EC-DED code with an extended code length in the next theorem.

Theorem 6.18 *The following **H** matrix shows the SbEC-DED code with a maximum code length in bits $N = b \cdot (2^{A(r-1)} + 2^{A(r-t-1)})$, where $1 \leq t \leq r - 2$, elements in **H***

are included in an additive coset of $GF(2^A)$ except $GF(2^A)$ itself, and ***0*** *and* ***I*** *are the zero element and the identity element in $GF(2^b)$, respectively*:

$$
\boldsymbol{H} = \left[\begin{array}{c|cccc}
 & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\
 & \vdots & \vdots & \cdots & \vdots \\
 & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\
\cline{2-5}
\boldsymbol{H}_r & & & & \\
 & & \boldsymbol{H}_{r-t} & & \\
 & & & &
\end{array}\right]
\begin{array}{c}
\uparrow \\ t \\ \downarrow \\ \uparrow \\ r-t \\ \downarrow
\end{array}
$$

$$
\begin{array}{ccccccc|cccccc}
\cdots & \cdots & i & \cdots & j & \cdots & \cdots & \cdots & \cdots & \cdots & k & \cdots & \cdots
\end{array}
$$

$$
= \left[\begin{array}{ccccccc|cccccc}
\boldsymbol{I} & \cdots & \boldsymbol{I} & \cdots & \boldsymbol{I} & \cdots & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \cdots & \boldsymbol{0} \\
\cdots & \cdots & \boldsymbol{T}^{i_1} & \cdots & \boldsymbol{T}^{j_1} & \cdots & \cdots & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \cdots & \boldsymbol{0} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\cdots & \cdots & \boldsymbol{T}^{i_{t-1}} & \cdots & \boldsymbol{T}^{j_{t-1}} & \cdots & \cdots & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} & \cdots & \boldsymbol{0} \\
\cline{8-13}
\cdots & \cdots & \boldsymbol{T}^{i_t} & \cdots & \boldsymbol{T}^{j_t} & \cdots & \cdots & \boldsymbol{I} & \boldsymbol{I} & \cdots & \boldsymbol{I} & \cdots & \boldsymbol{I} \\
\cdots & \cdots & \boldsymbol{T}^{i_{t+1}} & \cdots & \boldsymbol{T}^{j_{t+1}} & \cdots & \cdots & \cdots & \cdots & \cdots & \boldsymbol{T}^{k_{t+1}} & \cdots & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\cdots & \cdots & \boldsymbol{T}^{i_{r-1}} & \cdots & \boldsymbol{T}^{j_{r-1}} & \cdots & \cdots & \cdots & \cdots & \cdots & \boldsymbol{T}^{k_{r-1}} & \cdots & \cdots
\end{array}\right]
\begin{array}{c}
\uparrow \\ t \\ \downarrow \\ \uparrow \\ r-t \\ \downarrow
\end{array} .
$$

$$
\longleftarrow X \longrightarrow \longleftarrow Y \longrightarrow
$$

Proof Because every pair of columns in **H** is linearly independent, the **H** matrix satisfies the S*b*EC function. Next we consider the case of three types of errors ocurring in the word. Let the two single-bit errors be E_1 and E_2, and the single-byte error be E_3. Then there are five cases of errors depending on the locations of the errors.

1. All errors are located in one part of the word, corresponding to the region including $\mathbf{H}_r$ in **H**, that is, region X indicated in **H**, or the region including $\mathbf{H}_{r-t}$ in **H**, that is, the region Y in the **H** matrix not located in the span over the two regions.
2. Errors E_1 (or E_2) and E_3 are located in the different bytes of region X, and an error E_2 (or E_1) is located in region Y.
3. Error E_1 (or E_2) is located in region X, and errors E_2 (or E_1) and E_3 are located in the different bytes of region Y.
4. Error E_3 is located in region X, and errors E_1 and E_2 are located in the different bytes of region Y.
5. Errors E_1 and E_2 are located in the different bytes of region X, and error E_3 is located in region Y.

In case 1, it is apparent that the indicated **H** matrix satisfies the S*b*EC-DED function of Theorem 6.17 and the matrix shown in Eq. (6.9).

In cases 2 and 3, the errors E_1 and E_2 cannot be miscorrected to error E_3 depending on the structure of the matrix **H**.

In case 4, the error E_3 cannot be miscorrected to errors E_1 and E_2 for the nonzero error patterns of E_1, E_2, and E_3.

As for case 5, let errors E_1 and E_2 be located in the i-th byte and the j-th byte of the region X, respectively, and also let error E_3 be located in the k-th byte of region Y. Suppose that the errors E_1 and E_2 are miscorrected to error E_3. Then the following relations hold:

$$E_1 + E_2 = 0, \tag{6.11}$$

$$E_1 \cdot \mathbf{T}^{i_t} + E_2 \cdot \mathbf{T}^{j_t} = E_3, \tag{6.12}$$

$$E_1 \cdot \mathbf{T}^{i_{t+1}} + E_2 \cdot \mathbf{T}^{j_{t+1}} = E_3 \cdot \mathbf{T}^{k_{t+1}}. \tag{6.13}$$

Given Eq. (6.11), the Eqs. (6.12), and (6.13) are transformed into

$$E_1 \cdot (\mathbf{T}^{i_t} + \mathbf{T}^{j_t}) = E_3, \tag{6.14}$$

$$E_1 \cdot (\mathbf{T}^{i_{t+1}} + \mathbf{T}^{j_{t+1}}) = E_3 \cdot \mathbf{T}^{k_{t+1}}. \tag{6.15}$$

From Eqs. (6.14) and (6.15), we finally obtain the relation

$$(\mathbf{T}^{i_{t+1}} + \mathbf{T}^{j_{t+1}}) \cdot (\mathbf{T}^{i_t} + \mathbf{T}^{j_t})^{-1} = \mathbf{T}^{k_{t+1}}. \tag{6.16}$$

By Lemma 6.5, the result of the left-hand side in Eq. (6.16) is included in the subfield $GF(2^A)$, while the element of the right-hand side in Eq. (6.16) is included in the additive coset. This presents a contradiction. Hence the **H** matrix in the theorem shows the S*b*EC-DED code for this error case. The **H** matrix also satisfies S*b*EC-DED function for all error cases. The maximum code length in bits can be expressed easily from Eq. (6.10). Q.E.D.

Theorem 6.19 *The following* $\boldsymbol{H}$ *matrix shows the SbEC-DED code with code parameters of* $N = b \cdot 2^A \cdot (2^{A(r-1)} - 1)/(2^A - 1)$ *and* $R = r \cdot b$ *in bits for* $r \geq 2$:

$$\boldsymbol{H} = \left[\begin{array}{c|c|c|c|c} \boldsymbol{H}_r & \begin{array}{ccc} \boldsymbol{0} & \cdots & \boldsymbol{0} \end{array} & \begin{array}{ccc} \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \cdots & \boldsymbol{0} \end{array} & \cdots & \begin{array}{ccc} \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \cdots & \boldsymbol{0} \\ & \vdots & \\ \boldsymbol{0} & \cdots & \boldsymbol{0} \end{array} \\ & \boldsymbol{H}_{r-1} & \boldsymbol{H}_{r-2} & & \boldsymbol{H}_2 \end{array}\right],$$

where $\boldsymbol{H}_r, \boldsymbol{H}_{r-1}, \boldsymbol{H}_{r-2}, \ldots, \boldsymbol{H}_2$, *are the matrices defined in Theorems 6.17 and 6.18 and in the matrix shown in Eq. (6.9).*

Proof In addition to the error cases mentioned in Theorem 6.18, it is necessary to consider an error case where E_1, E_2, and E_3 errors, defined in the proof of Theorem 6.18, are located in different three regions. In this case it is apparent from the structure of the **H** matrix that the error E_3 cannot be miscorrected to the E_1 and E_2 errors.

From the above and the previous Theorems 6.17 and 6.18, the maximum code length in bits can be expressed as

$$\begin{aligned} N &= b \cdot \{(2^A)^{r-1} + (2^A)^{r-2} + \ldots + (2^A)^2 + 2^A\} \\ &= b \cdot 2^A \cdot \{(2^A)^{r-2} + (2^A)^{r-3} + \ldots + 2^A + 1\} \\ &= b \cdot 2^A \cdot \left\{ \sum_{i=0}^{r-2} 2^{Ai} \right\} \\ &= b \cdot 2^A \cdot \frac{2^{A(r-1)} - 1}{2^A - 1}. \end{aligned}$$

Q.E.D.

The code shown in Theorem 6.19 has better code length than the codes using elements of subfield or multiplicative coset.

Figure 6.18 shows the relation between the information-bit length K and the check-bit length R of the code shown in Theorem 6.19 for byte lengths $b = 4$, 6, and 8 bits. From this, the code shown in Theorem 6.19 is very practical from the point that the S4EC-DED code with information-bit length $K = 64$ is obtained with the same check-bit length $R = 12$ as that of the Hamming S4EC code with $K = 64$. On the other hand, the codes using elements of subfield or multiplicative coset require 13 check bits for $K = 64$.

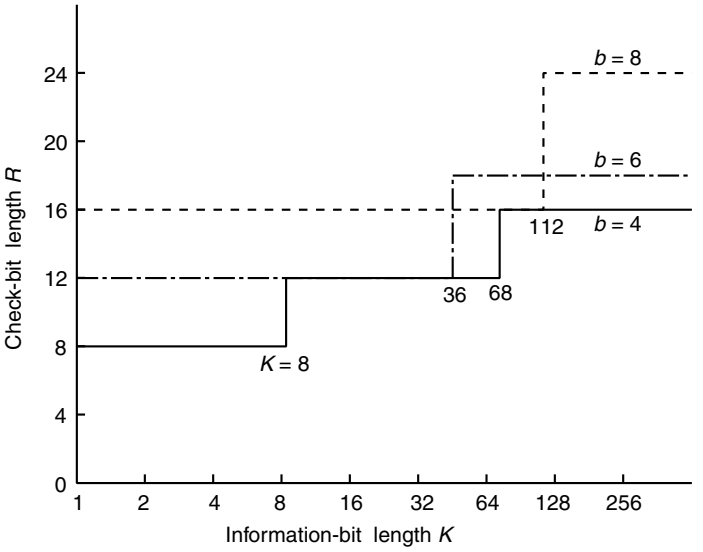


Figure 6.18 *Relationship between check-bit lengths and information-bit lengths of the SbEC-DED codes.* Source: *[FUJI93].* © *1993 IEICE Japan.*

The **H** matrix shown below is the (80, 68) S4EC-DED code using elements of an additive coset, such as the elements of $\mathbf{P}_1$ shown in Example 6.11, and also using the identity element **I** and zero element **0** in $GF(2^4)$.

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^2 & \mathbf{T}^4 & \mathbf{T}^4 & \mathbf{T}^4 & \mathbf{T}^4 & \mathbf{T}^8 & \mathbf{T}^8 & \mathbf{T}^8 & \mathbf{T}^8 & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^4 & \mathbf{T}^8 & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^4 & \mathbf{T}^8 & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^4 & \mathbf{T}^8 & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^4 & \mathbf{T}^8 & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^4 & \mathbf{T}^8 \end{bmatrix}. \tag{6.17}$$

The following matrix, shown below, is the (76, 64) S4EC-DED code deleted by one column having maximum weight from the **H** shown in Eq. (6.17), and it is transformed into, echelon canonical form. Computer simulation says that the code shown in Eq. (6.18) has very high error detection capabilities of random double-byte errors and random triple-bit errors, that is, 91.93% and 92.03%, respectively.

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^4 & \mathbf{T}^8 & \mathbf{T}^3 & \mathbf{T}^{11} & \mathbf{T}^{12} & \mathbf{T}^{14} & \mathbf{T}^6 & \mathbf{T}^7 & \mathbf{T}^9 & \mathbf{T}^{13} & \mathbf{I} & \mathbf{T}^5 & \mathbf{T}^5 & \mathbf{T}^{10} & \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{T}^4 & \mathbf{T}^8 & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^{12} & \mathbf{T}^{14} & \mathbf{T}^3 & \mathbf{T}^{11} & \mathbf{T}^9 & \mathbf{T}^{13} & \mathbf{T}^6 & \mathbf{T}^7 & \mathbf{I} & \mathbf{T}^5 & \mathbf{T}^{10} & \mathbf{T}^5 & \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{6.18}$$

Single-byte error correction procedure of the SbEC-DED code is same as that of the SbEC code. Double-bit error detection of the code requires the following logic operation:

$$(\text{Nonzero syndrome}) \bigwedge (\text{No byte error pointers existed}), \tag{6.19}$$

where $\bigwedge$ shows AND operation. That is, the code can detect double-bit errors when there exist no byte error pointers for nonzero syndromes. It can be easily understood that the logic operation of Eq. (6.19) can detect other multiple errors than double-bit errors, such as double-byte errors and triple-bit errors, within the range of error detection capability of the code. The hardware amount of this error detection logic is very small, around 10 to 20 gates.

Design by Using Elements of Subset of GF(2^b) The former design methods are based on using the cosets of a subfield of $GF(2^b)$. However, the code length depends on the size of the cosets of the subfield of $GF(2^b)$, and therefore they are inefficient when b is a prime number, since there exist no nontrivial subfield. To overcome this disadvantage, a more general construction method [XIAO96] using subsets of $GF(2^b)$ is presented here.

Definition 6.5 Suppose that **T** is a companion matrix corresponding to the primitive polynomial $\mathbf{g}(x)$ over $GF(2^b)$. We call $\Psi = \{\mathbf{T}^b,\ \mathbf{T}^{b+1},\ \ldots,\ \mathbf{T}^{2^b-b-1}\}$ the *weight-2 set* of $GF(2^b)$. □

Lemma 6.6 *The weight of any binary column vector of* $\boldsymbol{T}^i \in \Psi$ *is greater than or equal to 2.*

Proof Let α be a primitive root of $\mathbf{g}(x)$ over $GF(2^b)$. Then

$$\mathbf{T}^j = \begin{bmatrix} \alpha^j & \alpha^{j+1} & \cdots & \alpha^{j+b-1} \end{bmatrix} \qquad \text{for } j = 0, 1, 2, \ldots, 2^b - 2,$$

where α^i's weight is at least 2 for $b \le i \le 2^b - 2$. Observe that if $b \le j \le 2^b - b - 1$, and α^i is a column of $\mathbf{T}^j$, then $b \le i \le 2^b - 2$. Therefore α^i's weight is at least 2. Q.E.D.

Definition 6.6 Let $\mathbf{G} = \{\mathbf{T}^{i_1}, \mathbf{T}^{i_2}, \ldots, \mathbf{T}^{i_k}\}$ be a subset of $GF(2^b)$. We call $\mathbf{G}$ a *generating set* in $GF(2^b)$ if $(\mathbf{T}^{i_j} + \mathbf{T}^{i_n})(\mathbf{T}^{i_m} + \mathbf{T}^{i_n})^{-1} \in \Psi$ for all distinct $\mathbf{T}^{i_j}$, $\mathbf{T}^{i_m}$, and $\mathbf{T}^{i_n}$ in $\mathbf{G}$. □

Clearly, any subset of a generating set in $GF(2^b)$ is a generating set.

Theorem 6.20 *A subfield $GF(2^A)$ or an additive coset of the subfield $GF(2^A)$ of $GF(2^b)$ is a generating set. Also a multiplicative coset of a subfield $GF(2^A)$ of $GF(2^b)$ plus zero is a generating set.*

Theorem 6.21 *If $\Phi = \{\boldsymbol{T}^{i_1}, \boldsymbol{T}^{i_2}, \ldots, \boldsymbol{T}^{i_v}\}$ is a generating set in $GF(2^b)$, then*

$$\Phi + \{\boldsymbol{T}^m\} = \{\boldsymbol{T}^{i_1} + \boldsymbol{T}^m, \ldots, \boldsymbol{T}^{i_v} + \boldsymbol{T}^m\} \quad \textit{and}$$
$$\Phi \cdot \{\boldsymbol{T}^m\} = \{\boldsymbol{T}^{i_1} \cdot \boldsymbol{T}^m, \ldots, \boldsymbol{T}^{i_v} \cdot \boldsymbol{T}^m\}$$

are also generating sets in $GF(2^b)$, for all $\boldsymbol{T}^m \in GF(2^b)$.

Essentially Theorem 6.20 tells us that the generating set does exist and Theorem 6.21 tells us that varieties of generating sets can be found when an existing one is provided.

Example 6.12

Consider the field $GF(2^4) = \{\mathbf{0}, \mathbf{I}, \mathbf{T}, \mathbf{T}^2, \ldots, \mathbf{T}^{14}\}$. Clearly, $GF(2^2) = \{\mathbf{0}, \mathbf{I}, \mathbf{T}^5, \mathbf{T}^{10}\}$ is the nontrivial subfield of $GF(2^4)$. According to Theorem 6.20, $\Phi = GF(2^2)$ is a generating set in $GF(2^4)$. Moreover more generating sets can be obtained by Theorem 6.21. For example, $\Phi_1 = \Phi + \{\mathbf{T}\} = \{\mathbf{T}, \mathbf{T}^2, \mathbf{T}^4, \mathbf{T}^8\}$ and $\Phi_2 = \Phi \cdot \{\mathbf{T}^4\} = \{\mathbf{0}, \mathbf{T}^4, \mathbf{T}^9, \mathbf{T}^{14}\}$.

Because the generating set plays the key role in the design of the SbEC-DED code, finding the generating set is important. Here we introduce a simple computer search algorithm [XIAO96] for the generating set with given size in $GF(q)$.

Step 1. *Generate the finite field $GF(q) = \{\boldsymbol{0}, \boldsymbol{I}, \boldsymbol{T}, \boldsymbol{T}^2, \ldots, \boldsymbol{T}^{q-2}\}$.*

Step 2. *Input a positive integer v, the size of the desired generating set.*

Step 3. *Let $\Phi = \{\Phi_1, \Phi_2, \ldots, \Phi_n\}$ be the family of all subsets each having v elements in $GF(q)$.*

Step 4. *For i = 1 to n do*

If $(\boldsymbol{T}^{i_j} + \boldsymbol{T}^{i_n})(\boldsymbol{T}^{i_m} + \boldsymbol{T}^{i_n})^{-1} \in \Psi$, for all distinct $\boldsymbol{T}^{i_j}$, $\boldsymbol{T}^{i_n}$, and $\boldsymbol{T}^{i_m}$ in Φ_i, then output Φ_i and stop.

Example 6.13 [XIAO96]

By the algorithm above, we can find the following generating sets:

$$\Phi_1 = \{\mathbf{T}^{26}, \mathbf{T}^{27}, \mathbf{T}^{29}, \mathbf{T}^{30}\}, \quad \Phi_2 = \{\mathbf{T}^4, \mathbf{T}^{27}, \mathbf{T}^{29}, \mathbf{T}^{30}\},$$
$$\Phi_3 = \{\mathbf{T}, \mathbf{T}^{25}, \mathbf{T}^{29}, \mathbf{T}^{30}\}$$

in $GF(2^5)$, and

$$\Phi_4 = \{\mathbf{0},\ \mathbf{T}^{73},\ \mathbf{T}^{123},\ \mathbf{T}^{136},\ \mathbf{T}^{177},\ \mathbf{T}^{233},\ \mathbf{T}^{269},\ \mathbf{T}^{284},\ \mathbf{T}^{425},\ \mathbf{T}^{480},\ \mathbf{T}^{494}\} \quad \text{and}$$
$$\Phi_5 = \{\mathbf{T}^{2},\ \mathbf{T}^{15},\ \mathbf{T}^{48},\ \mathbf{T}^{76},\ \mathbf{T}^{143},\ \mathbf{T}^{253},\ \mathbf{T}^{270},\ \mathbf{T}^{281},\ \mathbf{T}^{342},\ \mathbf{T}^{372},\ \mathbf{T}^{420}\}$$

in $GF(2^9)$.

Using the elements of the generating set of $GF(2^b)$, the SbEC-DED code with two rows in its $\mathbf{H}$ matrix can be designed as below.

Theorem 6.22 *Let $\Phi = \{\boldsymbol{T}^{i_1}, \boldsymbol{T}^{i_2}, \cdots, \boldsymbol{T}^{i_v}\}$ be a generating set in $GF(2^b)$. The code expressed by the following $\boldsymbol{H}$ matrix is an SbEC-DED code having code length in bits $N = v \cdot b$ and check-bit length $R = 2b$:*

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I}_b & \boldsymbol{I}_b & \cdots & \boldsymbol{I}_b \\ \boldsymbol{T}^{i_1} & \boldsymbol{T}^{i_2} & \cdots & \boldsymbol{T}^{i_v} \end{bmatrix}.$$

Proof Every two column vectors of $\mathbf{H}$ are linearly independent and therefore the code is of distance three or higher. Now let E_1 and E_2 be the error patterns with weight one occurred at the m-th and the j-th bytes, respectively, and E_3 be the byte error occurred at the n-th byte. A miscorrection of two single-bit errors as a single-byte error implies that

$$E_3 = E_1 + E_2,$$
$$E_3 \cdot \mathbf{T}^{i_n} = E_1 \cdot \mathbf{T}^{i_m} + E_2 \cdot \mathbf{T}^{i_j}.$$

From the two equations above we have

$$(\mathbf{T}^{i_j} + \mathbf{T}^{i_n})(\mathbf{T}^{i_m} + \mathbf{T}^{i_n})^{-1} \cdot E_1 = E_2.$$

Let $\mathbf{T}^p = (\mathbf{T}^{i_j} + \mathbf{T}^{i_n})(\mathbf{T}^{i_m} + \mathbf{T}^{i_n})^{-1}$. Since $\mathbf{T}^{i_j}, \mathbf{T}^{i_n}, \mathbf{T}^{i_m} \in \Phi$, and Φ is a generating set in $GF(2^b)$, we have $\mathbf{T}^p \in \Psi$. By Lemma 6.6, we know that the weight of any column vector of $\mathbf{T}^p$ is 2 or higher. Thus $\mathbf{T}^p \cdot E_1 \neq E_2$ for any weight-one vectors E_1 and E_2. This proves that the code in this theorem is an SbEC-DED code. Q.E.D.

The code design presented here has the following advantages compared to the previous ones.

1. If we choose the subfields, additive cosets, and multiplicative cosets plus zero used in Theorems 6.12 through 6.17 as the generating sets, all the codes described in these six theorems can be obtained.
2. If b is prime, there is no nontrivial subfield of $GF(2^b)$. However, for the prime number of b, we can find some generating sets in $GF(2^b)$ as shown in Example 6.13.
3. For an odd number of b, we can find generating sets in $GF(2^b)$ with more elements than those of any proper subfield of $GF(2^b)$. For example, Φ_4 and Φ_5 in $GF(2^9)$ as shown in Example 6.13. In this case we can obtain better SbEC-DED codes than the previous ones.

Let Φ be a generating set with size v including the identity element $\mathbf{I}$, and let $\Phi^* = \Phi - \{\mathbf{I}\}$. In the following $\mathbf{H}$ matrix defined on Φ, the elements are chosen in such a way that any two columns are linearly independent:

$$\overline{\mathbf{H}}_r = \begin{bmatrix} \mathbf{I} & \cdots & \mathbf{I} & \cdots & \mathbf{I} \\ \cdots & \mathbf{T}^{i_1} & \cdots & \mathbf{T}^{j_1} & \cdots \\ \cdots & \mathbf{T}^{i_2} & \cdots & \mathbf{T}^{j_2} & \cdots \\ \cdots & \mathbf{T}^{i_{r-1}} & \cdots & \mathbf{T}^{j_{r-1}} & \cdots \end{bmatrix} \begin{matrix} \uparrow \\ \\ r \\ \\ \downarrow \end{matrix}, \tag{6.20}$$

where $\mathbf{T}^{i_1}, \mathbf{T}^{j_1} \in \Phi^*$ and $\mathbf{T}^{i_2}, \cdots, \mathbf{T}^{i_{r-1}}, \mathbf{T}^{j_2}, \cdots,$ and $\mathbf{T}^{j_{r-1}} \in \Phi$.

It can be easily proved that the matrix $\overline{\mathbf{H}}_r$ shows an SbEC-DED code with a code length of $(v-1)v^{r-2}$ bytes. In general, we can construct the SbEC-DED code on Φ as in the following theorem.

Theorem 6.23 *Let Φ be a generating set with size v containing the identity element $\boldsymbol{I}$ in $GF(2^b)$. Then the following $\boldsymbol{H}$ matrix defines an SbEC-DED code with code length in bits $N = b \cdot v^{r-1}$ and check-bit length $R = r \cdot b$ for $r \geq 2$:*

$$\boldsymbol{H} = \left[\begin{array}{c|c|c|c|c} & \boldsymbol{I}\ \boldsymbol{I}\ \cdots\ \boldsymbol{I} & \boldsymbol{I}\ \boldsymbol{I}\ \cdots\ \boldsymbol{I} & & \boldsymbol{I}\ \boldsymbol{I}\ \cdots\ \boldsymbol{I} \\ & & \boldsymbol{I}\ \boldsymbol{I}\ \cdots\ \boldsymbol{I} & & \boldsymbol{I}\ \boldsymbol{I}\ \cdots\ \boldsymbol{I} \\ \overline{\boldsymbol{H}}_r & & & \cdots & \vdots \\ & \overline{\boldsymbol{H}}_{r-1} & \overline{\boldsymbol{H}}_{r-2} & & \boldsymbol{I}\ \boldsymbol{I}\ \cdots\ \boldsymbol{I} \\ & & & & \\ & & & & \overline{\boldsymbol{H}}_2 \end{array}\right].$$

Example 6.14

For an odd number of $b = 9$, as we have shown in Example 6.13, there exist generating sets Φ_4 and Φ_5 containing 11 elements in $GF(2^9)$. By Theorem 6.21,

$$\Phi_6 = (\mathbf{T}^{73})^{-1} \cdot \Phi_4 = \{\mathbf{0},\ \mathbf{I},\ \mathbf{T}^{50},\ \mathbf{T}^{63},\ \mathbf{T}^{114},\ \mathbf{T}^{160},\ \mathbf{T}^{196},\ \mathbf{T}^{211},\ \mathbf{T}^{352},\ \mathbf{T}^{407},\ \mathbf{T}^{421}\}$$

is also a generating set that contains the identity element $\mathbf{I}$. Thus Φ_6 can be used as in Theorem 6.23, and we obtain an SbEC-DED code with code length in bits $N = 9 \cdot 11^{r-1}$ that is much larger than that of the code given in Theorem 6.19 meaning $N = 9 \cdot 8(8^{r-1} - 1)/7$.

6.3 SINGLE-BYTE ERROR CORRECTING AND DOUBLE-BIT ERROR CORRECTING (SbEC-DEC) CODES

This section deals with the codes correcting both single b-bit byte errors and random double-bit errors, but not correcting simultaneously. These type of codes are applied

today's semiconductor memories with wide I/O data organization, which are extensively found in the systems that are continually exposed by strong electromagnetic waves, neutrons, and other cosmic rays. In these situations memory systems are highly vulnerable to random double-bit errors, and therefore, in addition to correcting b-bit byte errors, random double-bit error correction is needed to reduce the bit error rate to an acceptable level. The SbEC-DEC codes with $b = 4$ bits will be applied to the (4, 2) concept machine in Subsection 12.4.1. Here we consider the situation where double-bit errors occur simultaneously in one chip with wide I/O data B rather than randomly in the whole word. The later subsection deals with this new type of SbEC-DEC codes, denoted as SbEC-(DEC)$_B$ codes.

6.3.1 SbEC-DEC Codes

This subsection demonstrates how these type of codes can be designed, how to determine the code length bounds, and evaluation of the codes designed [UMAN02a].

Code Conditions and Bounds The following theorems are fundamental to SbEC-DEC code constructions.

Theorem 6.24 *Let $\boldsymbol{E}_{sb}$ be the set of single b-bit byte error patterns, and $\boldsymbol{E}_d$ be the set of double-bit error patterns that corrupt exactly two bytes. The null space of $\boldsymbol{H}$ describes a binary linear SbEC-DEC code if and only if*

1. $E \cdot \boldsymbol{H}^T \neq 0$ *for all* $E \in \{\boldsymbol{E}_{sb} \cup \boldsymbol{E}_d\}$,
2. $E_1 \cdot \boldsymbol{H}^T \neq E_2 \cdot \boldsymbol{H}^T$ *for all* $E_1, E_2 \in \{\boldsymbol{E}_{sb} \cup \boldsymbol{E}_d\}$ *with* $E_1 \neq E_2$,

where $\boldsymbol{E}_{sb} \cap \boldsymbol{E}_d = \Phi$, the null set, and $\boldsymbol{H}^T$ is the transpose of $\boldsymbol{H}$.

This theorem can be easily proved, and therefore its proof is omitted.

Theorem 6.25 *A linear SbEC-DEC code needs at least 2b check bits.*

Proof Condition 2 of Theorem 6.24 implies that $w(E_1 + E_2)$ binary columns of $\mathbf{H}$ are linearly independent. Since $w(E_1 + E_2)$ takes all the integer values between 1 and $2b$ inclusive for all $E_1, E_2 \in \mathbf{E}_{sb}$, $2b$ binary columns of $\mathbf{H}$ are linearly independent. Consequently a linear SbEC-DEC code needs at least $2b$ check bits. Q.E.D.

Theorem 6.26 *A linear $(N,\ K)$ SbEC-DEC code exists only if*

$$N - K \geq \left\lceil \log_2 \left[\frac{N}{b}(2^b - 1) + \frac{1}{2}N(N - b) + 1 \right] \right\rceil,$$

where $\lceil x \rceil$ shows the smallest integer greater than or equal to x.

Proof A codeword of length N (bits) has $N(2^b - 1)/b$ distinct single b-bit byte errors. This includes all the single-bit errors as well as double-bit errors occurring within a b-bit byte. The number of distinct double-bit errors that do not occur within a b-bit byte is

given by $N(N-b)/2$. For the code capable of correcting random double-bit errors and single b-bit byte errors, all these error patterns should generate $N(2^b-1)/b+N(N-b)/2$ distinct error syndromes. From this, we have

$$2^R - 1 = 2^{N-K} - 1 \geq \frac{N}{b}(2^b - 1) + \frac{1}{2}N(N-b).$$

By simply re-arranging variables, we can show that the inequality in Theorem 6.26 holds. Q.E.D.

Code Design Method Here a generic code design method, that is, a code design method applicable to any value of byte length b is demonstrated.

Theorem 6.27 *Let $r \geq b$ and α be a primitive element of $GF(2^r)$. Define the $r \times b$ binary matrix $\boldsymbol{T}^i$ and $\widetilde{\boldsymbol{T}}^i$ as follows:*

$$\boldsymbol{T}^i = \begin{bmatrix} | & | & | & & | \\ \alpha^i & \alpha^{i+1} & \alpha^{i+2} & \cdots & \alpha^{i+b-1} \\ | & | & | & & | \end{bmatrix},$$

$$\widetilde{\boldsymbol{T}}^i = \begin{bmatrix} | & | & | & & | \\ \alpha^{3i} & \alpha^{3(i+1)} & \alpha^{3(i+2)} & \cdots & \alpha^{3(i+b-1)} \\ | & | & | & & | \end{bmatrix},$$

where $i = 0, 1, 2, \cdots, 2^r - 2$ and α^j denotes a binary column vector of $GF(2^r)$ for $0 \leq j < 2^r - 2$. The null space of

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I}_b & \boldsymbol{I}_b & \boldsymbol{I}_b & \boldsymbol{I}_b & \cdots & \boldsymbol{I}_b \\ \boldsymbol{T}^0 & \boldsymbol{T}^b & \boldsymbol{T}^{2b} & \boldsymbol{T}^{3b} & \cdots & \boldsymbol{T}^{(n-1)b} \\ \widetilde{\boldsymbol{T}}^0 & \widetilde{\boldsymbol{T}}^b & \widetilde{\boldsymbol{T}}^{2b} & \widetilde{\boldsymbol{T}}^{3b} & \cdots & \widetilde{\boldsymbol{T}}^{(n-1)b} \end{bmatrix}$$

is an SbEC-DEC code with the code length in bits $N = n \cdot b$ and a check-bit length $R = 2r + b$, where $\boldsymbol{I}_b$ denotes the $b \times b$ identity matrix and $n = \lfloor 2^r/b \rfloor$.

Proof We know that the submatrix containing the upper $b + r$ binary rows of the $\mathbf{H}$ matrix represents the parity-check matrix of an SbEC code and the submatrix containing the lower $2r$ binary rows represents the parity-check matrix of a BCH DEC code. Therefore to show that the code is indeed SbEC-DEC, we only need to show that condition 2 of Theorem 6.24 is satisfied for $E_1 \in \mathbf{E}_d$ and $E_2 \in \mathbf{E}_{sb}$; that is, double-bit errors, when they occur in two byte positions, generate syndromes that are not equal to single-byte error syndromes. Let $E_1 \in \mathbf{E}_d$ and $E_2 \in \mathbf{E}_{sb}$ such that $E_1 \cdot \mathbf{H}^T = S_1$ and $E_2 \cdot \mathbf{H}^T = S_2$, where S_1 and S_2 are syndrome patterns corresponding to errors E_1 and E_2, respectively. Since $E_1 \in \mathbf{E}_d$, there exists $u_1, u_1' \in GF(2^b)$ with $w(u_1) = w(u_1') = 1$ such that $S_1 = [u_1 + u_1' \ s_1 \ s_1']^T$, where $s_1, s_1' \in GF(2^r)$. Similarly $S_2 = [u_2 \ s_2 \ s_2']^T$ for some $u_2 \in GF(2^b)$ with $w(u_2) \neq 0$, where $s_2, s_2' \in GF(2^r)$. Suppose $S_1 = S_2$; then, $u_1 + u_1' = u_2$, meaning $w(u_1 + u_1') = w(u_2)$ as well. Clearly, $w(u_1 + u_1') = 0$ cannot happen because $w(u_2) \neq 0$. Therefore we have $w(u_2) = 2$. This means that $w(E_1) + w(E_2) = 4$, which contradicts the DEC capability;

that is, none of the four binary columns of $\mathbf{H}$ are linearly dependent. It is apparent that the code length in bytes is given by $n = \lfloor 2^r/b \rfloor$. Q.E.D.

Decoding Procedure The SbEC-DEC codes derived by using Theorem 6.27 can be easily decoded by combining the decoding procedures of SbEC and DEC codes. Let v be the received word. The syndrome S can be calculated as follows:

$$v \cdot \mathbf{H}^T = S = [S_0\, S_1\, S_2],$$

where $S_0 \in GF(2^b)$ and $S_1, S_2 \in GF(2^r)$. If $S = 0$, the received word is assumed to be error free. Otherwise, if $w(S_0) = 0$ or $w(S_0) = 2$, we assume that there exist random double-bit errors; then we apply the existing decoding method of double-bit error correction by using the syndromes S_1 and S_2 to find the error location. If, on the other hand, $w(S_0) \neq 0$ and $w(S_0) \neq 2$, we apply the decoding procedure of single-byte error correction.

Evaluation Figure 6.19 illustrates the relationship between the information-bit length and the check-bit length of the S4EC-DEC code [DAVY89] and the SbEC-DEC codes with $b = 6$, and 8 bits, and includes the bounds for these codes. We observe that for the practical information-bit length of 64 the S8EC-DEC code requires 24 check bits, and the Davydov-Labinskaya S4EC-DEC code shown in next subsection requires only 15 check bits. In comparison, the S4EC-DEC code shown here requires 18 check bits. However, RAM chips with wide I/O data, such as 16 and 32 bits, are usually made up of highly independent memory subarrays [SAEK96, SUNA95, NUMA89]. In these cases we can consider the subarray output, which is typically 4 or 8 bits, as a byte. Hence the indicated codes can be applied to memory systems using RAM chips with wide I/O data.

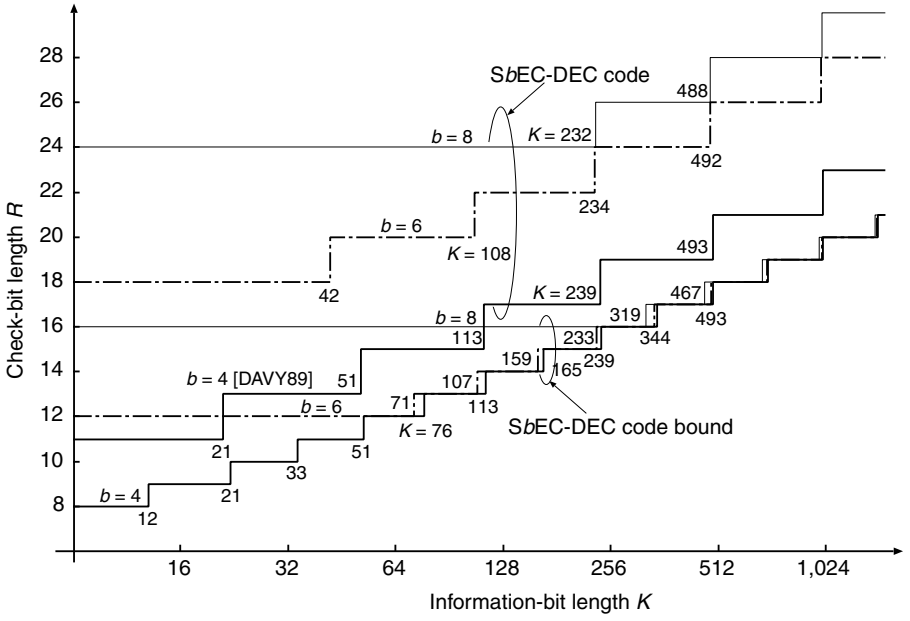


Figure 6.19 *Check-bit lengths compared with information-bit lengths of the SbEC-DEC codes with $b = 4$, 6, and 8 bits.* Source: *[UMAN02a]. © 2002 IEICE Japan.*

6.3.2 S4EC-DEC Codes — Davydov-Labinskaya Code —

Davydov and Drozhzhina-Labinskaya have constructed an excellent SbEC-DEC code with the practical code parameter of $b = 4$ bits [DAVY89].

Code Design Method Let α be a primitive element of $GF(2^r)$. Define

$$\mathbf{H}_{(\alpha^i,\ \alpha^{i+1})} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ \begin{pmatrix} 0 \\ \alpha^i \end{pmatrix} & \begin{pmatrix} 1 \\ \alpha^i \end{pmatrix} & \begin{pmatrix} 0 \\ \alpha^{i+1} \end{pmatrix} & \begin{pmatrix} 1 \\ \alpha^{i+1} \end{pmatrix} \\ \begin{pmatrix} 0 \\ \alpha^i \end{pmatrix}^3 & \begin{pmatrix} 1 \\ \alpha^i \end{pmatrix}^3 & \begin{pmatrix} 0 \\ \alpha^{i+1} \end{pmatrix}^3 & \begin{pmatrix} 1 \\ \alpha^{i+1} \end{pmatrix}^3 \end{bmatrix},$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 \\ h_{i0} & h_{i1} & h_{i2} & h_{i3} \\ h_{i0}^3 & h_{i1}^3 & h_{i2}^3 & h_{i3}^3 \end{bmatrix} \begin{matrix} \updownarrow 1 \\ \updownarrow r+1, \\ \updownarrow r+1 \end{matrix}$$

$$h_{i0} = \begin{pmatrix} 0 \\ \alpha^i \end{pmatrix}, h_{i1} = \begin{pmatrix} 1 \\ \alpha^i \end{pmatrix}, h_{i2} = \begin{pmatrix} 0 \\ \alpha^{i+1} \end{pmatrix}, h_{i3} = \begin{pmatrix} 1 \\ \alpha^{i+1} \end{pmatrix},$$

$$0, 1 \in GF(2),$$

where $h_{i0},\ h_{i1},\ h_{i2}$, and h_{i3} are elements of $GF(2^{r+1})$. This $\mathbf{H}_{(\alpha^i,\ \alpha^{i+1})}$ shows the parity-check matrix of the BCH code with a minimum Hamming distance of 6. In the $\mathbf{H}_{(\alpha^i,\ \alpha^{i+1})}$ matrix above, there exist the following properties:

Property 1.

$$h_{i0} + h_{i1} + h_{i2} + h_{i3} = \begin{bmatrix} 0 \\ \mathbf{0}_r \end{bmatrix} = \mathbf{0}_{r+1}.$$

Property 2.

$$h_{i0} + h_{i1} = \begin{bmatrix} 1 \\ \mathbf{0}_r \end{bmatrix} = \mathbf{1}_{r+1},$$

$$h_{i2} + h_{i3} = \begin{bmatrix} 1 \\ \mathbf{0}_r \end{bmatrix} = \mathbf{1}_{r+1}.$$

The $\mathbf{0}_x$ and $\mathbf{1}_{r+1}$ stand for binary column vectors with x zeros and $(1\ \mathbf{0}_r^T)^T = (1\ \overbrace{0\ \cdots\ 0}^{r})^T$, respectively.

Lemma 6.7

$$\begin{aligned} h_{i0}^3 + h_{i1}^3 + h_{i2}^3 + h_{i3}^3 &= (h_{i0} + h_{i2})^2 + (h_{i0} + h_{i2}) \\ &= (h_{i1} + h_{i3})^2 + (h_{i1} + h_{i3}) \\ &\neq \mathbf{0}_{r+1}. \end{aligned}$$

Proof From property 2, $h_{i1} = \mathbf{1} + h_{i0}$ and $h_{i3} = \mathbf{1} + h_{i2}$, where $\mathbf{1} = \mathbf{1}_{r+1}$. Substituting these to $h_{i0}^3 + h_{i1}^3 + h_{i2}^3 + h_{i3}^3$, we have

$$\begin{aligned} & h_{i0}^3 + h_{i1}^3 + h_{i2}^3 + h_{i3}^3 \\ &= h_{i0}^3 + (\mathbf{1} + h_{i0})^3 + h_{i2}^3 + (\mathbf{1} + h_{i2})^3 \\ &= h_{i0}^2 + h_{i2}^2 + h_{i0} + h_{i2} \\ &= (h_{i0} + h_{i2})^2 + (h_{i0} + h_{i2}) \\ &\neq \mathbf{0}_{r+1}. \end{aligned}$$

Similarly, substituting $h_{i0} = \mathbf{1} + h_{i1}$ and $h_{i2} = \mathbf{1} + h_{i3}$, we have

$$\begin{aligned} & h_{i0}^3 + h_{i1}^3 + h_{i2}^3 + h_{i3}^3 \\ &= (\mathbf{1} + h_{i1})^3 + h_{i1}^3 + (\mathbf{1} + h_{i3})^3 + h_{i3}^3 \\ &= h_{i1}^2 + h_{i3}^2 + h_{i1} + h_{i3} \\ &= (h_{i1} + h_{i3})^2 + (h_{i1} + h_{i3}) \\ &\neq \mathbf{0}_{r+1}. \end{aligned}$$

Q.E.D.

Lemma 6.8 *Addition of any three columns in* $\boldsymbol{H}_{(\alpha^i,\ \alpha^{i+1})}$ *gives* $[1\ \boldsymbol{x}\ \boldsymbol{y}]^T$, *where* $\boldsymbol{x}$ *is equal to the remaining one column other than these three columns, and* $\boldsymbol{y} + \boldsymbol{x}^3 = h_{i0}^3 + h_{i1}^3 + h_{i2}^3 + h_{i3}^3$.

Proof If we choose distinct three column vectors such as the a-th, b-th, and c-th columns in $\mathbf{H}_{(\alpha^i,\ \alpha^{i+1})}$ (i.e., the remaining column is the d-th one), $a,\ b,\ c,\ d \in \{0,\ 1,\ 2,\ 3\}$, then from property 1,

$$\begin{aligned} \boldsymbol{x} = h_{ia} + h_{ib} + h_{ic} = h_{id} \quad & (\because\ h_{ia} + h_{ib} + h_{ic} + h_{id} = 0), \\ \boldsymbol{y} + \boldsymbol{x}^3 &= \left(h_{ia}^3 + h_{ib}^3 + h_{ic}^3\right) + (h_{ia} + h_{ib} + h_{ic})^3 \\ &= \left(h_{ia}^3 + h_{ib}^3 + h_{ic}^3\right) + h_{id}^3 \\ &= h_{i0}^3 + h_{i1}^3 + h_{i2}^3 + h_{i3}^3. \end{aligned}$$

Q.E.D.

Theorem 6.28 *Let* $\boldsymbol{H}$ *be defined as follows*:

$$\boldsymbol{H} = \left[\boldsymbol{H}_{(0,\ 1)}\ \boldsymbol{H}_{(\alpha,\ \alpha^2)}\ \boldsymbol{H}_{(\alpha^3,\ \alpha^4)}\ \cdots\ \boldsymbol{H}_{(\alpha^i,\ \alpha^{i+1})}\ \cdots\ \boldsymbol{H}_{(\alpha^{2^r-3},\ \alpha^{2^r-2})}\ \right].$$

Then the null space of $\mathbf{H}$ *is the S4EC-DEC code with the code length in bits* $N = 2^{r+1}$ *and the check-bit length* $R = 2r + 3$.

Proof Syndromes are presented in Table 6.5 for correctable errors such as single-bit errors, double-bit errors, weight-three errors as well as weight-four errors in the i-th byte.

From this table we see that the syndromes of all correctable errors are nonzero and distinct from each other. Also syndromes of weight-three errors and weight-four errors in the i-th byte are different from those in the j-th byte, $i \neq j$, because $h_{i0}^3 + h_{i1}^3 + h_{i2}^3 + h_{i3}^3 \neq h_{j0}^3 + h_{j1}^3 + h_{j2}^3 + h_{j3}^3$ and $h_{i\,d} \neq h_{j\,d}$.

It is apparent that the code expressed by $\mathbf{H}$ has a code length in bits $N = 2^{r+1}$ and a check-bit length $R = 2r + 3$. Q.E.D.

Example 6.15 (80, 65) S4EC-DEC code [DAVY89]

If we use $r = 6$ in the code design, then we have $N = 2^7 = 128$ bits and $R = 15$ bits. This gives us a (128, 113) S4EC-DEC code, where α is a root of primitive polynomial $\mathbf{g}(x) = x^6 + x + 1$, and expressed as binary column vector with sixth degree:

$$\left[\begin{array}{cccc:cccc:c:cccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & 1 & 1 & 1 & 1 \\
\binom{0}{\mathbf{0}} & \binom{1}{\mathbf{0}} & \binom{0}{\mathbf{1}} & \binom{1}{\mathbf{1}} & \binom{0}{\alpha} & \binom{1}{\alpha} & \binom{0}{\alpha^2} & \binom{1}{\alpha^2} & & \binom{0}{\alpha^{61}} & \binom{1}{\alpha^{61}} & \binom{0}{\alpha^{62}} & \binom{1}{\alpha^{62}} \\
\binom{0}{\mathbf{0}}^3 & \binom{1}{\mathbf{0}}^3 & \binom{0}{\mathbf{1}}^3 & \binom{1}{\mathbf{1}}^3 & \binom{0}{\alpha}^3 & \binom{1}{\alpha}^3 & \binom{0}{\alpha^2}^3 & \binom{1}{\alpha^2}^3 & & \binom{0}{\alpha^{61}}^3 & \binom{1}{\alpha^{61}}^3 & \binom{0}{\alpha^{62}}^3 & \binom{1}{\alpha^{62}}^3
\end{array}\right]$$

To obtain a practical code with 64 information bits, we can shorten this code. After deleting the first two columns and the last 46 columns from the matrix above, we obtain the parity-check matrix in Figure 6.20, showing an (80, 65) S4EC-DEC code in its binary form.

Figure 6.21 shows the relationship between the check-bit length and the information-bit length of S4EC-DEC code and its bound.

Decoding Procedure For a received word $\boldsymbol{v}$, we compute the syndrome as

$$\boldsymbol{v} \cdot \mathbf{H}^T = S = \begin{bmatrix} p \\ \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}.$$

TABLE 6.5 Syndromes for Correctable Errors

Correctable Errors	Syndrome $\begin{bmatrix} p \\ \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \begin{matrix} \updownarrow 1 \\ \updownarrow r+1 \\ \updownarrow r+1 \end{matrix}$
Single-bit errors	$\begin{bmatrix} 1 \\ \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}$ $p = 1$, $\boldsymbol{y} = \boldsymbol{x}^3$
Double-bit errors	$\begin{bmatrix} 0 \\ \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}$ $p = 0$, $\boldsymbol{x} \neq \mathbf{0}_{r+1}$
Weight-3 errors in the *i*-th byte	$\begin{bmatrix} 1 \\ \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix}$ $p = 1$, $\boldsymbol{x} = h_{id}$, $\boldsymbol{y} \neq \boldsymbol{x}^3$ $\left(\because \boldsymbol{y} + \boldsymbol{x}^3 = h_{i0}^3 + h_{i1}^3 + h_{i2}^3 + h_{i3}^3 \neq \mathbf{0}_{r+1} \right)$
Weight-4 errors in the *i-th byte*	$\begin{bmatrix} 0 \\ \mathbf{0} \\ \boldsymbol{y} \end{bmatrix}$ $p = 0$, $\boldsymbol{x} = \mathbf{0}_{r+1}$, $\boldsymbol{y} = h_{i0}^3 + h_{i1}^3 + h_{i2}^3 + h_{i3}^3 \neq \mathbf{0}_{r+1}$

The decoding is done as follows:

1. $S = \mathbf{0} \longrightarrow$ No errors.
2. $S \neq \mathbf{0}$.

   a. $(p = 1) \wedge (\boldsymbol{y} = \boldsymbol{x}^3) \longrightarrow$ single-bit errors.
   b. $(p = 1) \wedge (\boldsymbol{y} \neq \boldsymbol{x}^3) \longrightarrow$ weight-three single-byte errors.
   c. $(p = 0) \wedge (\boldsymbol{x} \neq \mathbf{0}) \longrightarrow$ random double-bit errors.
   d. $(p = 0) \wedge (\boldsymbol{x} = \mathbf{0}) \longrightarrow$ weight-four single-byte errors.

Figure 6.22 shows a decoding procedure for the S4EC-DEC code.

1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111
0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101	0101
1100	0000	0000	1100	0000	0011	1100	0000	1100	1100	0011	1111	1100	1100	0000	1111	1100	0011	0000	1100
0011	0000	0000	1111	0000	0011	0011	0000	1111	1111	0011	0000	0011	1111	0000	1100	0011	0011	1100	1111
0000	1100	0000	0011	1100	0000	1100	1100	0011	1111	1100	1100	0000	1111	1100	0011	0000	1100	1111	0011
0000	0011	0000	0000	1111	0000	0011	0011	0000	1111	1111	0011	0000	0011	1111	0000	1100	0011	0011	1100
0000	0000	1100	0000	0011	1100	0000	1100	1100	0011	1111	1100	1100	0000	1111	1100	0011	0000	1100	1111
0000	0000	0011	0000	0000	1111	0000	0011	0011	0000	1111	1111	0011	0000	0011	1111	0000	1100	0011	0011
0101	0101	1001	0110	1001	1010	1001	0110	1010	1010	0101	0110	1010	0101	0110	0101	0110	1001	1010	0110
0100	0010	1101	0111	1001	0111	0101	1100	0110	1110	0010	1110	1001	1010	0100	0011	1100	0100	1100	0001
0101	1100	0000	0010	1111	0011	1001	0011	1110	0001	0011	1010	0101	1110	1100	0001	1010	1111	1000	1110
1100	0100	1011	1110	0110	1011	0111	0011	0101	0100	1101	1100	0100	1001	0010	1001	1110	0100	1101	1011
0001	0011	0010	1010	0011	0111	0001	1110	1011	1001	1011	0101	1111	0101	1101	0010	1110	0110	1010	1000
0000	1111	0101	1111	0001	1101	0011	0110	1001	1101	0000	0010	0101	1100	0111	1101	0001	0111	0110	0100
0011	0100	0110	1110	1001	1110	0100	1101	1000	0100	0100	1001	0101	1010	0000	1100	1110	1111	0011	1010

Figure 6.20 *(80, 65) S4ED-DEC code.*

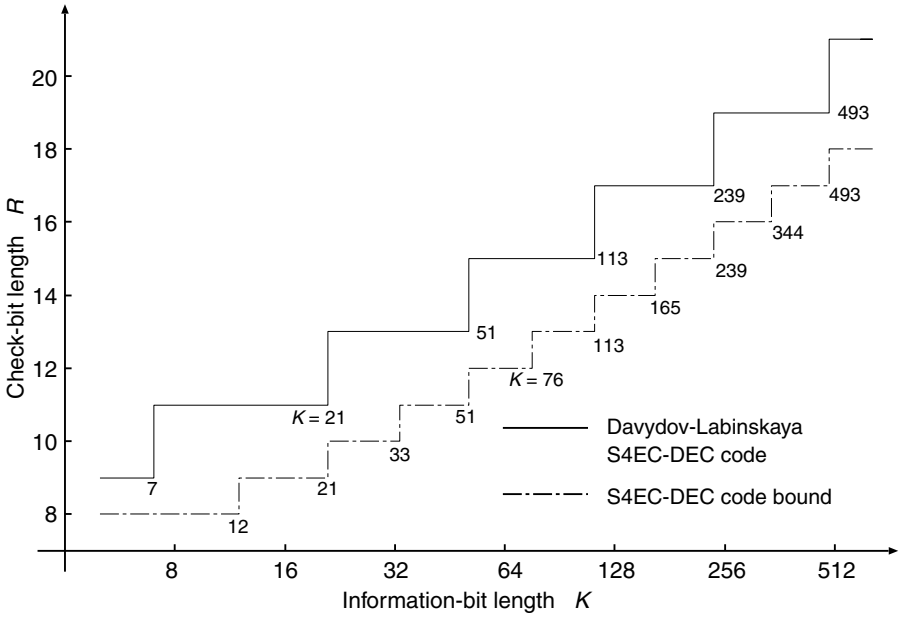


Figure 6.21 *Check-bit lenghts compared with information-bit lengths of the Davydov-Labinskaya S4EC-DEC codes.*

6.3.3 S*b*EC-(DEC)$_B$ Codes

Existing byte error control codes require too many check bits if applied to a memory system that uses any of the recent semiconductor RAM chips with wide I/O data such as 16 or 32 bits. However, semiconductor RAM chips are highly vulnerable to random double-bit within a chip errors when they are used in some applications, such as in satellite memory systems. In satellite systems it is therefore necessary to design suitable new codes with a double-bit within a chip error correcting capability for the computer memory [UMAN02b].

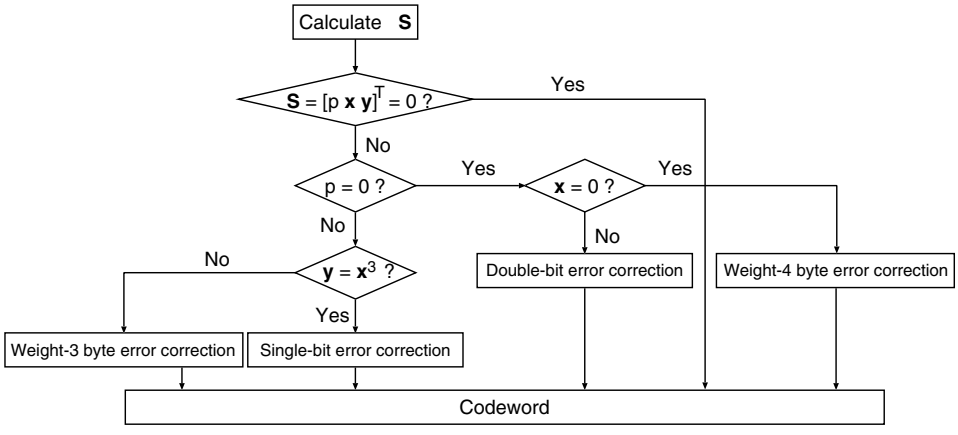


Figure 6.22 *Decoding procedure for the S4EC-DEC code.*

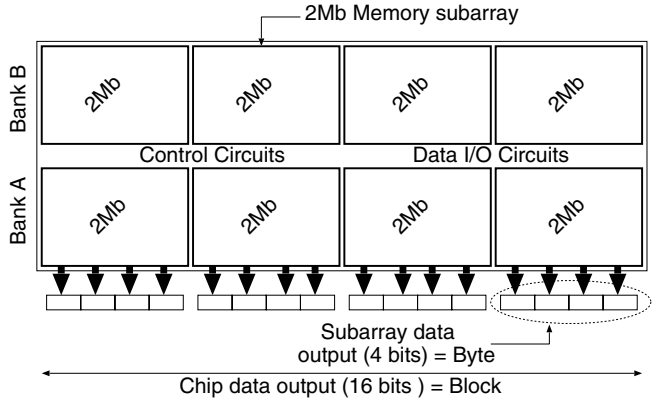


Figure 6.23 *Organization of the 16 Mb semiconductor DRAM chip with 16-bit input / output.*

The latest semiconductor RAM chips have a multi-bank architecture. Each bank is generally constructed with highly independent subdivided memory arrays called memory subarrays. Figure 6.23 shows an example of the architecture of a 16 Mb DRAM chip [SUNA95]. Note how a 16-bit data output, consisting of four 4-bit outputs from four 2 Mb memory subarrays is readout from the chip. Since the subarrays within the chip are highly independent of each other, the 4-bit data output from a 2 Mb memory subarray forms a group of data bits called a byte. The entire 16-bit output of the memory chip is called a *block*. Figure 6.24 illustrates the hierarchical organization of the bit, byte, and block of a chip data output. Unlike the previous byte-organized memory systems where a chip output is called a byte, here the subarray output is called a *byte* and the chip output is called a *block*.

Because of the presently extensive use of the above type of RAM chips in high-speed memories, this subsection is devoted to a class of codes called single b-bit byte error correcting and double-bit within a B-bit block error correcting (SbEC-(DEC)$_B$) codes. These codes correct single b-bit byte errors and random double-bit errors occurring within a chip output.

Code Conditions and Bounds Throughout this subsection we will consider two sets of error patterns, $\mathbf{E}_b$ and $\mathbf{E}_{2/B}$, where $\mathbf{E}_b = \{E \in GF(2^B) \mid E$ is a single b-bit byte error pattern with $w(E) \neq 2\}$ and $\mathbf{E}_{2/B} = \{E \in GF(2^B) \mid w(E) = 2\}$. The error set $\mathbf{E}_{2/B}$ includes all double-bit error patterns that corrupt a single memory chip, in other words, error patterns

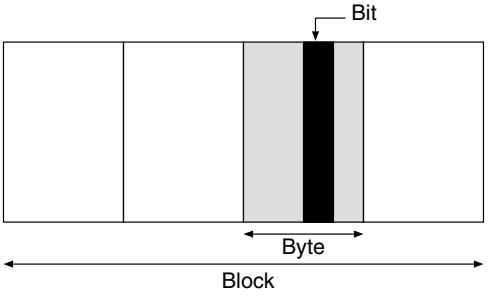


Figure 6.24 *Organization of the bit, byte, and block of memory chip data output.* Source: *[UMAN02b]. © 2002 IEICE Japan.*

of all random double-bit within a B-bit block errors. The error set $\mathbf{E}_b$ contains all single b-bit byte error patterns excluding the double-bit error patterns. Since, for all $E_0 \in \mathbf{E}_b$, $0 \neq w(E_0) \neq 2$ and, for all $E_1 \in \mathbf{E}_{2/B}$, $w(E_1) = 2$, we have $\mathbf{E}_b \cap \mathbf{E}_{2/B} = \emptyset$, where $\emptyset$ is the empty set. The following theorems are fundamental to the SbEC-(DEC)$_B$ code design.

Theorem 6.29 *Let $\boldsymbol{H} = [\boldsymbol{H}_0\ \boldsymbol{H}_1\ \boldsymbol{H}_2\ \cdots\ \boldsymbol{H}_{n-1}]$ where $\boldsymbol{H}_i$, for $0 \leq i < n$ is an $R \times B$ binary submatrix. The null space of $\boldsymbol{H}$ describes a binary linear $(nB,\ nB - R)$ SbEC-(DEC)$_B$ code if and only if*

1. $E \cdot \boldsymbol{H}_i^T \neq 0$ *for all* $E \in \{\boldsymbol{E}_b \cup \boldsymbol{E}_{2/B}\}$, $0 \leq i < n$,
2. $E_1 \cdot \boldsymbol{H}_i^T \neq E_2 \cdot \boldsymbol{H}_i^T$ *for all* $E_1, E_2 \in \{\boldsymbol{E}_b \cup \boldsymbol{E}_{2/B}\}$, $E_1 \neq E_2$, $0 \leq i < n$,
3. $E_1 \cdot \boldsymbol{H}_i^T \neq E_2 \cdot \boldsymbol{H}_j^T$ *for all* $E_1, E_2 \in \{\boldsymbol{E}_b \cup \boldsymbol{E}_{2/B}\}$, $0 \leq i < j < n$,

where $\boldsymbol{H}^T$ is the transpose of $\boldsymbol{H}$.

Proof To correct all single b-bit byte errors, the following three conditions are necessary and sufficient:

1. $E \cdot \mathbf{H}_i^T \neq 0$ for all $E \in \mathbf{E}_b$, $0 \leq i < n$.
2. $E_1 \cdot \mathbf{H}_i^T \neq E_2 \cdot \mathbf{H}_i^T$ for all $E_1, E_2 \in \mathbf{E}_b$, $E_1 \neq E_2$, $0 \leq i < n$.
3. $E_1 \cdot \mathbf{H}_i^T \neq E_2 \cdot \mathbf{H}_j^T$ for all $E_1, E_2 \in \mathbf{E}_b$, $0 \leq i < j < n$.

To correct all double-bit errors within a B-bit block, the following three conditions are necessary and sufficient:

1. $E \cdot \mathbf{H}_i^T \neq 0$ for all $E \in \mathbf{E}_{2/B}$, $0 \leq i < n$.
2. $E_1 \cdot \mathbf{H}_i^T \neq E_2 \cdot \mathbf{H}_i^T$ for all $E_1, E_2 \in \mathbf{E}_{2/B}$, $E_1 \neq E_2$, $0 \leq i < n$.
3. $E_1 \cdot \mathbf{H}_i^T \neq E_2 \cdot \mathbf{H}_j^T$ for all $E_1, E_2 \in \mathbf{E}_{2/B}$, $0 \leq i < j < n$.

To distinguish between the single b-bit byte errors and the double-bit within a B-bit block errors, the following condition is necessary and sufficient:

$$E_1 \cdot \mathbf{H}_i^T \neq E_2 \cdot \mathbf{H}_j^T \quad \text{for all } E_1 \in \mathbf{E}_b, \text{all } E_2 \in \mathbf{E}_{2/B}, \text{ and } 0 \leq i,j < n.$$

Conditions 1, 2, and 3 of Theorem 6.29 include all these conditions and inversely all these conditions result in the conditions of Theorem 6.29. Therefore we have the necessary and sufficient conditions for correcting single b-bit byte errors and random double-bit errors within a B-bit block. Q.E.D.

Theorem 6.30 *A linear $(N,\ K)$ SbEC-(DEC)$_B$ code exists only if*

$$N - K \geq \left\lceil \log_2 \left[\frac{N}{b}(2^b - 1) + \frac{1}{2}N(B - b) + 1 \right] \right\rceil,$$

where $\lceil x \rceil$ shows the smallest integer greater than or equal to x.

Proof A codeword of length N (bits) can have $N(2^b - 1)/b$ distinct single b-bit byte errors. The number of double-bit errors occurring within a B-bit block, not the single b-bit byte errors themselves, is $N(B - b)/2$. From this we have

$$\begin{aligned} 2^R - 1 &= 2^{N-K} - 1 \\ &\geq \frac{N}{b}(2^b - 1) + \frac{1}{2}N(B - b). \end{aligned}$$

By simple re-arrangement of variables in this inequality, we can show that the inequality in Theorem 6.30 holds. Q.E.D.

Theorem 6.31 *A binary linear SbEC-(DEC)$_B$ code needs at least $2b$ check bits.*

Proof An SbEC-(DEC)$_B$ code needs at least $2b$ check bits because conditions 2 and 3 of Theorem 6.29 imply that maximum $2b$ binary columns of the parity-check matrix $\mathbf{H}$ are linearly independent. Q.E.D.

Code Design Method The code design method presented here uses short $(B,\ B - R')$ SbEC-DEC codes, shown previously in Subsection 6.3.1, to obtain practical SbEC-(DEC)$_B$ codes with longer code lengths. The codes obtained by this method are practical in that they do not require too many redundant bits at the usual information-bit lengths and they are easily parallel decodable.

Theorem 6.32 *Let $\hat{\boldsymbol{H}}$ be a parity-check matrix of a $(B,\ B - R')$ SbEC-DEC code. Let α be a primitive element of $GF(2^r)$ such that $r \geq max(b, \lceil log_2(B + 1) \rceil)$. Define the $r \times B$ binary submatrix*

$$\boldsymbol{M}_j = \begin{bmatrix} | & | & | & & | \\ \alpha^j & \alpha^{j+1} & \alpha^{j+2} & \cdots & \alpha^{j+B-1} \\ | & | & | & & | \end{bmatrix}$$

for $0 \leq j \leq 2^r - 2$, where $\begin{vmatrix} \\ \alpha^k \end{vmatrix}$ denotes the binary column vector of $GF(2^r)$ for $0 \leq k \leq 2^r - 2$. The null space of

$$\boldsymbol{H} = \begin{bmatrix} \hat{\boldsymbol{H}} & \hat{\boldsymbol{H}} & \hat{\boldsymbol{H}} & \cdots & \hat{\boldsymbol{H}} \\ \boldsymbol{O} & \boldsymbol{M}_0 & \boldsymbol{M}_1 & \cdots & \boldsymbol{M}_{2^r-2} \end{bmatrix}$$

is an SbEC-(DEC)$_B$ code with code length in bits $N = 2^r B$ and check-bit length $R = R' + r$. Here $\boldsymbol{O}$ is an all zero $r \times B$ binary submatrix.

Proof Since $\hat{\mathbf{H}}$ is a parity-check matrix of a $(B,\ B - R')$ SbEC-DEC code, we have $E \cdot \hat{\mathbf{H}}^T \neq 0$, and also $E_1 \cdot \hat{\mathbf{H}}^T \neq E_2 \cdot \hat{\mathbf{H}}^T$ for any $E,\ E_1,\ E_2 \in \{\mathbf{E}_b \cup \mathbf{E}_{2/B}\}$ with $E_1 \neq E_2$. Therefore conditions 1 and 2 of Theorem 6.29 are clearly satisfied.

On the other hand, we observe that the condition $r \geq \max(b, \lceil \log_2(B+1) \rceil)$ implies that $2^r \geq B+1$ and $r \geq b$. Subsequently $2^r \geq B+1$ implies that $0 \neq E \cdot \mathbf{M}_0^T \in GF(2^r)$ for all $E \in \mathbf{E}_{2/B}$, and $r \geq b$ implies that $0 \neq E \cdot \mathbf{M}_0^T \in GF(2^r)$ for all $E \in \mathbf{E}_b$. Therefore for all $E \in \{\mathbf{E}_b \cup \mathbf{E}_{2/B}\}$, $0 \neq E \cdot \mathbf{M}_0^T \in GF(2^r)$ holds. Let E_1 and $E_2 \in \{\mathbf{E}_b \cup \mathbf{E}_{2/B}\}$ be such that

$$E_1 \cdot \begin{bmatrix} \hat{\mathbf{H}} \\ \mathbf{M}_i \end{bmatrix}^T = E_2 \cdot \begin{bmatrix} \hat{\mathbf{H}} \\ \mathbf{M}_j \end{bmatrix}^T \quad \text{or}$$

$$E_1 \cdot \begin{bmatrix} \hat{\mathbf{H}} \\ \mathbf{M}_i \end{bmatrix}^T = E_2 \cdot \begin{bmatrix} \hat{\mathbf{H}} \\ \mathbf{O} \end{bmatrix}^T$$

for $0 \leq i \neq j < n$. We know that $E_1 \cdot \hat{\mathbf{H}}^T = E_2 \cdot \hat{\mathbf{H}}^T$ implies $E_1 = E_2$ because $\hat{\mathbf{H}}$ itself is a parity-check matrix of a $(B,\ R')$ SbEC-DEC code. Therefore $E_1 \cdot \mathbf{M}_i^T = E_2 \cdot \mathbf{M}_j^T$ or $E_1 \cdot \mathbf{M}_i^T = E_2 \cdot \mathbf{O}^T$ implies $\alpha^i \cdot (E_1 \cdot \mathbf{M}_0^T) = \alpha^j \cdot (E_2 \cdot \mathbf{M}_0^T)$ or $\alpha^i \cdot (E_1 \cdot \mathbf{M}_0^T) = 0$, where $0 \neq E_1 \cdot \mathbf{M}_0^T = E_2 \cdot \mathbf{M}_0^T \in GF(2^r)$. This is a contradiction because $0 \neq \alpha^i \neq \alpha^j$; thus condition 3 of Theorem 6.29 is also satisfied. Q.E.D.

As an example of the code we consider the practical case where the chip data output is 16 bits and the subarray data output is 4 bits, meaning $b = 4$ and $B = 16$. We need to choose r such that $r \geq b = 4$ and $2^r \geq B + 1 = 17$. This explains $r \geq 5$.

By using the (16, 7) S4EC-DEC code presented in [DAVY89] and taking $r = 5$, we can design a S4EC-(DEC)$_{16}$ code with code length $N = 2^5 \times 16 = 512$ bits and a check-bit length $R = (16 - 7) + 5 = 14$. We can shorten this (512, 498) S4EC-(DEC)$_{16}$ code to obtain a practical code with information-bit lengths 64, 128, and 256. Figure 6.25 shows the first and the last four blocks of the (512, 498) S4EC-(DEC)$_{16}$ code in binary form.

Decoding Procedure Let v be the received word. The syndrome S can be calculated as follows: $v \cdot \mathbf{H}^T = S = [S_0\, S_1]$, where $S_0 \in GF(2^{R'})$ and $S_1 \in GF(2^r)$. Then

$$\begin{bmatrix} \hat{\mathbf{H}} & \hat{\mathbf{H}} & \hat{\mathbf{H}} & \cdots & \hat{\mathbf{H}} \\ \mathbf{O} & \mathbf{M}_0 & \mathbf{M}_1 & \cdots & \mathbf{M}_{2^r-2} \end{bmatrix} \begin{matrix} \rightarrow & S_0 \\ \rightarrow & S_1 \end{matrix}.$$

The syndrome vector S_0 corresponds to the $(B,\ B - R')$ SbEC-DEC code. We can decode S_0 by using any $(B,\ B - R')$ SbEC-DEC decoding methods [DAVY89, MASS96]. The decoding of S_0 could detect an uncorrectable error pattern or yield a correctable error pattern $E \in \{\mathbf{E}_b \cup \mathbf{E}_{2/B}\}$. In the latter case, if $S_1 = 0$, the first block is in error; otherwise, we calculate $E \cdot \mathbf{M}_i^T$ for $0 \leq i \leq 2^r - 2$ until we find some j, where $0 \leq j \leq 2^r - 2$, such that $E \cdot \mathbf{M}_j^T = S_1$ holds. If such a j is found successfully, the $(j+1)$-th block is in error, otherwise the error pattern is uncorrectable.

Evaluation Figure 6.26 shows the relationship between the information-bit lengths and the check-bit lengths of the SbEC-(DEC)$_B$ codes designed by Theorem 6.32 and the SbEC-(DEC)$_B$ bounds indicated in Theorem 6.30 for the two practical cases where $B = 16$ and $b = 4$, and $B = 8$ and $b = 4$. The S4EC-(DEC)$_8$ code presented here requires only 12 check bits for the practical information length of 64 bits. Further, for longer information-bit lengths

1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111
0101 0101 0101 0101	0101 0101 0101 0101	0101 0101 0101 0101	0101 0101 0101 0101
0000 0011 0011 1111	0000 0011 0011 1111	0000 0011 0011 1111	0000 0011 0011 1111
0011 0011 1111 0000	0011 0011 1111 0000	0011 0011 1111 0000	0011 0011 1111 0000
0000 1100 1111 1100	0000 1100 1111 1100	0000 1100 1111 1100	0000 1100 1111 1100
0100 0111 0000 1001	0100 0111 0000 1001	0100 0111 0000 1001	0100 0111 0000 1001
0001 1100 0100 1001	0001 1100 0100 1001	0001 1100 0100 1001	0001 1100 0100 1001
0010 0100 0001 1101	0010 0100 0001 1101	0010 0100 0001 1101	0010 0100 0001 1101
0011 1100 1111 1111	0011 1100 1111 1111	0011 1100 1111 1111	0011 1100 1111 1111
0000 0000 0000 0000	1000 0100 1011 0011	0000 1001 0110 0111	0001 0010 1100 1111
0000 0000 0000 0000	0100 0010 0101 1001	1000 0100 1011 0011	0000 1001 0110 0111
0000 0000 0000 0000	0010 0101 1001 1111	0100 1011 0011 1110	1001 0110 0111 1100
0000 0000 0000 0000	0001 0010 1100 1111	0010 0101 1001 1111	0100 1011 0011 1110
0000 0000 0000 0000	0000 1001 0110 0111	0001 0010 1100 1111	0010 0101 1001 1111

. . .

. . .

1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111	1111 1111 1111 1111
0101 0101 0101 0101	0101 0101 0101 0101	0101 0101 0101 0101	0101 0101 0101 0101
0000 0011 0011 1111	0000 0011 0011 1111	0000 0011 0011 1111	0000 0011 0011 1111
0011 0011 1111 0000	0011 0011 1111 0000	0011 0011 1111 0000	0011 0011 1111 0000
0000 1100 1111 1100	0000 1100 1111 1100	0000 1100 1111 1100	0000 1100 1111 1100
0100 0111 0000 1001	0100 0111 0000 1001	0100 0111 0000 1001	0100 0111 0000 1001
0001 1100 0100 1001	0001 1100 0100 1001	0001 1100 0100 1001	0001 1100 0100 1001
0010 0100 0001 1101	0010 0100 0001 1101	0010 0100 0001 1101	0010 0100 0001 1101
0011 1100 1111 1111	0011 1100 1111 1111	0011 1100 1111 1111	0011 1100 1111 1111
0101 1001 1111 0001	1011 0011 1110 0011	0110 0111 1100 0110	1111 0001 1011 1010
0010 1100 1111 1000	0101 1001 1111 0001	1011 0011 1110 0011	1111 1000 1101 1101
1100 1111 1000 1101	1001 1111 0001 1011	0011 1110 0011 0111	1000 1101 1101 0100
0110 0111 1100 0110	1100 1111 1000 1101	1001 1111 0001 1011	1100 0110 1110 1010
1011 0011 1110 0011	0110 0111 1100 0110	1100 1111 1000 1101	1110 0011 0111 0101

Figure 6.25 *Example of (512, 498) S4EC-(DEC)$_{16}$ code.* Source: *[UMAN02b]. © 2002 IEICE Japan.*

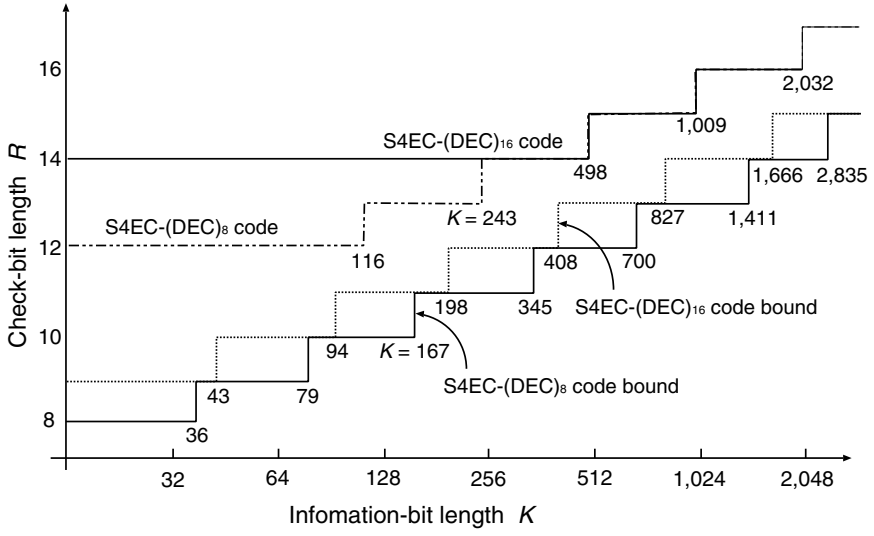


Figure 6.26 *Check-bit lengths compared with information-bit lengths of the S4EC-(DEC)$_B$ codes for $B = 8$ and 16 bits.* Source: *[UMAN02b]. © IEICE Japan.*

TABLE 6.6 Error Detection Capabilities of (270, 256) S4EC-(DEC)$_{16}$ Code (Code1) and (512, 498) S4EC-(DEC)$_{16}$ Code (Code2)

Errors	Code1 (%)	Code2 (%)
Triple-bit error	87.58	75.44
Bit plus byte error	75.98	51.70
Double-byte error	78.27	56.56
Double-byte within a block error	88.13	79.56
Block error	94.84	90.48

Source: [UMAN02b]. © 2002 IEICE Japan.

TABLE 6.7 Decoder Gate Count for S4EC-(DEC)$_{16}$ Code

Circuits	$K = 64$ bits	$K = 128$ bits	$K = 256$ bits
Syndrome generator	748	1,482	3,010
Syndrome decoder	617	867	1,367
Error corrector	117	213	405
Total	1,482	2,562	4,782

Source: [UMAN02b]. © 2002 IEICE Japan.

($K \geq 198$), the check-bit length of the S4EC-(DEC)$_{16}$ code has either 1 or 2 extra bits than the bound of S4EC-(DEC)$_{16}$ codes. For most of the practical cases where information-bit length $K = 64$, 128, or 256, however, the S4EC-(DEC)$_{16}$ code requires 14 (< 16) check bits, thus making it possible to dedicate one chip for check bits.

Table 6.6 shows the error detection capabilities of the (270, 256) S4EC-(DEC)$_{16}$ and (512, 498) S4EC-(DEC)$_{16}$ codes for five types of errors such as random triple-bit errors, bit plus byte errors, double-byte errors, double-byte within a block errors, and block errors. The (270, 256) S4EC-(DEC)$_{16}$ code considered here is a shortened code of the (512, 498) S4EC-(DEC)$_{16}$ code obtained by deleting the last 242 binary columns of the original parity-check matrix. The decoder hardware complexity of the S4EC-(DEC)$_{16}$ code is shown in Table 6.7 for the practical information lengths, such as 64, 128, and 256 bits. In this table a four-input AND / OR gate counts as one gate and a two-input exclusive-OR gate as 1.5 gates.

6.4 SINGLE-BYTE ERROR CORRECTING AND SINGLE-BYTE PLUS SINGLE-BIT ERROR DETECTING (S*b*EC-(S*b* + S)ED) CODES

Since the vast majority of the errors in the byte-organized semiconductor memory systems are random bit errors, which may be caused by α particles, cell failures, or external noises, it is more likely that a random bit error occurs lined up in a codeword with another existing byte error due to a chip failure than that as many as two chips fail to yield a double-byte error. We refer to such an error (i.e., an error that corrupts both one byte and one bit in another byte) as a *single-byte plus single-bit error*. In other words, single-byte plus single-bit errors are a type of double-byte error where at least one of the two byte errors has Hamming weight one. The codes that control this type of error are presented in [HAMA93, HAMA97, DUNN94, CHEN98]. Figure 6.27 shows examples of single-byte plus single-bit errors as well as some other types of errors that occur in byte-organized memory systems.

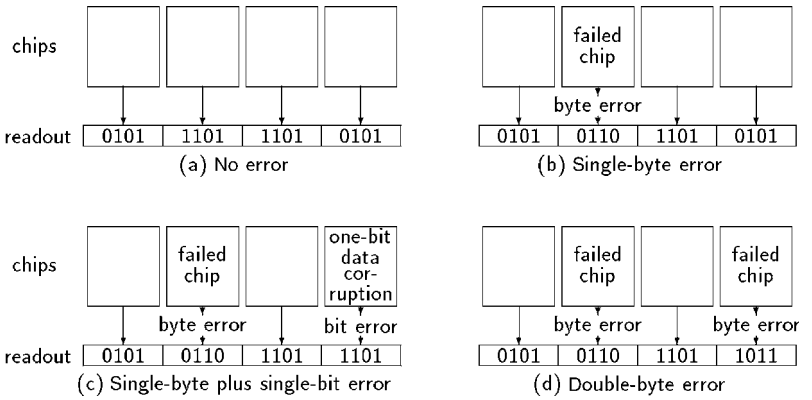


Figure 6.27 *Examples of errors in byte-organized memory systems.* Source: *[HAMA97]. © 1997 IEEE.*

SbEC-DbED codes can detect any double-byte errors that are caused by two failed chips, such as the one shown in Figure 6.27 (d), which in fact occur less often than single-bit plus single-byte errors, such as the one given in Figure 6.27 (c). Because double-byte errors do not occur unless as many as two chips fail catastrophically, it is usually sufficient for memory systems to provide against single-bit plus single-byte errors rather than protect themselves for all the double-byte errors. For this reason the SbEC-(Sb + S)ED codes have been applied to the main storage of the server systems [DOET97, SPAI99].

Therefore the discussion of this section covers a new class of linear codes, called *single b-bit byte error correcting and single b-bit byte plus single-bit error detecting codes*, or SbEC-(Sb + S)ED codes. This class of codes can correct all single-byte errors and detect any error that corrupts both one byte and one bit in another byte. Suppose that an SbEC-(Sb + S)ED code is employed in a system. If a chip failure occurs, the code can correct the single-byte error caused by the failure. If an α particle or an external noise should induce a soft error in addition to the single-byte hard error, the code can detect these errors without miscorrection.

6.4.1 Code Conditions and Bounds

Three lower bounds on the check-bit length of a linear SbEC-(Sb+S)ED code are given in this subsection [HAMA97].

As a class of single b-bit byte error correcting codes, the SbEC-(Sb+S)ED codes can detect any double-byte error such that at least one of the two byte errors has Hamming weight one. The next theorem follows directly from the definition of this class of linear codes.

Theorem 6.33 *A linear (N, K) code with parity-check matrix* $\boldsymbol{H} = [\boldsymbol{H}_0\ \boldsymbol{H}_1\ \ldots\ \boldsymbol{H}_{n-1}]$ *is an SbEC-(Sb+S)ED code if and only if*

1. $E_1 \cdot \boldsymbol{H}_i^T \neq E_2 \cdot \boldsymbol{H}_j^T$ *for all* $i,\ j \in \{0,\ 1,\ \ldots,\ n-1\}\ (i \neq j)$, *and for all* $E_1, E_2 \in GF(2^b),\ [E_1\ E_2] \neq \boldsymbol{o}$,
2. $E_1 \cdot \boldsymbol{H}_i^T \neq E_2 \cdot \boldsymbol{H}_j^T + E_3 \cdot \boldsymbol{H}_k^T$ *for all* $i,\ j,\ k \in \{0,\ 1,\ \ldots,\ n-1\}\ (i \neq j \neq k \neq i)$, *and for all* $E_1, E_2, E_3 \in GF(2^b)$, *where* E_3 *has Hamming weight one.*

Here $\boldsymbol{H}_i,\ i = 0,\ 1,\ \ldots,\ n-1$, *is the submatrix corresponding to the i-th byte.*

This theorem can be easily proved, and therefore the proof is omitted.

Theorem 6.34 *A linear SbEC-(Sb + S)ED code needs at least* $2b + 1$ *check bits.*

Another lower bound for the check-bit length of an SbEC-(Sb + S)ED code can be obtained by the *puncturing technique* [MCWI77]. Two lemmas explain this technique with regard to single-bit error correcting and single b-bit byte error detecting (SEC-SbED) codes.

Lemma 6.9 *Puncturing a linear (N, K) SbEC-(Sb + S)ED code by one byte gives a linear (N−b, K) SEC-SbED code.*

Proof Let $\boldsymbol{C}$ be a linear SbEC-(Sb + S)ED code and $\boldsymbol{C}'$ be the punctured linear code that can be obtained by deleting the i-th byte of each codeword in $\boldsymbol{C}$. From the first condition in Theorem 6.33, a single-byte error cannot be a codeword in $\boldsymbol{C}'$, and neither can a single-byte plus single-bit error from the second condition in the theorem. These imply that no pair of distinct single-bit errors can result in the same syndrome and that the syndrome of any single-byte error is different from that of any single-bit error occurring in another byte. Therefore $\boldsymbol{C}'$ can correct any single-bit errors and detect any single-byte errors. Q.E.D.

Lemma 6.10 *A linear (N, K) SEC-SbED code must satisfy the following inequality*:

$$N \leq 2^{N-K} - 2^b + b.$$

Proof All the $2^b - 1$ byte errors in a fixed position and the $N - b$ single-bit errors in the remaining positions must result in nonzero syndromes that differ from one another. From this it follows that

$$(2^b - 1) + (N - b) \leq 2^{N-K} - 1.$$

Hence the inequality in the theorem holds. Q.E.D.

From the two lemmas, we can say that if a linear (N, K) SbEC-(Sb + S)ED code exists, so does a linear ($N - b$, K) SEC-SbED code, which is written as

$$N - b \leq 2^{N-b-K} - 2^b + b,$$

and is the bound described by the next theorem.

Theorem 6.35 *A linear (N, K) SbEC-(Sb + S)ED code must satisfy the next inequality*

$$R = N - K \geq b + \lceil log_2(N - 2b + 2^b) \rceil,$$

where $\lceil x \rceil$ *gives the smallest integer greater than or equal to* x.

Another bound can be obtained as follows.

Theorem 6.36 *If a linear (N, K) SbEC-(Sb+S)ED code exists, the following inequality holds*:

$$R = N - K \geq \left\lceil log_2 \left[\frac{(b+1)(2^b - 1)}{b} N - b(2^b - 1) + 1 \right] \right\rceil.$$

Proof The syndromes of any single-byte errors and those of single-bit plus single-byte errors that corrupt both the first byte in the codewords and one bit in another byte are all different from one another and not equal to the zeros vector. Consequently

$$(2^b - 1)\frac{N}{b} + (2^b - 1)(N - b) \leq 2^R - 1.$$

That is to say, the inequality in the theorem holds. Q.E.D.

In the case where $b = 1$, these bounds are the same as those for SEC-DED codes, which are a natural consequence of the definition of SbEC-(Sb+S)ED codes. In this case, Theorem 6.34 states that a linear SEC-DED code must have at least three check bits, which is the Singleton bound [SING64, MCWI77] for SEC-DED codes. The inequalities in Theorems 6.35 and 6.36 can both be written as $N \leq 2^{R-1}$, which is the Hamming bound [MCWI77]. Roughly speaking, the bound in Theorem 6.35 is tighter than the one in Theorem 6.36 when N is relatively small, and vice versa.

6.4.2 Design for SbEC-(Sb + S)ED Codes

Two code design methods of SbEC-(Sb + S)ED codes are presented here [HAMA97]. The first one derives codes of various byte lengths and code lengths. The second method provides more efficient codes than the first, but lacks flexibility for code parameters. That is, the second method allows the byte length to have an even integer not smaller than 4, and then the code length is determined uniquely by the byte length.

Design Method I The following procedure derives the SbEC-(Sb + S)ED codes from Sb'EC codes, where $b' = b - 1$:

Step 1. Let $\mathbf{H}' = [\mathbf{H}'_0\ \mathbf{H}'_1\ \ldots\ \mathbf{H}'_{n-1}]$ denote the $R' \times N'$ parity-check matrix of an Sb'EC code where R', b' and n are positive integers, $N' = nb'$, and $\mathbf{H}'_i$, $i = 0,\ 1,\ \ldots,\ n-1$, is the submatrix corresponding to the i-th byte. If $b' = 1$, regard the Sb'EC code as a simple SEC code.

Step 2. Transform $\mathbf{H}' = [\mathbf{H}'_0\ \mathbf{H}'_1\ \ldots\ \mathbf{H}'_{n-1}]$ into an $R' \times nb$ matrix $\hat{\mathbf{H}} = [\hat{\mathbf{H}}_0\ \hat{\mathbf{H}}_1\ \ldots\ \hat{\mathbf{H}}_{n-1}]$, widening each $\mathbf{H}'_i$ by one bit in the following way: Let f_i be the sum of an even number of column vectors in $\mathbf{H}'_i$ for $i = 0,\ 1,\ \ldots,\ n-1$. It is possible for f_i to be the zero vector. Annex each f_i to $\mathbf{H}'_i$ to make $\hat{\mathbf{H}}_i = [\mathbf{H}'_i\ f_i]$ so that an $R' \times nb$ matrix $\hat{\mathbf{H}} = [\hat{\mathbf{H}}_0\ \hat{\mathbf{H}}_1\ \ldots\ \hat{\mathbf{H}}_{n-1}]$ is obtained.

Step 3. Let $u_0,\ u_1,\ \ldots,\ u_{n-1} \in GF(2^{R''})$ be odd-weight-column vectors such that $u_i \neq u_j$ for $i \neq j$. Let an $R'' \times b$ matrix $\mathbf{U}_i = [u_i\ u_i\ \cdots\ u_i]$ denote the collection of b vectors u_i's for $i = 0,\ 1,\ \ldots,\ n-1$. Prepare $\mathbf{U} = [\mathbf{U}_0\ \mathbf{U}_1\ \ldots\ \mathbf{U}_{n-1}]$ consisting of those $\mathbf{U}_i$'s.

Step 4. In the final matrix note that the null space of the following matrix is an SbEC-(Sb + S)ED code of byte length $b = b' + 1$, code length $N = nb$ and check-bit length $R = R' + R''$:

$$\begin{bmatrix} \hat{\mathbf{H}} \\ \mathbf{U} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{H}}_0 & \hat{\mathbf{H}}_1 & \cdots & \hat{\mathbf{H}}_{n-1} \\ \mathbf{U}_0 & \mathbf{U}_1 & \cdots & \mathbf{U}_{n-1} \end{bmatrix}.$$

Theorem 6.37 *A code obtained with the procedure above is an SbEC-(Sb + S)ED code.*

Proof Let $\mathbf{H} = [\mathbf{H}_0\ \mathbf{H}_1\ \ldots\ \mathbf{H}_{n-1}]$ be the parity-check matrix of the code and $\mathbf{V}_i$ denote the space spanned by the b column vectors in $\mathbf{H}_i$ for $i = 0, 1, \cdots, n-1$. For the code to be SbEC, it is necessary and sufficient that $\mathbf{V}_i \cap \mathbf{V}_j = \{\mathbf{o}\}$ for $i \neq j$, and $\mathbf{V}_i$ has dimension b for each i. For $i \neq j$, the space spanned by the column vectors in $\hat{\mathbf{H}}_i$ and the one spanned by those of $\hat{\mathbf{H}}_j$ have no vector in common other than $\mathbf{o}$, since $\mathbf{H}'$ is the parity-check matrix of an Sb'EC code. The space spanned by the column vectors in $\mathbf{U}_i$ and the one spanned by those of $\mathbf{U}_j$ have no vector in common other than $\mathbf{o}$ for $i \neq j$, either. Hence it follows that $\mathbf{V}_i \cap \mathbf{V}_j = \{\mathbf{o}\}$ for $i \neq j,\ i,\ j \in \{0, 1, \cdots, n-1\}$.

To prove that $\mathbf{V}_i$ has dimension b for $i = 0,\ 1,\ \ldots,\ n-1$, we have only to show that the first $b-1$ column vectors in $\mathbf{H}_i$ are linearly independent and the last column vector cannot be a linear combination of the first $b-1$ column vectors. From the upper part $\hat{\mathbf{H}}_i$ of $\mathbf{H}_i$, we see that the first $b-1$ column vectors in $\hat{\mathbf{H}}_i$ are linearly independent and the last column vector is a linear combination of the even number of vectors taken from the first $b-1$ columns in $\hat{\mathbf{H}}_i$. On the other hand, from the lower part $\hat{\mathbf{U}}_i$ of $\mathbf{H}_i$, we see that the last column vector in $\mathbf{U}_i$ cannot be a linear combination of the even number of vectors taken from $\mathbf{U}_i$, which must be $\mathbf{o}$. This implies that the b column vectors in $\mathbf{H}_i$ are linearly independent. Consequently the code has the capability of SbEC.

Next we will show that the code is capable of detecting any single-bit plus single-byte error owing to the lower part $\mathbf{U}$ of the parity-check matrix. Suppose that for some $i,\ j,\ k \in \{0,\ 1,\ \ldots,\ n-1\} (i \neq j \neq k \neq i)$ and $E_1,\ E_2,\ E_3 \in GF(2^b)$, E_3 has Hamming weight one and

$$E_1 \cdot \mathbf{H}_i^T + E_2 \cdot \mathbf{H}_j^T + E_3 \cdot \mathbf{H}_k^T = \mathbf{o}. \tag{6.21}$$

Then each codeword in the code must have even weight, since each column in $\mathbf{U}$ has odd weight. Therefore, with E_3 having odd weight, either E_1 or E_2 is of odd weight and the other is of even weight. If E_1 has odd weight, however, it follows that $u_i = u_k$ from Eq. (6.21), which contradicts the assumption. The case where E_2 has odd weight also leads to a contradiction. Consequently the code is an SbEC-(Sb + S)ED code. Q.E.D.

Example 6.16 [HAMA97]

The following is the parity-check matrix of an Sb'EC code with $b' = 2$:

$$\mathbf{H}' = \begin{bmatrix} 10 & 10 & 10 & 10 \\ 01 & 01 & 01 & 01 \\ 10 & 01 & 11 & 00 \\ 01 & 11 & 10 & 00 \end{bmatrix}.$$

From this matrix the parity-check matrix of an S3EC-(S3 + S)ED code can be obtained as follows:

$$\mathbf{H} = \left[\begin{array}{cccc} 100 & 100 & 100 & 100 \\ 010 & 010 & 010 & 010 \\ 100 & 010 & 110 & 000 \\ 010 & 110 & 100 & 000 \\ \hline 111 & 000 & 111 & 000 \\ 000 & 111 & 111 & 000 \\ 000 & 000 & 111 & 111 \end{array}\right].$$

Given byte length b (bits) and check-bit length R, the maximum code length in bits of an SbEC-(Sb + S)ED code derived from the maximal Sb'EC code [HONG72] previously known in Subsection 5.1.4 is as follows:

$$N = \begin{cases} b \cdot 2^{R-2b+1} & \text{for } 2b+1 \le R < 3b-2, \\ b \cdot 2^{(R-b)/2} & \text{for } 3b-2 \le R,\ R-b : even, \\ b \cdot 2^{(b-1)+\gamma}\{2^{(b-1)\tau}-1\}/(2^{b-1}-1)+b & \text{for } 3b-2 \le R,\ R-b : odd, \end{cases}$$

where τ and γ are the integers such that $(R-b-1)/2 = \tau(b-1)+\gamma$ and $0 \le \gamma < b-1$.

Design Method II The following code design uses natures of finite fields, their subfields, and minimal polynomials over the subfields.

Theorem 6.38 *Let l be an integer greater than 1, and b equal $2l$. Consider a matrix*

$$\boldsymbol{H} = [\boldsymbol{H}_1\ \boldsymbol{H}_2\ \cdots\ \boldsymbol{H}_n]$$

composed of the following matrices:

$$\boldsymbol{H}_i = \begin{bmatrix} \alpha^i\gamma_{i,1} & \alpha^i\gamma_{i,2} & \cdots & \alpha^i\gamma_{i,b} \\ \alpha^{-i}\delta_{i,1} & \alpha^{-i}\delta_{i,2} & \cdots & \alpha^{-i}\delta_{i,b} \end{bmatrix} \qquad (i = 1, 2, \cdots, n),$$

where α is a primitive element in $GF(2^{3l})$, $n = (2^{3l}-1)/(2^l-1) = 2^{2l}+2^l+1$ and $\gamma_{i,1}, \gamma_{i,2}, \cdots, \gamma_{i,b}, \delta_{i,1}, \delta_{i,2}, \cdots, \delta_{i,b}$ are nonzero elements that belong to a subfield $GF(2^l) = \{0, 1, \alpha^n, \alpha^{2n}, \cdots, \alpha^{(2^l-2)n}\}$ of $GF(2^{3l})$ for $i = 1, 2, \cdots, n$. All the elements that appear in the matrices are expressed as $3l$-tuples over $GF(2)$. Suppose that the $3b \times b$ matrix $\boldsymbol{H}_i$ over $GF(2)$ has rank b for each i. Then the code with the parity-check matrix $\boldsymbol{H}$ is an SbEC-(Sb + S)ED code.

Proof Let $\mathbf{V}_i$ denote the space spanned by the b column vectors in $\mathbf{H}_i$, $i = 1, 2, \cdots, n$. All we have to show is that the space $\mathbf{V}_i$ has dimension b for each i and that $\mathbf{V}_i \cap \mathbf{V}_j = \{\mathbf{o}\}$ for $i \ne j$ to prove that the code can correct any single b-bit byte errors. From the nature of the

finite fields, it is known that $\beta = \alpha^n$ is a primitive element of a subfield $GF(2^l)$ [MCWI77]. Consequently $GF(2^l)^* = GF(2^l) - \{0\} = \{1, \beta, \beta^2, \cdots, \beta^{2^l-2}\}$ makes up a normal subgroup of the multiplicative group $GF(2^{3l})^* = GF(2^{3l}) - \{0\}$ so that a quotient group $GF(2^{3l})^*/GF(2^l)^* = \{\alpha^i GF(2^l)^* \mid i = 1, 2, \cdots, n\}$ can be obtained. Note that the elements of $GF(2^{3l})$ that appear in $\mathbf{H}_i$ are taken from $\alpha^i GF(2^l)^* = \{\alpha^i, \alpha^i\beta, \alpha^i\beta^2, \cdots, \alpha^i\beta^{2^l-2}\}$. Since no pair of members $\alpha^i GF(2^l)^*$ and $\alpha^j GF(2^l)^*$ in $GF(2^{3l})^*/GF(2^l)^*$, $i \neq j$, have any elements in common, it follows that $\mathbf{V}_i \cap \mathbf{V}_j = \{\mathbf{o}\}$ for $i \neq j$, $i, j \in \{1, 2, \cdots, n\}$. Thus, from the assumption that $\mathbf{H}_i$'s have rank b, it follows that the code is SbEC.

Next, we will show the detecting capability of the code. Suppose that there exist $i, j, k \in \{1, 2, \cdots, n\} (i \neq j \neq k \neq i)$ and $E_1, E_2, E_3 \in GF(2^b)$ such that E_3 has Hamming weight one and

$$E_1 \cdot \mathbf{H}_i^T + E_2 \cdot \mathbf{H}_j^T + E_3 \cdot \mathbf{H}_k^T = \mathbf{o}. \tag{6.22}$$

Then, for some elements x, y, z, u, v, w in the subfield $GF(2^l)$,

$$E_1 \cdot \mathbf{H}_i^T = \begin{bmatrix} \alpha^i x \\ \alpha^{-i} u \end{bmatrix}, \quad E_2 \cdot \mathbf{H}_j^T = \begin{bmatrix} \alpha^j y \\ \alpha^{-j} v \end{bmatrix}, \quad E_3 \cdot \mathbf{H}_k^T = \begin{bmatrix} \alpha^k z \\ \alpha^{-k} w \end{bmatrix}.$$

Therefore Eq. (6.22) can be expressed as

$$\begin{cases} \alpha^i x + \alpha^j y + \alpha^k z = 0, \\ \alpha^{-i} u + \alpha^{-j} v + \alpha^{-k} w = 0. \end{cases} \tag{6.23}$$

Note that $z, w \neq 0$, since E_3 has Hamming weight one. From Eq. (6.23) we obtain

$$(\alpha^i x + \alpha^j y)(\alpha^{-i} u + \alpha^{-j} v) = zw, \tag{6.24}$$

that is,

$$a_2 \alpha^{2(j-i)} + a_1 \alpha^{j-i} + a_0 = 0, \tag{6.25}$$

where $a_2 = yu, a_1 = xu + yv + zw$, and $a_0 = xv$. Since $a_2, a_1, a_0 \in GF(2^l)$, Eq. (6.25) implies that α^{j-i} is a root of the next equation over $GF(2^l)$ whose degree is less than or equal to 2:

$$a_2 X^2 + a_1 X + a_0 = 0. \tag{6.26}$$

According to the theory of finite fields, the degree of the *minimal polynomial* over $GF(2^l)$ of an element in an extension field $GF(2^{lm})$, where m is a positive integer, must be a divisor of m [MCWI77]. Therefore no element in $GF(2^{3l})$ has a minimal polynomial of degree 2 over $GF(2^l)$. Hence in Eq. (6.26) either the coefficient $a_0 = xv$ or $a_2 = yu$ must equal zero. If $y = 0$, however, the first equation in Eq. (6.23) implies $\alpha^i x = \alpha^k z$, which contradicts the

fact $\alpha^i GF(2^l) \cap \alpha^k GF(2^l)^* = \emptyset$ $(i \neq k)$. Letting $x = 0$, $v = 0$ or $u = 0$ leads to a contradiction in the same way. Clearly, α^{j-i} must have a minimal polynomial of degree 1. However, 2^l elements $0,\ 1,\ \alpha^n,\ \alpha^{2n},\ \ldots,\ \alpha^{(2^l-2)n}$, and no others have such minimal polynomials in $GF(2^{3l})$, which contradicts the fact $0 < |i-j| < n$. Therefore, the code has the detecting capability. Finally, we conclude that the code specified in the theorem is an SbEC-(Sb + S)ED code. Q.E.D.

In the theorem we have assumed that $\mathbf{H}_i$ has rank $b = 2l$ for $i = 1, 2, \cdots, n$; that is, any $b = 2l$ columns of $\mathbf{H}_i$ are linearly independent. It is apparent that we can make each $\mathbf{H}_i$ have rank b in any case, since $\alpha^i GF(2^l) = \{0,\ \alpha^i,\ \alpha^i\beta,\ \alpha^i\beta^2,\ \ldots,\ \alpha^i\beta^{2^l-2}\}$ is a linear space over $GF(2)$ of dimension l for $i = 1,\ 2,\ \ldots,\ n$. For instance, provided $\beta = \alpha^n$,

$$\mathbf{H}_i = \begin{bmatrix} \alpha^i(1+\beta) & \alpha^i & \alpha^i & \cdots & \alpha^i & \alpha^i & \alpha^i\beta & \alpha^i\beta^2 & \cdots & \alpha^i\beta^{l-1} \\ \alpha^{-i}\beta & \alpha^{-i}(1+\beta) & \alpha^{-i}(1+\beta^2) & \cdots & \alpha^{-i}(1+\beta^{l-1}) & \alpha^{-i} & \alpha^{-i}\beta & \alpha^{-i}\beta^2 & \cdots & \alpha^{-i}\beta^{l-1} \end{bmatrix}$$

$$(i = 1, 2, \cdots, n)$$

are such matrices.

Example 6.17 [HAMA97]

The theorem yields the following parity-check matrix of an (84, 72) S4EC-(S4 + S)ED code when $l = 2$:

$$\mathbf{H} = [\mathbf{H}_1\ \mathbf{H}_2\ \cdots\ \mathbf{H}_n],$$

where $n = 21$, α is a root of a primitive polynomial $\mathbf{g}(x) = x^6 + x + 1$ over $GF(2)$, and

$$\mathbf{H}_i = \begin{bmatrix} \alpha^i & \alpha^{i+n} & \alpha^{i+2n} & \alpha^i \\ \alpha^{-i+n} & \alpha^{-i} & \alpha^{-i} & \alpha^{-i+2n} \end{bmatrix} \qquad (i = 1,\ 2,\ \ldots,\ n).$$

A binary expression for this matrix can be found in Figure 6.28 (a). Deleting 2 bytes of the code yields a practical (76, 64) S4EC-(S4 + S)ED code. In this case the 13-th and the 17-th bytes are deleted. Then the systematic form of this code determined by row operations is presented in Figure 6.28 (b).

6.4.3 Evaluation

Figure 6.29 shows the check-bit lengths of the most efficient SbEC-(Sb + S)ED codes, where $b = 4$ bits, which can be obtained with the design methods given in Subsection 6.4.2, as well as the tightest of those bounds on check-bit length mentioned in Subsection 6.4.1 and the check-bit lengths of the most efficient SbEC-DbED codes known [CHEN92]. For any byte length $b \geq 2$, design method I provides

$$
\left[\begin{array}{cccccccccccccccccccccc}
1101 & 0110 & 0110 & 0110 & 0000 & 0110 & 1011 & 0000 & 0000 & 0110 & 0110 & 1101 & 1011 & 0000 & 0110 & 0000 & 1011 & 0110 & 1011 & 0110 & 0110 \\
0110 & 1011 & 0000 & 0000 & 0110 & 0110 & 1101 & 1011 & 0000 & 0110 & 0000 & 1011 & 0110 & 1011 & 0110 & 0110 & 1011 & 1101 & 1101 & 1101 & 0000 \\
0000 & 0110 & 1011 & 0000 & 0000 & 0110 & 0110 & 1101 & 1011 & 0000 & 0110 & 0000 & 1011 & 0110 & 1011 & 0110 & 0110 & 1011 & 1101 & 1101 & 1101 \\
0110 & 0000 & 0110 & 1011 & 0000 & 0000 & 0110 & 0110 & 1101 & 1011 & 0000 & 0110 & 0000 & 1011 & 0110 & 1011 & 0110 & 0110 & 1011 & 1101 & 1101 \\
0110 & 0110 & 0000 & 0110 & 1011 & 0000 & 0000 & 0110 & 0110 & 1101 & 1011 & 0000 & 0110 & 0000 & 1011 & 0110 & 1011 & 0110 & 0110 & 1011 & 1101 \\
0110 & 0110 & 0110 & 0000 & 0110 & 1011 & 0000 & 0000 & 0110 & 0110 & 1101 & 1011 & 0000 & 0110 & 0000 & 1011 & 0110 & 1011 & 0110 & 0110 & 1011 \\
\hline
1110 & 0111 & 0111 & 1110 & 0111 & 1110 & 0000 & 0111 & 0000 & 1110 & 1001 & 0111 & 0111 & 0000 & 0000 & 1110 & 0111 & 0000 & 0111 & 0111 & 0111 \\
1001 & 0000 & 1001 & 1001 & 1001 & 1110 & 0111 & 0111 & 1110 & 0111 & 1110 & 0000 & 0111 & 0000 & 1110 & 1001 & 0111 & 0111 & 0000 & 0000 & 1110 \\
0000 & 1001 & 1001 & 1001 & 1110 & 0111 & 0111 & 1110 & 0111 & 1110 & 0000 & 0111 & 0000 & 1110 & 1001 & 0111 & 0111 & 0000 & 0000 & 1110 & 0111 \\
1001 & 1001 & 1001 & 1110 & 0111 & 0111 & 1110 & 0111 & 1110 & 0000 & 0111 & 0000 & 1110 & 1001 & 0111 & 0111 & 0000 & 0000 & 1110 & 0111 & 0000 \\
1001 & 1001 & 1110 & 0111 & 0111 & 1110 & 0111 & 1110 & 0000 & 0111 & 0000 & 1110 & 1001 & 0111 & 0111 & 0000 & 0000 & 1110 & 0111 & 0000 & 0111 \\
1001 & 1110 & 0111 & 0111 & 1110 & 0111 & 1110 & 0000 & 0111 & 0000 & 1110 & 1001 & 0111 & 0111 & 0000 & 0000 & 1110 & 0111 & 0000 & 0111 & 0111
\end{array}\right]
$$

(a) (84, 72) S4EC-(S4 + S)ED code

$$
\left[\begin{array}{cccccccccccccccccccc}
1000 & 0001 & 1101 & 1100 & 0111 & 0001 & 1001 & 0101 & 1000 & 0010 & 1000 & 1111 & 0011 & 1101 & 0100 & 0011 & 1000 & 0000 & 0000 \\
0100 & 0011 & 1011 & 1000 & 1110 & 0011 & 0111 & 1111 & 0100 & 0001 & 0100 & 1010 & 0010 & 1011 & 1100 & 0010 & 0100 & 0000 & 0000 \\
0010 & 0101 & 0110 & 0011 & 1110 & 0101 & 0111 & 0100 & 0010 & 1010 & 0010 & 1100 & 1111 & 0110 & 0001 & 1111 & 0010 & 0000 & 0000 \\
0001 & 1111 & 1101 & 0010 & 1001 & 1111 & 1110 & 1100 & 0001 & 0101 & 0001 & 1000 & 1010 & 1101 & 0011 & 1010 & 0001 & 0000 & 0000 \\
0011 & 0010 & 1010 & 1100 & 1011 & 1101 & 1000 & 0001 & 0110 & 0111 & 0001 & 1100 & 1110 & 1000 & 1111 & 0010 & 0000 & 1000 & 0000 \\
0010 & 0001 & 0101 & 1000 & 0110 & 1011 & 0100 & 0011 & 1101 & 1110 & 0011 & 1000 & 1001 & 0100 & 1010 & 0001 & 0000 & 0100 & 0000 \\
1111 & 1010 & 1000 & 0011 & 1101 & 0110 & 0010 & 0101 & 1011 & 1110 & 0101 & 0011 & 1001 & 0010 & 1100 & 1010 & 0000 & 0010 & 0000 \\
1010 & 0101 & 0100 & 0010 & 1011 & 1101 & 0001 & 1111 & 0110 & 1001 & 1111 & 0010 & 0111 & 0001 & 1000 & 0101 & 0000 & 0001 & 0000 \\
0010 & 0110 & 0100 & 1001 & 0010 & 1110 & 1010 & 1100 & 0011 & 1100 & 0101 & 1101 & 0110 & 1000 & 1001 & 1100 & 0000 & 0000 & 1000 \\
0001 & 1101 & 1100 & 0111 & 0001 & 1001 & 0101 & 1000 & 0010 & 1000 & 1111 & 1011 & 1101 & 0100 & 0111 & 1000 & 0000 & 0000 & 0100 \\
1010 & 1011 & 0001 & 0111 & 1010 & 1001 & 1000 & 0011 & 1111 & 0011 & 0100 & 0110 & 1011 & 0010 & 0111 & 0011 & 0000 & 0000 & 0010 \\
0101 & 0110 & 0011 & 1110 & 0101 & 0111 & 0100 & 0010 & 1010 & 0010 & 1100 & 1101 & 0110 & 0001 & 1110 & 0010 & 0000 & 0000 & 0001
\end{array}\right]
$$

(b) Systematic (76, 64) S4EC-(S4 + S)ED code

Figure 6.28 *Parity-check matrices of S4EC-(S4 + S)ED codes.* Source of part(a): *[HAMA97]. © 1997 IEEE.*

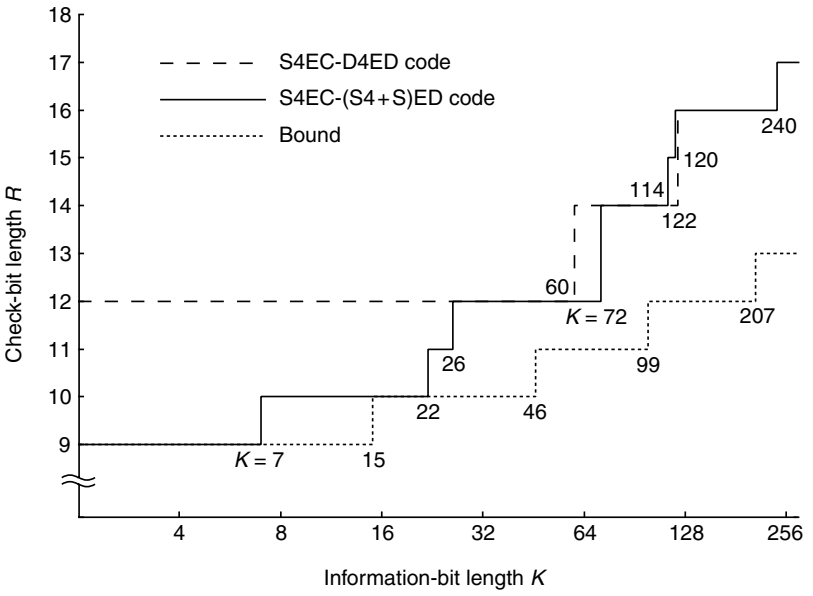


Figure 6.29 *Check-bit lengths compared with information-bit lengths of the S4EC-(S4+S)ED codes.* Source: *[HAMA97]. © 1997 IEEE.*

SbEC-(Sb + S)ED codes that meet the bound given in Theorem 6.34, namely codes with $2b + 1$ check bits. The SbEC-DbED codes can never achieve this check-bit length, since an SbEC-DbED code requires at least $3b$ check bits regardless of its code rate from the Singleton bound [SING64, MCWI77]. When b is an even integer not less than 4, design method II provides the codes with $b(2^b + 2^{b/2} - 2)$ information bits and $3b$ check bits, while it is known that the information-bit lengths of the SbEC-DbED codes with $3b$ check bits are at most $b(2^b - 1) < b(2^b + 2^{b/2} - 2)$ [CHEN86].

Particularly, in the practical case where byte length $b = 4$ bits and information-bit length $K = 64$, design method II gives the SbEC-(Sb + S)ED codes of check-bit length $R = 12$, which is same as that of the Hamming SbEC codes with $b = 4$ bits and $K = 64$ bits. In contrast, the previously known SbEC-DbED codes require at least 14 check bits [CHEN92].

Figure 6.30 shows the parity-check matrix of another systematic (76, 64) S4EC-(S4 + S)ED code [CHEN98]. The error detection capabilities of the codes shown in Figure 6.28 (b) and Figure 6.30 are as Follow:

(76, 64) S4EC-(S4 + S)ED codes	Double-byte errors (%)	Triple-bit errors (%)	**H** matrix weight
Code shown in Figure 6.28 (b)	92.84	93.97	394
Code shown in Figure 6.30	92.84	93.85	354

$$
\mathbf{H} = \left[\begin{array}{c:c:c:c:c:c:c:c:c:c:c:c:c:c:c:c:c:c:c}
1000 & 1000 & 1000 & 1000 & 1000 & 1000 & 1000 & 1000 & 1001 & 1101 & 1111 & 0011 & 1100 & 1100 & 1110 & 0110 & 1000 & 0000 & 0000 \\
0100 & 0100 & 0100 & 0100 & 0100 & 0100 & 0100 & 0100 & 0111 & 1011 & 1010 & 0010 & 1000 & 1000 & 1001 & 1101 & 0100 & 0000 & 0000 \\
0010 & 0010 & 0010 & 0010 & 0010 & 0010 & 0010 & 0010 & 0111 & 0110 & 1100 & 1111 & 0011 & 0011 & 1001 & 1011 & 0010 & 0000 & 0000 \\
0001 & 0001 & 0001 & 0001 & 0001 & 0001 & 0001 & 0001 & 1110 & 1101 & 1000 & 1010 & 0010 & 0010 & 0111 & 0110 & 0001 & 0000 & 0000 \\
\hline
0110 & 1000 & 0010 & 1011 & 1010 & 1101 & 0101 & 1110 & 1000 & 0001 & 1000 & 0111 & 1000 & 1110 & 1000 & 0100 & 0000 & 1000 & 0000 \\
1101 & 0100 & 0001 & 0110 & 0101 & 1011 & 1111 & 1001 & 0100 & 0011 & 0100 & 1110 & 0100 & 1001 & 0100 & 1100 & 0000 & 0100 & 0000 \\
1011 & 0010 & 1010 & 1101 & 1000 & 0110 & 0100 & 1001 & 0010 & 0101 & 0010 & 1110 & 0010 & 1001 & 0010 & 0001 & 0000 & 0010 & 0000 \\
0110 & 0001 & 0101 & 1011 & 0100 & 1101 & 1100 & 0111 & 0001 & 1111 & 0001 & 1001 & 0001 & 0111 & 0001 & 0011 & 0000 & 0001 & 0000 \\
\hline
1000 & 1110 & 0111 & 1010 & 1001 & 0010 & 0110 & 0001 & 0101 & 1000 & 1011 & 1000 & 0110 & 1000 & 0100 & 1000 & 0000 & 0000 & 1000 \\
0100 & 1001 & 1110 & 0101 & 0111 & 0001 & 1101 & 0011 & 1111 & 0100 & 0110 & 0100 & 1101 & 0100 & 1100 & 0100 & 0000 & 0000 & 0100 \\
0010 & 1001 & 1110 & 1000 & 0111 & 1010 & 1011 & 0101 & 0100 & 0010 & 1101 & 0010 & 1011 & 0010 & 0001 & 0010 & 0000 & 0000 & 0010 \\
0001 & 0111 & 1001 & 0100 & 1110 & 0101 & 0110 & 1111 & 1100 & 0001 & 1011 & 0001 & 0110 & 0001 & 0011 & 0001 & 0000 & 0000 & 0001
\end{array}\right]
$$

Figure 6.30 *Systematic (76, 64) S4EC-(S4+S)ED code [CHEN98].*

EXERCISES

6.1 Design four SEC-DED-BED codes with $b = 4$ bits and $K = 56$ bits by using the methods of Theorems 6.2, 6.3, 6.5, and 6.6.

6.2 Prove properties 1 through 3 of the grouping matrix **A**.

6.3 Design an SEC-DED-S4ED code with information-bit length $K = 41$ and check-bit length $R = 7$.

6.4 The following **H** matrix shows a (40, 32) SEC-DED-S4ED code [LUI78]. Explain how this code satisfies the SEC-DED-S4ED function.

$$
\mathbf{H} = \left[\begin{array}{cccc:cccc:cccc:cccc:cccc:cccc:cccc:cccc:cccc:cccc}
1&1&1&1 & 1&1&1&1 & &&& & 1&&& & 1&&& & 1&&& & 1&&& & &&& & 1&&& & &&& \\
1&&& & 1&&& & 1&1&1&1 & 1&1&1&1 & &&& & &1&& & &&& & 1&&& & &1&& & &&& \\
&1&& & &&& & 1&&& & &&& & 1&1&1&1 & 1&1&1&1 & &1&& & &1&& & &&1& & &&& \\
&&1& & &1&& & &1&& & &1&& & &&& & &&& & 1&1&1&1 & 1&1&1&1 & &&&1 & &&& \\
&&& & 1&1&1&1 & &&1& & &&1& & &1&& & 1&1&1&1 & &&1& & &&& & &&& & 1&&& \\
&&1& & &&1& & &&&1 & &&&1 & 1&1&1&1 & &&& & &&& & 1&1&1&1 & &&& & &1&& \\
1&1&1&1 & &&& & 1&1&1&1 & &&& & &&1& & &&1& & &&&1 & &&1& & &&& & &&1& \\
&&& & &&&1 & &&& & 1&1&1&1 & &&&1 & &&&1 & 1&1&1&1 & &&&1 & &&& & &&&1
\end{array}\right]
$$

6.5 Prove that the SEC-DED-SbED code corrects a single-byte error in the presence of a single-byte error pointer (i.e., erases a single-byte error).

6.6 Verify that the code shown in Figure 6.12 has the property of eight bits error detection over two bytes.

6.7 Prove Lemmas 6.2 and 6.3.

6.8 For the syndrome space of an $(N, N-R)$ SbEC-DED code, show that the following inequality holds:

$$2^R - 1 \geq \frac{N}{b}\left(2^b - 1\right) + N - b.$$

6.9 By using Theorem 6.14 design the S4EC-DED code with code parameters of $K = 114$ bits, and $R = 14$ bits.

6.10 Prove Theorem 6.14.

6.11 Prove that the **H** matrix shown in Eq. (6.17) defines the SbEC-DED code with code length $(v-1)\cdot v^{r-2}$ bytes, where v shows the size of a generating set.

6.12 Prove that the following **H** matrix shows an SbEC-DED code over $GF(2^b)$ with maximum code length in bits $N = b\cdot\{(v-1)v^{r-2} + (v-1)v^{r-t-2}\}$, where $1 \le t \le r-2$. The elements in **H** are included in a generating set with size v containing the identity element **I** in $GF(2^b)$.

$$
\mathbf{H} = \left[\begin{array}{c|c}
 & \begin{array}{cccc} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & \vdots & \cdots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{array} \\
\mathbf{H}_r & \\ \cline{2-2}
 & \mathbf{H}_{r-t}
\end{array}\right]
\begin{array}{c} \updownarrow t \\ \\ \updownarrow r-t \end{array}
$$

$$
= \begin{array}{c}
\begin{array}{cccccccc|cccccc}
\cdots & \cdots & i & \cdots & j & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & k & \cdots & \cdots
\end{array}\\
\left[\begin{array}{ccccccc|cccccc}
\mathbf{I} & \cdots & \mathbf{I} & \cdots & \mathbf{I} & \cdots & \mathbf{I} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\
\cdots & \cdots & \mathbf{T}^{i_1} & \cdots & \mathbf{T}^{j_1} & \cdots & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\cdots & \cdots & \mathbf{T}^{i_{t-1}} & \cdots & \mathbf{T}^{j_{t-1}} & \cdots & \cdots & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \cline{8-13}
\cdots & \cdots & \mathbf{T}^{i_t} & \cdots & \mathbf{T}^{j_t} & \cdots & \cdots & \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} & \cdots & \mathbf{I} \\
\cdots & \cdots & \mathbf{T}^{i_{t+1}} & \cdots & \mathbf{T}^{j_{t+1}} & \cdots & \cdots & \cdots & \cdots & \cdots & \mathbf{T}^{k_{t+1}} & \cdots & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\cdots & \cdots & \mathbf{T}^{i_{r-1}} & \cdots & \mathbf{T}^{j_{r-1}} & \cdots & \cdots & \cdots & \cdots & \cdots & \mathbf{T}^{k_{r-1}} & \cdots & \cdots
\end{array}\right]
\begin{array}{c} \updownarrow t \\ \\ \updownarrow r-t \end{array}\\
\longleftarrow X \longrightarrow \longleftarrow Y \longrightarrow
\end{array}.
$$

6.13 Prove Theorem 6.20.

6.14 Let $\Phi = \{\mathbf{T}^{i_1}, \mathbf{T}^{i_2}, \cdots, \mathbf{T}^{i_k}\}$ be a generating set in $GF(2^b)$. We call Φ a *strong generating set* in $GF(2^b)$ if

$$
(\mathbf{T}^{i_k} + \mathbf{T}^{i_j})(\mathbf{T}^{i_m} + \mathbf{T}^{i_n})^{-1} \notin \Phi
$$

for all distinct $\mathbf{T}^{i_m}$, $\mathbf{T}^{i_n}$, $\mathbf{T}^{i_k}$, and $\mathbf{T}^{i_j}$ in Φ.

(a) Prove that any additive coset or multiplicative coset of a subfield $GF(2^A)$ of $GF(2^b)$ plus zero is a strong generating set except $GF(2^A)$ itself.

(b) Prove that if the additive cosets used in Theorems 6.18 and 6.19 are replaced by strong generating sets of size v, then the code lengths in bits corresponding to Theorems 6.18 and 6.19 are $N_1 = b \cdot (v^{r-1} + v^{r-t-1})$ and $N_2 = b \cdot v \cdot (v^{r-1} - 1)/(v - 1)$, respectively.

6.15 Prove Theorem 6.23.

6.16 In the design of the Davydov-Labinskaya code with $r = 7$, we have $N = 2^8 = 256$ bits and $R = 17$ bits. This will give us a (256, 239) S4EC-DEC code. Design the practical (144, 128) S4EC-DEC code by shortening the code to achieve a one-bit reduction in check-bit length.

6.17 Find the bound on code length of the Davydov-Labinskaya S4EC-DEC code.

6.18 With using design method I of the SbEC-(Sb + S)ED code, design the (64, 52) S4EC-(S4 + S)ED code derived from the maximal (52, 45) S3EC code shown in Subsection 5.1.4.

6.19 With using design method II of the SbEC-(Sb + S)ED code, design the (438, 420) S6EC-(S6+S)ED code.

6.20 Consider the codes capable of single-bit error correction and single-bit plus single-byte error detection (SEC-(S + Sb)ED).

(a) Find the necessary and sufficient conditions of such codes.

(b) Prove that the bound on the code length of the $(N,\ N - R)$ SEC-(S + Sb)ED code is expressed as

$$N \leq 2^{R-b} + b - 1.$$

(c) Show that the following code is an $(N,\ N - R)$ SEC-(S + Sb)ED code with maximum code length in bits $N = b \cdot 2^c + c$ and check-bit length $R = b + 2c$:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I}_b & \mathbf{I}_b & \mathbf{I}_b & \cdots & \mathbf{I}_b & \mathbf{I}_b & \mathbf{0}_{c'} \\ \mathbf{M}_0 & \mathbf{M}_1 & \mathbf{M}_2 & \cdots & \mathbf{M}_{2^c-2} & \mathbf{0}_c & \mathbf{I}_c \\ \mathbf{Q}_0 & \mathbf{Q}_1 & \mathbf{Q}_2 & \cdots & \mathbf{Q}_{2^c-2} & \mathbf{0}_c & \mathbf{0}_{c''} \end{bmatrix}.$$

Here $\mathbf{0}_c$, $\mathbf{0}_{c'}$, $\mathbf{0}_{c''}$ are $c \times b$, $b \times c$, $c \times c$ zero matrices, respectively; $\mathbf{I}_b$, $\mathbf{I}_c$ are $b \times b$, $c \times c$ identity matrices, respectively; $\mathbf{M}_i$ is a $c \times b$ matrix defined by

$$\mathbf{M}_i = \begin{bmatrix} | & | & & | \\ \alpha^i & \alpha^{i+1} & \cdots & \alpha^{i+b-1} \\ | & | & & | \end{bmatrix}, \qquad 0 \leq i \leq 2^c - 1,$$

where $\begin{vmatrix} \\ \alpha^i \\ \\ \end{vmatrix}$ is a binary coefficient vector of $x^i \ mod\ \mathbf{p}(x)$, $\mathbf{p}(x)$ being a primitive polynomial with degree c and $b \leq 2^c - 1$; and $\mathbf{Q}_j,\ 0 \leq j \leq 2^c - 2$, is a $c \times b$ matrix whose columns are b copies of the binary representation of integer j.

(d) Try to design more efficient codes than the code shown in (**c**).

6.21 Consider a code capable of adjacent double-bit error correction and single-byte error detection (ADEC-SbED) [UMAN02b].

(a) Find the necessary and sufficient conditions of such a code.

(b) Prove that a linear binary $(N, N-R)$ ADEC-SbED code exists only if

$$N \leq 2^{R-1} - 2^{b-1} + b.$$

(c) Prove that the null space of

$$\mathbf{H} = \left[\begin{array}{c|cccccc} \mathbf{I}^{\dagger} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} \\ \hline \mathbf{O} & \mathbf{Q}_0 & \mathbf{Q}_1 & \mathbf{Q}_2 & \cdots & \mathbf{Q}_{2^r-2} \end{array}\right]$$

is an ADEC-SbED code with code length in bits $N = b \cdot 2^r$ and check-bit length $R = b + r$, where the elements in $\mathbf{H}$ are defined as the following: $\mathbf{O}$ is an $r \times b$ binary zero matrix,

α, β are primitive elements of $GF(2^b)$ and $GF(2^r)(r > 2)$, respectively,

$\mathbf{I} = \left[\alpha^0\ \alpha^1\ \alpha^2\ \cdots\ \alpha^{b-1}\right]_{b\times b}$, $\mathbf{I}^{\dagger} = \left[\alpha^{b-1}\ \alpha^{b-2}\ \alpha^{b-3}\ \cdots\ \alpha^0\right]_{b\times b}$, where α^i denotes a binary column vector of $GF(2^b)$ such that the i-th coordinate is one and all other coordinates are zeros for $i = 0,\ 1,\ 2,\ \cdots, b-1$, and

$\mathbf{Q}_i = \left[\beta^i\ \beta^{i+1}\ \beta^{i+2}\ \cdots\ \beta^{i+b-2}\ \beta^i\right]_{r\times b}$, where β^i denotes a binary column vector of $GF(2^r)$ for $i = 0,\ 1,\ 2,\ \cdots,\ 2^r - 2$.

(d) Design the (128, 119) ADEC-S4ED code based on the $\mathbf{H}$ matrix presented in (**c**).

6.22 Consider the codes capable of correcting adjacent double-bit errors occurring within a b-bit byte and detecting b-bit byte errors ((ADEC)$_b$-SbED), where $b > 2$.

(a) Find the necessary and sufficient conditions of such codes.

(b) Prove that a linear binary $(N, N-R)$ (ADEC)$_b$-SbED code exists only if

$$N \leq \left\lfloor b \times \left\lceil \frac{2^b(2^{R-b}-1)}{2b-1} + 1 \right\rceil \right\rfloor.$$

(c) Prove that the null space of

$$\mathbf{H} = [\mathbf{M}_0\ \ \mathbf{M}_1\ \ \mathbf{M}_2 \cdots \mathbf{M}_{n-1}],$$

is an (ADEC)$_b$-SbED code only if $\{\alpha_i,\ \beta_i,\ \alpha_i + \beta_i\} \bigcap \{\alpha_j,\ \beta_j,\ \alpha_j + \beta_j\} = \emptyset$, where $0 \leq i \neq j \leq n-1$, $\emptyset$ denotes the null set, $\mathbf{M}_i$ is an $(r + b - 2) \times b$

matrix defined by

$$\mathbf{M}_i = \left[\begin{array}{cc|cccccc} 0 & 0 & & & & & & \\ 0 & 0 & & & & & & \\ 0 & 0 & & & \mathbf{A}_i & & & \\ \vdots & \vdots & & & & & & \\ 0 & 0 & & & & & & \\ \hline \alpha_i & \beta_i & \alpha_i & \beta_i & \alpha_i & \beta_i & \cdots & \alpha_i \ \beta_i \end{array}\right] \begin{array}{l} \updownarrow b-2 \\ \updownarrow r(\geq 4) \end{array}$$

$\longleftarrow b \longrightarrow$

for $0 \leq i \leq n-1$, α_i, $\beta_i \in GF(2^r) - \{0\}$ with $r \geq 4$, $\alpha_i \neq \beta_i$, and $\mathbf{A}_i$ is a binary $(b-2) \times (b-2)$ nonsingular matrix [UMAN02c].

(d) Design the (72, 64) $(\text{ADEC})_4$-S4ED code and the (140, 128) $(\text{ADEC})_8$-S8ED code.

REFERENCES

[BOSS78] D. C. Bossen, L. C. Chang, and C. L. Chen, "Measurement and Generation of Error Correcting Codes for Package Failures," *IEEE Trans. Comput.*, C-27 (March 1978): 201–204.

[BOYA87] I. M. Boyarinov, A. A. Davydov, and B. M. Shabanov, "Error Correction in Main Memory of a High-Capacity Computer," *Automation and Remote Control*, Plenum (1987). (Original in Russian, *Automatika i Telemekhanika*, no. 7 [July 1987]: 152–165, 48 [July 1987]: 956–965.)

[CHEN83] C. L. Chen, "Error-Correcting Codes with Byte Error-Detection Capability," *IEEE Trans. Comput.*, C-32 (July 1983): 615–621.

[CHEN84] C. L. Chen and M. Y. Hsiao, "Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review," *IBM J. Res. Dev.*, 28 (March 1984): 124–134.

[CHEN86] C. L. Chen, "Error-Correcting Codes for Byte-Organized Memory Systems," *IEEE Trans. Info. Theory*, IT-32 (March 1986): 181–185.

[CHEN92] C. L. Chen, "Symbol Error-Correcting Codes for Computer Memory Systems," *IEEE Trans. Comput.*, 41 (February 1992): 252–256.

[CHEN98] C. L. Chen and M. Y. Hsiao, "Error Detection and Correction for Four-Bit-per-Chip Memory System," US Patent 5,757,823 (May 26, 1998).

[DAVY89] A. A. Davydov and A. Yu. Drozhzhina-Labinskaya, "Length 4 byte error and double independent error correction by BCH code in semiconductor memories," *Automation and Remote Control,* 50 [November 1989]: 1570–1579, Plenum. (Original in Russian, *Automatika i Telemekhanika,* 50 [November 1989]: 135–145.)

[DAVY91] A. A. Davydov and L. M. Tombak, "An Alternative to the Hamming Code in the Class of SEC-DED Codes in Semiconductor Memory," *IEEE Trans. Info. Theory*, 37 (May 1991): 897–902.

[DOET97] G. Doetting, K. J. Getzlaff, B. Leppla, W. Lipponar, T. Pflueger, T. Shlipf, D. Schmunkamp, and U. Wille, "S/390 Parallel Enterprise Server Generation 3: A Balanced System and Cache Structure," *IBM J. Res. Dev.*, 41 (July–September 1997): 405–428.

[DUNN83] L. A. Dunning and M. R. Varanasi, "Code Constructions for Error Control in Byte Organized Memory Systems," *IEEE Trans. Comput.*, C-32 (July 1983): 535–542.

[DUNN85] L. A. Dunning, "SEC-BED-DED Code for Error Control in Byte-Organized Memory Systems," *IEEE Trans. Comput.*, C-34 (June 1985): 557–562.

[DUNN94] L. A. Dunning, ''A SEC-BED-DED Code with Byte Plus Bit Error Detection,'' *Dig. 24th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1994): 208–211.

[FUJI80a] E. Fujiwara, ''Error Control for Byte-per-Package Organized Memory Systems'' (in Japanese), *Trans. IECE Japan*, E-63 (February 1980): 98–103.

[FUJI81b] E. Fujiwara and S. Kaneda, ''Rotational Byte Error Detecting Codes for Memory Systems,'' *Trans. IECE Japan*, E-64 (May 1981): 342–349.

[FUJI91a] E. Fujiwara and T. Gohya, ''Single Byte Error Correcting — Double Bit Error Detecting (S*b*EC-DED) Codes,'' *Proc. 1991 IEEE Int. Symp. on Information Theory* (January 1991): 140.

[FUJI91b] E. Fujiwara and T. Gohya, ''A Design Method for Single Byte Error Correcting — Double Bit Error Detecting Codes'' (in Japanese), *IEICE Technical Report*, FTS91-13 (May 1991).

[FUJI93] E. Fujiwara and M. Hamada, ''Single *b*-bit Byte Error Correcting and Double Bit Error Detecting Codes for High-Speed Memory Systems,'' *Trans. IEICE Japan*, E76-A (September 1993): 1442–1448.

[GILS86] W. J. van Gils, ''An Error-Control Coding System for Storage of 16-bit Words in Memory Arrays Composed of Three 9-bit Wide Units,'' *Philips J. Res.*, 41 (1986): 391–399.

[HAMA91] M. Hamada and E. Fujiwara, ''A New Design Method for Single *b*-Bit Byte Error Correcting and Double Bit Error Detecting (S*b*EC-DED) Codes'' (in Japanese), *Proc. 1991 Symp. on Infomation Theory and Its Applications (SITA'91)*, (December 1991): 61–64.

[HAMA93] M. Hamada and E. Fujiwara, ''A Class of Error Control Codes for Memory Systems — S*b*EC-(S*b*+S)ED Codes — ,'' *Proc. 1993 IEEE Int. Symp. on Information Theory* (January 1993): 244.

[HAMA97] M. Hamada and E. Fujiwara, ''A Class of Error Control Codes for Byte Organized Memory System—S*b*EC-(S*b* + S)ED Codes—,'' *IEEE Trans. Comput.*, 46 (January 1997): 105–109.

[HOLM99] T. J. Holman, ''Encoder and Decoder for an SEC-DED-S4ED Rotational Code,'' US Patent 5,856,987 (January 5, 1999).

[HONG72] S. J. Hong and A. M. Patel, ''A General Class of Maximal Codes for Computer Applications,'' *IEEE Trans. Comput.*, C-21 (December 1972): 1322–1331.

[KANE83] S. Kaneda, ''A Class of SEC-DED-S*b*ED Codes Detecting Byte Error through *b*-Grouped Parity-Checking'' (in Japanese), *Trans. IECE Japan,* J66-D (June 1983): 699–706.

[KANE84] S. Kaneda, ''A Class of Odd-Weight-Column SEC-DED-S*b*ED Codes for Memory System Applications,'' *IEEE Trans. Comput.*, C-33 (August 1984): 737–739. (Also in *Dig. 14th An. Int. Symp. on Fault-Tolerant Comput.* [June 1984]: 88–93.)

[KANE85] S. Kaneda, ''A Class of SEC-DED-S*b*ED Codes for Semiconductor Memory Systems'' (in Japanese), *Trans. IECE Japan*, J68-D (January 1985): 17–24.

[LUI78] A. S. Lui and M. Arbab, ''Error Checking and Correcting Device,'' US Patent 4077028 (February 28, 1978).

[MASS96] L. W. Massengil, ''Cosmic and Terrestrial Single Event Radiation Effects in Dynamic Random Access Memories,'' *IEEE Trans. Nucl. Sci.*, 43 (April 1996): 576–593.

[MCWI77] F. J. McWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland (1977).

[NUMA89] K. Numata, Y. Oowaki, Y. Itoh, et al., ''New Nibbled-Page Architecture for High-Density DRAM's,'' *IEEE J. Solid-State Circ.*, 24 (August 1989): 900–904.

[PATE80] A. M. Patel, ''Error Recovery Scheme for the IBM 3850 Mass Storage System,'' *IBM J. Res. Dev.*, 24 (January 1980): 32–42.

[PENZ95] L. Penzo, D. Sciuto, and C. Silvano, ''Construction Techniques for Systematic SEC-DED codes with Single Byte Error Detection and Partial Correction Capability for Computer Systems,'' *IEEE Trans. Info. Theory*, 41 (March 1995): 584–591.

[REDD78] S. M. Reddy, "A Class of Linear Codes for Error Control in Byte-per-Card Organized Digital Systems," *IEEE Trans. Comput.*, C-27 (May 1978): 455–459.

[SAEK96] T. Saeki, Y. Nakaoka, and M. Fujita, et al., "A 2.5-ns Clock Access, 256 MHz, 256 Mb SDRAM with Synchronous Mirror Delay," *IEEE J. Solid-State Circ.*, 31 (November 1996): 1656–1668.

[SING64] R. C. Singleton, "Maximum Distance Q-Nary Codes," *IEEE Trans. Info. Theory*, IT-10 (April 1964): 116–118.

[SPAI99] L. Spainhower and T. A. Gregg, "IBM S/390 Parallel Enterprise Server G5 Fault Tolerance: A Historical Perspective," *IBM J. Res. Dev.*, 43 (September–November 1999): 863–873.

[SUNA95] T. Sunaga, K. Hosokawa, Y. Nakamura, et al., "A Full Bit Perfect Architecture for Synchronous DRAM's," *IEEE J. Solid-State Circ.*, 30 (November 1995): 998–1005.

[SUNM95] Sun Microsystems, Inc., "Ultra SPARC-I Data Buffer (UDB) DATA SHEET, Revision 0.3," SPARC Technology (May 1995).

[TSUC86] T. Tsuchimoto, K. Shimizu, M. Takamura, M. Shinohara, T. Miyazawa, and H. Tone, "A Large Computer System M-780" (in Japanese), *Nikkei Electronics*, 396 (June 2, 1986): 179–209.

[UMAN02a] G. Umanesan and E. Fujiwara, "Random Double Bit Error Correcting — Single *b*-bit Byte Error Correcting (DEC-S*b*EC) Codes for Memory Systems," *IEICE Trans. Fundamentals*, E85-A (January 2002): 273–276.

[UMAN02b] G. Umanesan and E. Fujiwara, "Single Byte Error Correcting Codes with Double Bit within a Block Error Correcting Capability for Memory Systems," *IEICE Trans. Fundamentals*, E85-A (February 2002): 513–517.

[UMAN02c] G. Umanesan and E. Fujiwara, "Adjacent Double Bit Error Correcting Codes with Single Byte Error Detecting Capability for Memory Systems," *IEICE Trans. Fundamentals*, E85-A (February 2002): 490–496.

[VARA83] M. R. Varanasi, T. R. N. Rao, and S. Pham, "Memory Package Error Detection and Correction," *IEEE Trans. Comput.*, C-32 (September 1983): 872–874.

[XIAO96] S. Xiao, X. Shi, G. Feng, and T. R. N. Rao, "A Generalization of the Single *b*-bit Byte Error Correcting and Double Bit Error Detecting Codes for High-Speed Memory Systems," *IEEE Trans. Comput.*, 45 (April 1996): 508–511.

CONTENTS

7

Codes for High-Speed Memories IV: Spotty Byte Error Control Codes

Some of the error control codes mentioned in the previous chapters have been applied to high-speed memory systems using RAM chips with either 1-bit I/O data $(b = 1)$ or 4-bit I/O data $(b = 4)$. However, modern large-capacity memory systems use RAM chips with 8, 16, or 32 bits of I/O data. A new class of codes called spotty byte error control codes has been developed for those memory systems that use high-density DRAM chips with wide I/O data [UMAN03a, 03b, KASH04, SUZU04, 05a, 05b].

Spotty byte error is a special type of byte error, defined as a t-bit error in a b-bit byte, where $1 \leq t \leq b$. This is based on the fact that the dominant errors in byte-organized chips, even in RAM chips with 8-bit, 16-bit, and 32-bit I/O data, are single-bit errors, or at most double-bit or triple-bit errors, which are sometimes called *low-density byte errors* [TANI92], or *sparse byte errors*.

Spotty byte error control codes, called *t/b-error control codes*, where t is larger than or equal to 2 and less than at most $b/2$, are more practical than the conventional byte error control codes because they require smaller number of check bits than the existing byte error control codes. Here we deal with designing the code having spotty byte error length t as taking any value from 1 to b. It is apparent that if $t = 1$, the codes are bit error control codes, and if $t = b$, the codes are byte error control codes.

In order to determine the code functions effectively applied to the solid-state data recorder of the satellite systems, the encoder and decoder of some spotty byte error control codes have been designed and implemented by FPGA to evaluate gate count and decoding delays [KANE05].

Code Design for Dependable Systems: Theory and Practical Applications, by Eiji Fujiwara

7.1 SPOTTY BYTE ERRORS

Large-capacity high-speed memory systems often adopt high-density DRAM chips with wide I/O data. Examples of this type of recent DRAM architecture are 16 Mb chips with 8-bit I/O data [NUMA89], 256 Mb chips with 16-bit I/O data [SUGI93], 16 Mb chips with 16-bit I/O data [SUNA95], and 256 Mb chips with 32-bit I/O data [SAEK96, WATA96]. Because of their high-density nature, these DRAM chips are strongly vulnerable to α-particles, neutrons, and so forth. [ZIEG96, OGOR96, SRIN96, MASS96]. In particular, the large-capacity memory systems used at airplane altitudes or in a cosmic environment, that is, memories in aircraft and spacecraft, need to be protected from high-energy neutrons and cosmic rays.

From the coding standpoint the multiple errors that occur within a chip can result in a byte error with only a few corrupted random bits. This is why we call such error a *spotty error*, meaning that only two or three bits (i.e., less than $b/2$ bits) are corrupted in a byte. We also refer to spotty errors within a byte as *spotty byte errors*. The burst / byte error detecting codes with the SEC capability do not correct any multiple random bit errors occurring within a single byte. Whereas the SbEC codes and SbEC-DbED codes can correct any single-byte errors, they require many check bits proportional to the byte size b. The byte error control codes treat a chip output as a byte. The minimum number of check bits required by the SbEC codes and SbEC-DbED codes is at least $2b$ bits and $3b$ bits, respectively. For example, an S16EC code applied to a memory system that adopts RAM chips with 16-bit I/O data requires 32 check bits. For practical information lengths of 64, 128, and 256 bits, 32-bit or larger redundancy, in general, is not preferable. Spotty byte error control codes, namely *t/b-error control codes*, offer a better practical solution to such problems.

For mathematical tractability, spotty byte errors are denoted as *t/b-errors*, which stands for random t-bit errors occurring within a single b-bit byte. In this chapter we use a t/b-error to represent a binary row vector $E \in GF(2^b)$ such that $1 \leq w(E) \leq t$, where $w(E)$ is the Hamming weight of E, and we often take the value of t as being either 2 or 3 and the value of b as being 8 or 16 for the practical parameters of the spotty byte errors.

In this chapter we deal with two types of spotty byte errors: s-spotty byte errors and m-spotty byte errors. An s-spotty byte error is defined as a set of random t or fewer bits errors confined to a b-bit byte, and an m-spotty byte error is defined as multiple spotty byte errors concentrated in one byte or distributed in multiple bytes. Sections 7.2 through 7.4 present a class of m-spotty byte error control codes correcting one spotty byte error, and codes detecting single-byte errors or two m-spotty byte errors besides correcting one spotty byte error. Section 7.5 shows a general class of s-spotty byte error control codes and m-spotty byte error control codes, both of which can correct and detect any number of spotty byte errors.

7.2 SINGLE SPOTTY BYTE ERROR CORRECTING ($S_{t/b}$EC) CODES

An $S_{t/b}$EC code corrects all single t/b-errors in a received word. This code can be considered a more generalized version of the SEC codes and the SbEC codes presented in the previous chapters. That is, for $t = 1$ and b, the $S_{1/b}$EC and $S_{b/b}$EC code functions are

equivalent to SEC and SbEC code functions, respectively. For any $(N, N-R)$S$_{t/b}$EC code, the following two inequalities hold:

$$R \geq 2t, \tag{7.1}$$

$$2^R \geq 1 + \frac{N}{b} \cdot \sum_{i=1}^{t} \binom{b}{i}. \tag{7.2}$$

Inequality (7.1) provides a lower bound on the number of check bits required by an S$_{t/b}$EC code. Inequality (7.2) provides the upper bound on code length or information-bit length for a given check-bit length. In the following subsections we will study two design methods of the S$_{t/b}$EC codes [UMAN03a].

7.2.1 Codes Based on Tensor Product of Matrices

S$_{t/b}$EC codes designed by a tensor product of two matrices were first implicitly suggested by J. K. Wolf [WOLF65] in 1965. In 1992, N. H. Vaidya and D. K. Pradhan [VAID92] explicitly proposed and evaluated S$_{t/b}$EC codes designed by a tensor product of two matrices for application to byte-organized systems. In this subsection we will first study the design of S$_{t/b}$EC codes using a tensor product of two matrices.

Let $\mathbf{H}' = [h'_{1k}]$ be a $1 \times q$ matrix and $\mathbf{H}'' = [h''_{ij}]$ be an $m \times n$ matrix. If $\mathbf{H}'$ and $\mathbf{H}''$ can be written as

$$\mathbf{H}' = \begin{bmatrix} h'_{11} & h'_{12} & \cdots & h'_{1q} \end{bmatrix} \quad \text{and} \quad \mathbf{H}'' = \begin{bmatrix} h''_{11} & h''_{12} & \cdots & h''_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ h''_{m1} & h''_{m2} & \cdots & h''_{mn} \end{bmatrix},$$

where h'_{1k} and h''_{ij} are elements of $GF(2^p)$, then the matrix $\mathbf{H}$, defined as the tensor product of matrices $\mathbf{H}''$ and $\mathbf{H}'$ (in this order), is the $m \times nq$ matrix over $GF(2^p)$ given by

$$\mathbf{H} = \mathbf{H}'' \otimes \mathbf{H}' = \begin{bmatrix} h''_{11} \cdot \mathbf{H}' & \cdots & h''_{1n} \cdot \mathbf{H}' \\ \vdots & \ddots & \vdots \\ h''_{m1} \cdot \mathbf{H}' & \cdots & h''_{mn} \cdot \mathbf{H}' \end{bmatrix}$$

$$= \begin{bmatrix} h''_{11}h'_{11} & h''_{11}h'_{12} & \cdots & h''_{11}h'_{1q} & \cdots & \cdots & h''_{1n}h'_{11} & h''_{1n}h'_{12} & \cdots & h''_{1n}h'_{1q} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ h''_{m1}h'_{11} & h''_{m1}h'_{12} & \cdots & h''_{m1}h'_{1q} & \cdots & \cdots & h''_{mn}h'_{11} & h''_{mn}h'_{12} & \cdots & h''_{mn}h'_{1q} \end{bmatrix},$$

where $h''_{ij}h'_{1k}$ denotes the usual multiplication of elements h''_{ij} and h'_{1k} in $GF(2^p)$. The resulting matrix $\mathbf{H}$ can be conveniently considered an $m \times nq$ matrix over $GF(2^p)$ or an $mp \times nq$ matrix over $GF(2)$, meaning a binary matrix. The following theorem, given in [VAID92], illustrates the construction of S$_{t/b}$EC codes when using a tensor product of two parity-check matrices.

Theorem 7.1 [VAID92] *Let $\boldsymbol{C}'$ be a binary $(b,\ b-r')$ code with parity-check matrix $\boldsymbol{H}'$ that corrects all t-bit errors. Let $\boldsymbol{C}''$ be a single error correcting $(N'',\ N''-R'')$ code over $GF(2^{r'})$. Let $\boldsymbol{C}''$ have a parity-check matrix $\boldsymbol{H}''=[\gamma_{ij}]$, γ_{ij} being an element in $GF(2^{r'})$. Then*

$$\boldsymbol{H}=\begin{bmatrix}\gamma_{11}\boldsymbol{H}' & \cdots & \gamma_{1N''}\boldsymbol{H}' \\ \vdots & \ddots & \vdots \\ \gamma_{R''1}\boldsymbol{H}' & \cdots & \gamma_{R''N''}\boldsymbol{H}'\end{bmatrix}$$

is a parity-check matrix of a $(bN'',\ bN''-r'R'')S_{t/b}EC$ code over $GF(2)$.

In this construction the binary column vectors of $\mathbf{H}'$ are treated as elements of $GF(2^{r'})$. The syndrome S consists of R'' component vectors where each such component vector is a binary column vector of $GF(2^{r'})$. The syndrome generated by a single t/b-error corrupting the k-th byte is given by

$$S=\begin{bmatrix}\gamma_{1k}(E\cdot\mathbf{H}'^{T}) \\ \gamma_{2k}(E\cdot\mathbf{H}'^{T}) \\ \vdots \\ \gamma_{R''k}(E\cdot\mathbf{H}'^{T})\end{bmatrix},$$

where $E\in GF(2^b)$ is a nonzero binary row vector such that the Hamming weight of E is less than or equal to t. Then, since $0\neq E\cdot\mathbf{H}'^{T}\in GF(2^{r'})$ and $\boldsymbol{C}''$ is a single-symbol error correcting code over $GF(2^{r'})$, we can determine both the error vector E and the error location k from the syndrome S. Knowing $E\cdot\mathbf{H}'^{T}$, which corresponds to the syndrome for t-bit error in a codeword of $\boldsymbol{C}'$, we can correct the bits in the k-th byte which are in error.

Example 7.1 [VAID92]

Let $\boldsymbol{C}'$ be a $(15,\ 7)$ distance-5 BCH code over $GF(2)$, and let $\boldsymbol{C}''$ be a $(257,\ 255)$ single symbol error correcting Hamming code over $GF(2^8)$. Then the $(3855,\ 3839)$ code over $GF(2)$ obtained by using Theorem 7.1 is an $S_{2/15}EC$ code.

7.2.2 Efficient $S_{t/b}EC$ Codes

Here we will study another code design technique that yields practical $S_{t/b}EC$ codes. The codes presented in this subsection require fewer check bits than the codes based on tensor products of parity-check matrices [UMAN03a].

Let $\mathbf{H}'=[h'_0\ h'_1\ \cdots\ h'_{b-1}]$ be a $q\times b$ binary matrix where any $\min(2t,\ b)$ or fewer column vectors are linearly independent. In this matrix $h'_0, h'_1, \cdots h'_{b-1}$ are all binary column vectors of $GF(2^q)$ and $\min(u,\ v)$ expresses the minimum value of u and v. If $\min(2t,\ b)=b$, the matrix $\mathbf{H}'$ can be any $b\times b$ nonsingular matrix, including the $b\times b$ identity matrix. On the other hand, if $\min(2t,\ b)=2t<b$, we consider $\mathbf{H}'$ to be a parity-check matrix of a linear binary $(b,\ b-q)$ code with minimum distance at least $2t+1$, meaning it is a parity-check matrix of a binary t-error correcting code. Similarly we let $\mathbf{H}''=[h''_0\,h''_1\,h''_2\,\cdots\,h''_i\,\cdots\,h''_{b-1}]$ be an $r\times b$ binary matrix where any t or fewer column

vectors are linearly independent. In this matrix, $h''_0,\ h''_1,\ h''_2,\ \cdots,\ h''_i,\ \cdots,\ h''_{b-1}$ are all binary column vectors of $GF(2^r)$. If $t = b$, the matrix $\mathbf{H}''$ can be any $b \times b$ nonsingular matrix, including the $b \times b$ identity matrix. If $t < b$, we consider $\mathbf{H}''$ to be a parity-check matrix of a linear binary $(b,\ b - r)$ code with minimum distance at least $t + 1$, meaning it is a parity-check matrix of a binary t-error detecting code. The following theorem shows how an efficient $S_{t/b}$EC code can be designed using the above-defined $\mathbf{H}'$ and $\mathbf{H}''$ matrices.

Theorem 7.2 *Let γ be a primitive element of $GF(2^{R-q})$, where $R \geq q + r$. Define the $R \times b \cdot 2^{R-q}$ binary submatrix $\boldsymbol{H}_R$ as follows:*

$$\boldsymbol{H}_R = \begin{bmatrix} \boldsymbol{H}' & \boldsymbol{H}' & \boldsymbol{H}' & \cdots & \boldsymbol{H}' & \boldsymbol{H}' \\ \gamma^0\boldsymbol{H}'' & \gamma^1\boldsymbol{H}'' & \gamma^2\boldsymbol{H}'' & \cdots & \gamma^{2^{(R-q)}-2}\boldsymbol{H}'' & \boldsymbol{0}_{(R-q)\times b} \end{bmatrix},$$

where, $\gamma^i\boldsymbol{H}'' = \begin{bmatrix} \gamma^i\phi(h''_0) & \gamma^i\phi(h''_1) & \gamma^i\phi(h''_2) & \cdots & \gamma^i\phi(h''_{b-1}) \end{bmatrix}$ for $0 \leq i \leq 2^{R-q} - 2$, and $\phi : GF(2^r) \rightarrow GF(2^{R-q})$ is a homomorphism of $GF(2^r)$ into $GF\left(2^{R-q}\right)$ under addition. Then the null space of

$$\begin{aligned} \boldsymbol{H} &= \left[\begin{array}{c|c} \boldsymbol{H}_R & \begin{matrix} \boldsymbol{0}_{q\times(R-q)} \\ \boldsymbol{I}_{R-q} \end{matrix} \end{array}\right] \\ &= \left[\begin{array}{ccccccc|c} \boldsymbol{H}' & \boldsymbol{H}' & \boldsymbol{H}' & \cdots & \boldsymbol{H}' & \boldsymbol{H}' & & \boldsymbol{0}_{q\times(R-q)} \\ \gamma^0\boldsymbol{H}'' & \gamma^1\boldsymbol{H}'' & \gamma^2\boldsymbol{H}'' & \cdots & \gamma^{2^{(R-q)}-2}\boldsymbol{H}'' & \boldsymbol{0}_{(R-q)\times b} & & \boldsymbol{I}_{R-q} \end{array}\right] \end{aligned}$$

is an $S_{t/b}$EC code with check-bit length R and its code length in bits $N = b \cdot 2^{R-q} + (R - q)$. The $\boldsymbol{I}_x$ and $\boldsymbol{0}_{x\times y}$ denote the $x \times x$ binary identity matrix and $x \times y$ binary all zero matrix, respectively.

Proof The syndrome consists of two binary vectors, a q-bit vector and an $(R - q)$-bit vector. Let S^* be the syndrome generated by any $(R - q)$-bit byte error E^* corrupting the last byte of the codeword. Similarly let $S^\dagger$ be the syndrome generated by any t/b-error $E^\dagger$ corrupting the second last byte of the codeword. Then

$$S^* = \begin{bmatrix} \mathbf{0} \\ E^* \end{bmatrix}^T, \quad S^\dagger = \begin{bmatrix} E^\dagger \cdot \mathbf{H}' \\ \mathbf{0} \end{bmatrix}^T,$$

where E^* is a nonzero vector of $GF(2^{R-q})$ and $E^\dagger$ is a nonzero vector of $GF(2^b)$ such that the Hamming weight of $E^\dagger$ is not greater than t. For $0 \leq k \leq 2^{R-q} - 2$, let S_k be the syndrome generated by the single t/b-error E_k corrupting the k-th byte. Then

$$S_k = \begin{bmatrix} E_k \cdot \mathbf{H}' \\ E_k \cdot \left(\gamma^k\mathbf{H}''\right) \end{bmatrix}^T, \tag{7.3}$$

where E_k is a nonzero vector of $GF(2^b)$ such that the Hamming weight of E_k is not greater than t. We know that $E^\dagger \cdot \mathbf{H}'^T \neq \mathbf{0} \neq E_k \cdot \mathbf{H}'^T$ because $\mathbf{H}'$ is a parity-check matrix of a binary t-error correcting code. Furthermore

$$E_k \cdot (\gamma^k\mathbf{H}'')^T = \gamma^k\phi(E_k \cdot \mathbf{H}''^T) \neq \mathbf{0}$$

because $\mathbf{H}''$ is a binary t-error detecting code. This implies that $S^* \neq S^\dagger \neq S_k \neq S^*$ for any k where $0 \leq k \leq 2^{R-q} - 2$. Now, to show that the code is $S_{t/b}EC$, we only need to show that any two t/b-errors corrupting any two different bytes generate two different syndromes, that is, $S_i \neq S_j$ whenever $i \neq j$. Suppose that

$$S_i = \begin{bmatrix} E_i \cdot \mathbf{H}' \\ E_i \cdot (\gamma^i \mathbf{H}'') \end{bmatrix}^T = \begin{bmatrix} E_j \cdot \mathbf{H}' \\ E_j \cdot (\gamma^j \mathbf{H}'') \end{bmatrix}^T = S_j \tag{7.4}$$

holds for some $0 \leq i \neq j \neq 2^{R-q} - 2$. Errors E_i and E_j are nonzero vectors of $GF(2^b)$ such that the Hamming weights of both vectors are not greater than t. Then, from Eq. (7.4), $E_i \cdot \mathbf{H}'^T = E_j \cdot \mathbf{H}'^T$ implies that $E_i = E_j$ because $\mathbf{H}'$ is a parity-check matrix of binary t-error correcting code. Subsequently $\phi\ (E_i \cdot \mathbf{H}''^T) = \phi\ (E_j \cdot \mathbf{H}''^T)$ where both $\phi\ (E_i \cdot \mathbf{H}''^T)$ and $\phi\ (E_j \cdot \mathbf{H}''^T)$ are nonzero vectors of $GF(2^{R-q})$. Now, again from Eq. (7.4), we have $E_i \cdot (\gamma^i \mathbf{H}'')^T = E_j \cdot (\gamma^i \mathbf{H}'')^T$, that is, $\gamma^i \phi\ (E_i \cdot \mathbf{H}''^T) = \gamma^j \phi\ (E_j \cdot \mathbf{H}''^T)$, which leads to $i = j$ and contradicts our initial assumption. This proves that the null space of $\mathbf{H}$ is an $S_{t/b}EC$ code. It is apparent that the code length (in bits) of the code is $N = b \cdot 2^{R-q} + (R - q)$, where R denotes the check-bit length. Q.E.D.

Note that we have conveniently grouped the last $R - q$ check bits into an irregular byte that is $(R - q)$-bit long. The code is capable of correcting all random t-bit errors that occur within this irregular byte. Therefore, if $R - q > b$, this byte can be divided into a regular b-bit byte and another $(R - q - b)$-bit byte. The code will still correct all random t-bit errors that occur within these bytes.

Lemma 7.1 *The null space of*

$$\boldsymbol{H} = \left[\boldsymbol{H}_R \;\middle|\; \begin{array}{c} \boldsymbol{0}_{q\times b}\ \boldsymbol{0}_{q\times b} \cdots \boldsymbol{0}_{q\times b} \\ \hline \boldsymbol{H}_{R-q} \end{array} \;\middle|\; \begin{array}{c} \boldsymbol{0}_{2q\times(R-2q)} \\ \hline \boldsymbol{I}_{R-2q} \end{array} \right],$$

where $R \geq 2q + r$, is an $S_{t/b}EC$ code with check-bit length R and code length in bits $N = b \cdot 2^{R-q} + b \cdot 2^{R-2q} + (R - 2q)$. Here $\boldsymbol{H}_R$ and $\boldsymbol{H}_{R-q}$ are $R \times b \cdot 2^{R-q}$ and $(R - q) \times b \cdot 2^{R-2q}$ binary submatrices, respectively, as defined in Theorem 7.2.

Proof According to Theorem 7.2, the null spaces of $\mathbf{H}_R$ and $\mathbf{H}_{R-q}$ individually denote two $S_{t/b}EC$ codes. Therefore the null space of

$$\left[\mathbf{H}_R \;\middle|\; \begin{array}{c} \mathbf{0}_{q\times b}\ \mathbf{0}_{q\times b} \cdots \mathbf{0}_{q\times b} \\ \hline \mathbf{H}_{R-q} \end{array} \right]$$

is also an $S_{t/b}EC$ code because any t-bit error pattern corrupting a byte in the first partition generates a nonzero syndrome with a nonzero upper q-bit column vector. In the second partition such a t-bit error pattern corrupting a byte generates a nonzero syndrome with an all-zero upper q-bit column vector. On the other hand, any t-bit error pattern corrupting the last byte generates a nonzero syndrome with an all-zero upper $2q$-bit column vector. Therefore the t-bit errors corrupting the last partition are distinguishable because no t-bit error pattern corrupting any byte in the other two partitions can generate such an all-zero $2q$-bit upper column vector. This proves that the null space of $\mathbf{H}$ is an $S_{t/b}EC$ code. It is

apparent that the check-bit length of the code is R and the code length in bits is $N = b \cdot 2^{R-q} + b \cdot 2^{R-2q} + (R - 2q)$. Q.E.D.

Theorem 7.2 and Lemma 7.1 suggest that if $R \geq \lambda q + r$, where λ is a positive integer, we can concatenate λ iterative partitions to obtain a long $S_{t/b}EC$ code. The following theorem uses an iterative concatenation of partitions to obtain a new class of $S_{t/b}EC$ codes. This design technique is same as the one used in the Hong-Patel maximal SbEC codes of Subsection 5.1.4. In this theorem we use $\mathbf{P}_c$ to denote the partition that corresponds to a check-bit portion of the codeword and $\mathbf{P}_i$, where $0 \leq i \leq \lambda - 1$, to denote the i-th partition included in other portion of the codeword.

Theorem 7.3 *Let $R \geq \lambda q + r$. Then the null space of*

$$\boldsymbol{H} = \left[\begin{array}{c|c|c|c|c|c} \boldsymbol{H}_R & \begin{array}{c}\boldsymbol{0}_{q\times b}\boldsymbol{0}_{q\times b}\cdots\boldsymbol{0}_{q\times b}\\ \hline \\ \boldsymbol{H}_{R-q}\end{array} & \begin{array}{c}\boldsymbol{0}_{q\times b}\boldsymbol{0}_{q\times b}\cdots\boldsymbol{0}_{q\times b}\\ \boldsymbol{0}_{q\times b}\boldsymbol{0}_{q\times b}\cdots\boldsymbol{0}_{q\times b}\\ \hline \\ \boldsymbol{H}_{R-2q}\end{array} & \cdots & \begin{array}{c}\boldsymbol{0}_{q\times b}\boldsymbol{0}_{q\times b}\cdots\boldsymbol{0}_{q\times b}\\ \boldsymbol{0}_{q\times b}\boldsymbol{0}_{q\times b}\cdots\boldsymbol{0}_{q\times b}\\ \vdots\\ \boldsymbol{0}_{q\times b}\boldsymbol{0}_{q\times b}\cdots\boldsymbol{0}_{q\times b}\\ \hline \boldsymbol{H}_{R-(\lambda-1)_q}\end{array} & \begin{array}{c}\boldsymbol{0}_{\lambda q\times(R-\lambda q)}\\ \\ \boldsymbol{I}_{R-\lambda q}\end{array} \end{array}\right]$$

$$= \left[\begin{array}{c|c|c|c|c} \boldsymbol{P}_0 & \boldsymbol{P}_1 & \boldsymbol{P}_2 & \cdots & \boldsymbol{P}_{\lambda-1} \end{array}\right] \quad \boldsymbol{P}_c \Big]$$

is an $S_{t/b}EC$ code with check-bit length R and code length in bits $N = (R - \lambda q) + \sum_{i=1}^{\lambda} b \cdot 2^{R-iq}$. Here $\lambda(\geq 1)$ is an integer, and for $p = 0, 1, \cdots, \lambda - 1$, $\boldsymbol{H}_{R-pq}$ denotes the $(R - pq) \times (b \cdot 2^{R-(p+1)q})$ binary submatrices, as defined in Theorem 7.2.

Proof From Theorem 7.2 we know that each partition corresponds to an $S_{t/b}EC$ code. Then, by iteratively applying Lemma 7.1, we can show that the null space of $\mathbf{H}$ is an $S_{t/b}EC$ code. The code length in bits of this code is given by

$$\begin{aligned} N &= (R - \lambda q) + b \cdot 2^{R-q} + b \cdot 2^{R-2q} + \cdots + b \cdot 2^{R-\lambda b} \\ &= (R - \lambda q) + \sum_{i=1}^{\lambda} b \cdot 2^{R-iq}, \end{aligned}$$

where R denotes the check-bit length. Q.E.D.

Corollary 7.1 *If any b column vectors both in $\boldsymbol{H}'$ and $\boldsymbol{H}''$ are linearly independent, the code obtained by applying Theorem 7.2 is a single b-bit byte error correcting (SbEC) code.*

Proof It is clear that when $t = b$, the $S_{t/b}EC$ code function becomes single b-bit byte error correction (SbEC). Therefore, if any b column vectors both in $\mathbf{H}'$ and $\mathbf{H}''$ are linearly independent, the code obtained by applying Theorem 7.3 is an SbEC code. Q.E.D.

For the case where $t = b$ we can choose the $b \times b$ binary identity matrix in the code design, meaning $\mathbf{H}' = \mathbf{H}'' = \mathbf{I}_b$. Then the code obtained by applying Theorem 7.3 represents the well-known maximal codes called Hong-Patel codes [HONG72]. In other words, the code denoted by Theorem 7.3 includes the Hong-Patel codes as a special case when $t = b$ and $\mathbf{H}' = \mathbf{H}'' = \mathbf{I}_b$.

Corollary 7.2 *If $t = b = 1$ and $\boldsymbol{H}' = \boldsymbol{H}'' = \boldsymbol{I}_1$, the code obtained by applying Theorem 7.3 is a single-bit error correcting (SEC) code. Here $\boldsymbol{I}_1$ denotes 1×1 binary identity matrix, which is a binary 1.*

Proof It is clear that when $t = b = 1$, the $S_{t/b}EC$ code function becomes single-bit error correction (SEC). Therefore, if any one column vector both in $\mathbf{H}'$ and in $\mathbf{H}''$ is linearly independent, the code obtained by applying Theorem 7.3 is an SEC code. In this case simply $\mathbf{H}' = \mathbf{H}'' = \mathbf{I}_1$. Q.E.D.

For example, let $t = b = 1$, $\mathbf{H}' = \mathbf{H}'' = \mathbf{I}_1$, and $\lambda = 3$. Then the following shows the parity-check matrix obtained by applying Theorem 7.3:

$$\left[\begin{array}{cccccccc|cccc|cc|c}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1
\end{array}\right].$$

Clearly, the matrix gives a single-bit error correcting Hamming code, which is the (15, 11) SEC code. This demonstrates that the code denoted by Theorem 7.3 includes single-bit error correcting Hamming code as a special case when $t = b = 1$ and $\mathbf{H}' = \mathbf{H}'' = \mathbf{I}_1$.

7.2.3 Practical Examples

In this section we consider two practical $S_{t/b}EC$ codes that can be applied to high-speed memory systems using 8-bit or 16-bit I/O data memory chips. First, we take the 16-bit case and illustrate the design of the practical $S_{t/b}EC$ code where $t = 3$ and $b = 16$. The resulting code is called an $S_{3/16}EC$ code, and it corrects up to 3-bit random errors corrupting a single 16-bit byte. We can design this code by using the binary parity-check matrices of distance-7 and distance-4 codes as $\mathbf{H}'$ and $\mathbf{H}''$ matrices, respectively.

To this end, we know that $\binom{16}{0} + \binom{16}{1} + \binom{16}{2} + \binom{16}{3} = 697 > 2^9 = 512$. Nevertheless, a computer search of the entire $GF(2^{10})$ space indicates that there is no (16, 6) distance-7 code. A search of the $GF(2^{11})$ space yields the following (16, 5) distance-7 code:

$$\mathbf{H}' = \left[\alpha^0\ \alpha^1\ \alpha^2\ \alpha^3\ \alpha^4\ \alpha^5\ \alpha^6\ \alpha^7\ \alpha^8\ \alpha^9\ \alpha^{10}\ \alpha^{275}\ \alpha^{337}\ \alpha^{863}\ \alpha^{1412}\ \alpha^{1849}\right],$$

where α is a primitive element of $GF(2^{11})$ corresponding to primitive polynomial $\mathbf{p}(x) = x^{11} + x^2 + 1$. On the other hand, the parity-check matrix of a distance-4 binary

extended cyclic Hamming code with code length 16 bits, meaning a (16, 11) SEC-DED code, can be considered as $\mathbf{H}''$ matrix. Then we have

$$\mathbf{H}'' = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & \beta^0 & \beta^1 & \beta^2 & \beta^3 & \beta^4 & \beta^5 & \beta^6 & \beta^7 & \beta^8 & \beta^9 & \beta^{10} & \beta^{11} & \beta^{12} & \beta^{13} & \beta^{14} \end{bmatrix}$$

$$= \begin{bmatrix} \gamma^0 & \gamma^{18} & \gamma^5 & \gamma^{29} & \gamma^{10} & \gamma^{11} & \gamma^8 & \gamma^{25} & \gamma^{26} & \gamma^{27} & \gamma^{22} & \gamma^{23} & \gamma^{14} & \gamma^{15} & \gamma^{16} & \gamma^{17} \end{bmatrix},$$

where β is a primitive element of $GF(2^4)$ corresponding to primitive polynomial $\mathbf{p}(x) = x^4 + x + 1$ and γ is a primitive element of $GF(2^5)$ corresponding to primitive polynomial $\mathbf{p}(x) = x^5 + x^2 + 1$. Clearly, $\mathbf{H}'$ is a parity-check matrix of a binary 3-bit error correcting code and $\mathbf{H}''$ is a parity-check matrix of a binary 3-bit error detecting code, as required by Theorem 7.2. The parity-check matrix of the resulting (517, 501) $S_{3/16}$EC code is given by the following matrix:

$$\left[\begin{array}{ccccccc} \mathbf{H}' & \mathbf{H}' & \mathbf{H}' & \cdots & \mathbf{H}' & \mathbf{H}' & \mathbf{O} \\ \gamma^0\mathbf{H}'' & \gamma^1\mathbf{H}'' & \gamma^2\mathbf{H}'' & \cdots & \gamma^{30}\mathbf{H}'' & \mathbf{O} & \mathbf{I}_5 \end{array}\right].$$

Here

$$\gamma^i\mathbf{H}'' = \begin{bmatrix} \gamma^i & \gamma^{i+18} & \gamma^{i+5} & \gamma^{i+29} & \gamma^{i+10} & \gamma^{i+11} & \gamma^{i+8} & \gamma^{i+25} & \gamma^{i+26} & \gamma^{i+27} & \gamma^{i+22} & \gamma^{i+23} & \gamma^{i+14} \\ \gamma^{i+15} & \gamma^{i+16} & \gamma^{i+17} \end{bmatrix}$$

for $0 \le i \le 14$. Notice that in this case the homomorphism $\phi : GF(2^5) \to GF(2^5)$ is simply given by $\phi(x) = x$ for any $x \in GF(2^5)$. A shortened $S_{3/16}$EC code with 256 information bits (i.e., an (272, 256) $S_{3/16}$EC code) can be obtained by simply choosing the first 272 binary columns from the original (517, 501) $S_{3/16}$EC code. Furthermore this shortened code can be made systematic by row operations. Figure 7.1 shows the binary parity-check matrix of this shortened (272, 256) $S_{3/16}$EC code in systematic form. This code requires only one RAM chip (with 16-bit I/O data) for check bits.

Our second example is an $S_{3/8}$EC code that corrects up to three random bit errors corrupting a single 8-bit byte. This code is suitable for memory systems that use chips with 8-bit I/O data. Figure 7.2 shows an example of an $S_{3/8}$EC code, that is, the binary parity-check matrix of a (132, 121) $S_{3/8}$EC code. In this case we use the parity-check matrix of an (8, 1) 3-error correcting code as the $\mathbf{H}'$ matrix, and that of an (8, 4) 3-error detecting code as the $\mathbf{H}''$ matrix. These $\mathbf{H}'$ and $\mathbf{H}''$ matrices are given below in binary form:

$$\mathbf{H}' = \begin{bmatrix} 10000001 \\ 01000001 \\ 00100001 \\ 00010001 \\ 00001001 \\ 00000101 \\ 00000011 \end{bmatrix}, \quad \mathbf{H}'' = \begin{bmatrix} 11111111 \\ 01001011 \\ 00101110 \\ 00010111 \end{bmatrix}.$$

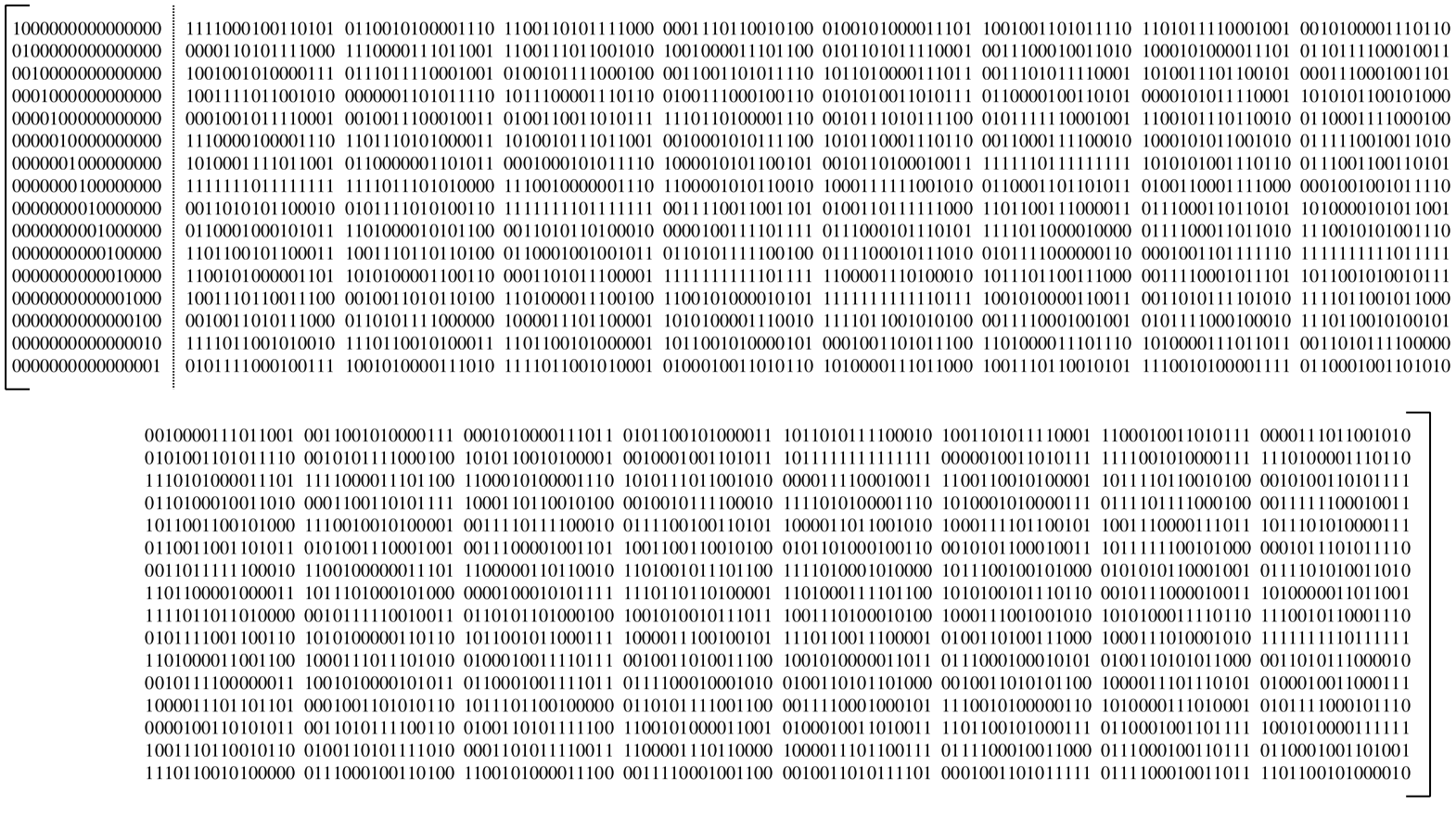


Figure 7.1 *Example of a practical systematic (272, 256)$S_{3/16}$EC code.*

10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	0000
01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	0000
00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	0000
00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	0000
00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	0000
00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	0000
00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	0000
11111111	00010111	00101110	01001011	11101000	00111001	01100101	10100011	11010001	01011100	11000110	01110010	10001101	10011010	10110100	00000000	1000
01001011	11101000	00111001	01100101	10100011	11010001	01011100	11000110	01110010	10001101	10011010	10110100	11111111	00010111	00101110	00000000	0100
00101110	01001011	11101000	00111001	01100101	10100011	11010001	01011100	11000110	01110010	10001101	10011010	10110100	11111111	00010111	00000000	0010
00010111	00101110	01001011	11101000	00111001	01100101	10100011	11010001	01011100	11000110	01110010	10001101	10011010	10110100	11111111	00000000	0001

Figure 7.2 *Example of a practical (132, 121)$S_{3/8}$EC code.* Source: *[UMAN03a]. © 2003 IEICE Japan.*

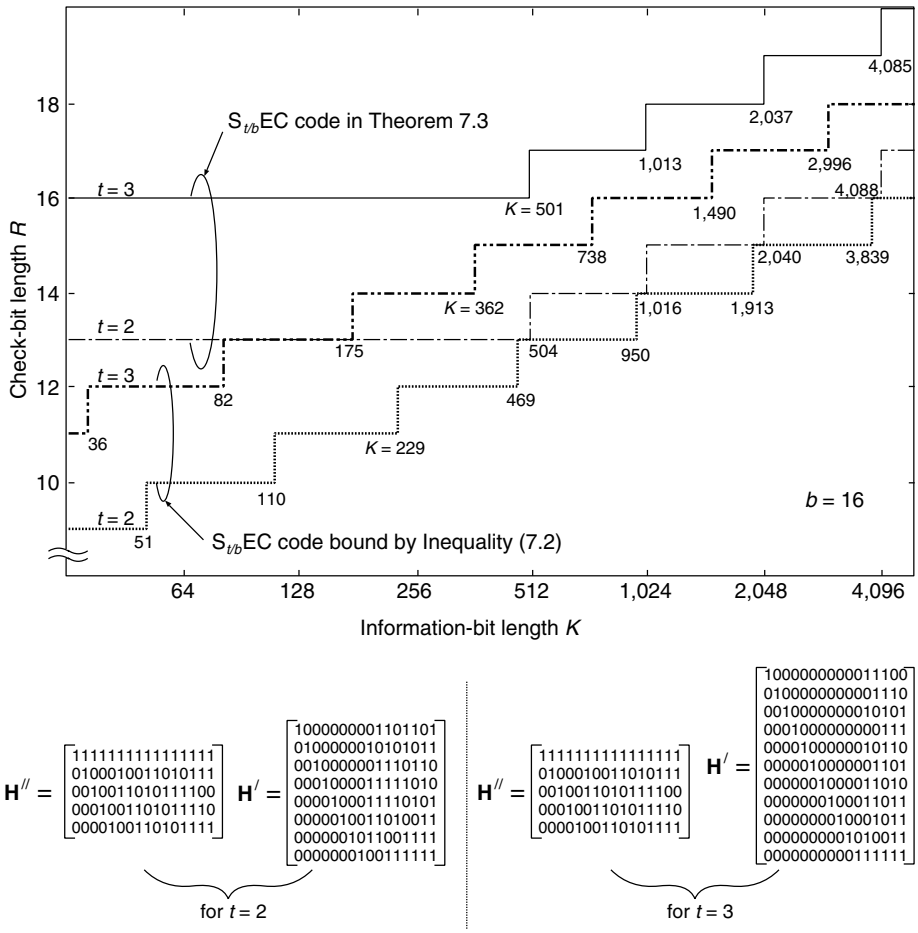


Figure 7.3 *Check-bit lengths, compared with information-bit lengths of the $S_{t/b}$EC codes and the corresponding $\mathbf{H}'$, and $\mathbf{H}''$ matrices for $b = 16$ and $t = 2, 3$.* Source: *[UMAN03a]. © 2003 IEICE Japan.*

Figures 7.3 and 7.4 show the check-bit length and information-bit length relationship of the $S_{t/b}$EC codes for the practical values of b and t. These figures also give the corresponding $\mathbf{H}'$ and $\mathbf{H}''$ matrices that are used in plotting these graphs. The graphs clearly indicate that in all cases, when practical information lengths such as 64, 128, or 256 bits are considered, the number of check bits required by an $S_{t/b}$EC code is significantly less than that of the counterpart SbEC code.

7.3 SINGLE SPOTTY BYTE ERROR CORRECTING AND SINGLE-BYTE ERROR DETECTING ($S_{t/b}$EC-SbED) CODES

In this section we consider a class of codes known as $S_{t/b}$EC-SbED codes [UMAN03b]. An $S_{t/b}$EC-SbED code corrects all single t/b-errors, and detects single-byte errors. In other words, correction is performed if t or fewer bits are in error in a byte, and detection is

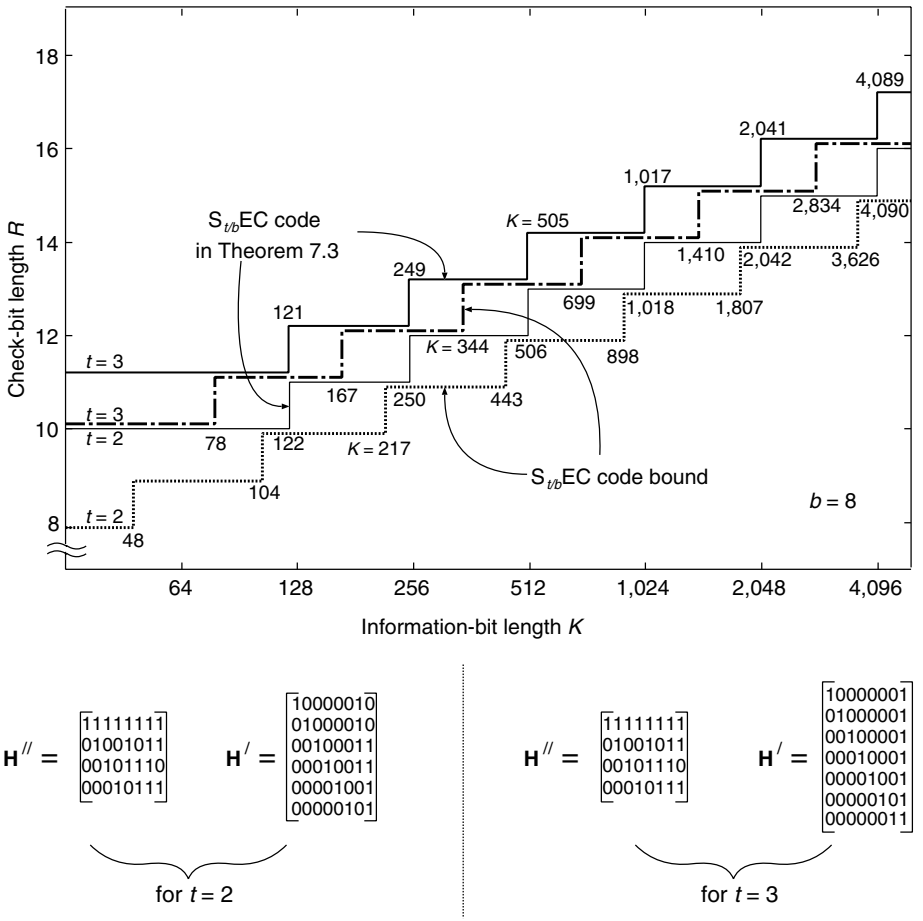


Figure 7.4 *Check-bit lengths compared with information-bit lengths of the $S_{t/b}$EC codes and the corresponding $\boldsymbol{H}'$, $\boldsymbol{H}''$ matrices for $b = 8$ and $t = 2, 3$.* Source: *[UMAN03a]. © 2003 IEICE Japan.*

performed if more than t bits in a byte are in error. For any $(N, N-R)$$S_{t/b}$EC-S$b$ED code, the following two inequalities hold:

$$R \geq b + t, \tag{7.5}$$

$$N/b \leq \frac{2^R - 2^b + \sum_{i=1}^{t} \binom{b}{i}}{\sum_{i=1}^{t} \binom{b}{i}}. \tag{7.6}$$

Inequality (7.5) provides a lower bound on the number of check bits required by an $S_{t/b}$EC-SbED code. Inequality (7.6) provides the upper bound on code length or information-bit length for a given check-bit length.

Theorem 7.4 *If $\boldsymbol{H}' = \boldsymbol{I}_b$, the codes described by the parity-check matrices of Theorem 7.2, Lemma 7.1, and Theorem 7.3 are all systematic $S_{t/b}$EC-SbED codes.*

Proof It is obvious that if $\mathbf{H}' = \mathbf{I}_b$, all single t/b-errors can be corrected. Let $E_1 \in GF(2^b)$ denote a single t/b-error and $E_2 \in GF(2^b)$ denote a b-bit byte error. Then $\mathbf{I}_b \cdot E_1 \neq \mathbf{I}_b \cdot E_2$ because $w(E_1) \leq t$ and $w(E_2) > t$. This proves that the code also detects all single-byte errors. Q.E.D.

Figure 7.5 gives a practical systematic $(76, 64)$ $S_{3/8}$EC-S8ED code that is suitable for application to memory systems with 8-bit RAM chips. This code is a shortened version of the original $(132, 120)S_{3/8}$EC-S8ED code that corrects all single-byte errors with three or fewer bits corrupted, and detects all single-byte errors with more than three bits corrupted. For $b = 8$ and $2 \leq t \leq 8$, Figure 7.6 shows the check-bit lengths plotted against the information-bit lengths of the $S_{t/b}$EC-SbED code. Table 7.1 presents the error detection capabilities of the $S_{3/8}$EC-S8ED code.

7.3.1 Decoding $S_{t/b}$EC-SbED Codes

The decoding method is presented here for the $S_{t/b}$EC-SbED codes derived in Theorem 7.4. This decoding method corrects all single t/b-errors plus some b-bit byte errors that are correctable. Uncorrectable b-bit byte error patterns are only detected. Let v be the received word. The syndrome S can be calculated as follows:

$$v \cdot \mathbf{H}^T = S = [\, S_0,\ S_1 \,],$$

where $S_0 \in GF(2^b)$ and $S_1 \in GF(2^r)$ are, respectively, the b-bit vector and the r-bit vector of the syndrome

$$\left[\begin{array}{cccc|cc} \mathbf{I}_b & \mathbf{I}_b & \cdots & \mathbf{I}_b & \mathbf{I}_b & \mathbf{O}_{b\times r} \\ \gamma^0\mathbf{H}'' & \gamma^1\mathbf{H}'' & \cdots & \gamma^{2^r-2}\mathbf{H}'' & \mathbf{O}_{r\times b} & \mathbf{I}_r \end{array}\right] \begin{array}{l} \rightarrow \\ \rightarrow \end{array} \begin{array}{c} S_0 \\ S_1 \end{array}.$$

We know that the syndrome vector S_0 corresponds to the error pattern in $GF(2^b) - \{0\}$, whereas the syndrome vector S_1 indicates the location of the single t/b-error or a correctable b-bit byte error pattern, except for the case where the error has corrupted the last byte. Based on this knowledge, we can devise a decoding algorithm as follows:

Step 1. If $S_0 = 0$ and $S_1 = 0$, there are no errors. The received word is a codeword.

Step 2. If $S_0 = 0$, $S_1 \neq 0$, the last byte is in error. This byte has r bits in it. The error pattern itself is given by S_1.

TABLE 7.1 Error-Detection Capabilities of the $S_{3/8}$EC-S8ED Code

	Error detection capability (%)		
Error types	$K = 64$ $(R = 12)$	$K = 128$ $(R = 13)$	$K = 256$ $(R = 14)$
Double-bit errors	49.54	48.81	49.45
Triple-bit errors	45.00	47.34	48.70
Byte plus bit errors	80.79	80.81	80.83

Source: *[UMAN03b]. © 2003 IEEE.*

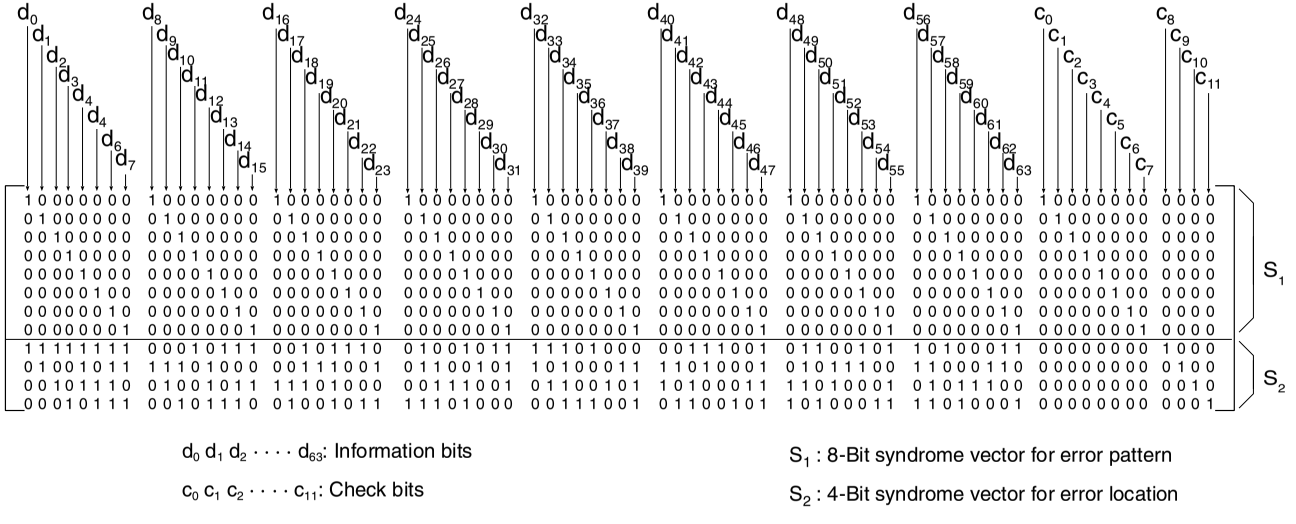


Figure 7.5 *Example of a practical systematic (76, 64) $S_{3/8}EC$-S8ED code.*

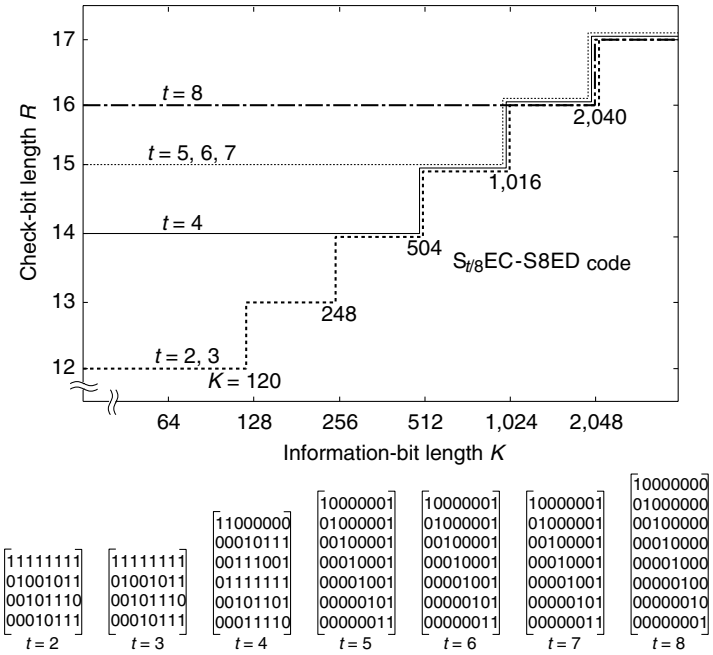


Figure 7.6 *Check-bit lengths compared with information-bit lengths of the $S_{t/8}$EC-S8ED codes and $\mathbf{H}''$ matrices for $2 \leq t \leq 8$.* Source: *[UMAN03b]. © 2003 IEEE.*

Step 3. If $S_0 \neq 0$, $S_1 = 0$, and $S_0 \cdot \mathbf{H}''^T \neq 0$, the error pattern is correctable. The error pattern is given by S_0, and the error location is the second last byte in the received word. On the other hand, if $S_0 \cdot \mathbf{H}''^T = 0$, the error pattern cannot be corrected. In this case we generate a signal to detect such an error.

Step 4. If $S_0 \neq 0$, $S_1 \neq 0$, the error pattern is given by S_0. To find the error location, we calculate $S_0 \cdot (\gamma^i \mathbf{H}'')^T$ in parallel for $0 \leq i \leq 2^r - 2$. The i-th byte is in error if $S_0 \cdot (\gamma^i \mathbf{H}'')^T = S_1$ holds; otherwise, it is not.

Figure 7.7 shows the 8-bit byte error detector for the $S_{3/8}$EC-S8ED code shown in Figure 7.5. This implements the function $S_0 \cdot \mathbf{H}''^T$ and outputs a logical 1 to indicate error detection whenever $S_0 \cdot \mathbf{H}''^T = 0$ for a nonzero syndrome. Figure 7.8 shows the syndrome decoder and the error corrector circuitry corresponding to the first byte of the same $S_{3/8}$EC-S8ED code.

7.3.2 Perfect $S_{t/b}$EC-SbED Code with $t = b - 1$

We know that for the case where $t = b$, the $S_{t/b}$EC-SbED code becomes a single b-bit byte error correcting code (i.e., an SbEC code). It is further well known that perfect SbEC codes do exist, as Hong-Patel codes [HONG72] shown in Subsection 5.1.4. Here we consider the case where $t = b - 1$, that is, $S_{(b-1)/b}$EC-SbED codes that correct all single b-bit byte error patterns except when all b bits in a b-bit byte are in error. It will be shown that perfect $S_{(b-1)/b}$EC-SbED codes exist whenever $R - 1$ is an integer multiple of $b - 1$ and $N = b \cdot (2^{R-1} - 1)/(2^{b-1} - 1)$.

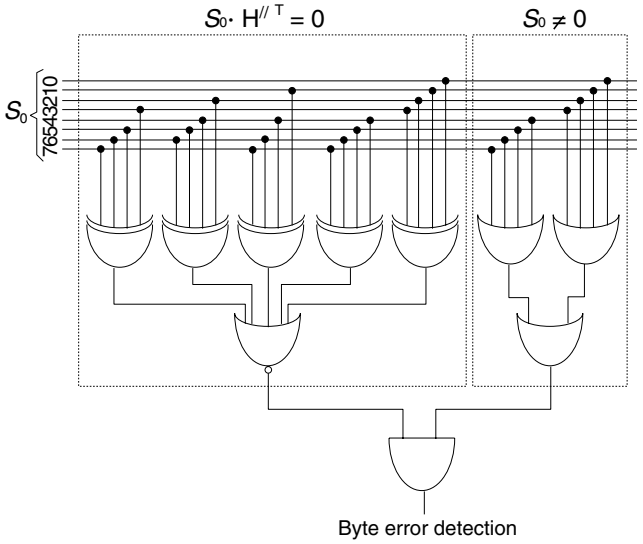


Figure 7.7 *Byte error detector for the $S_{3/8}$EC-S8ED code shown in Figure 7.5.* Source: *[UMAN03b]. © 2003 IEEE.*

It is obvious that an $S_{(b-1)/b}$EC-SbED code requires at least $2b - 1$ check bits. The following lemma states that a binary linear $S_{(b-1)/b}$EC-SbED code with one shortened byte cannot be a perfect code.

Lemma 7.2 *A perfect binary linear $S_{(b-1)/b}$EC-SbED code with one shortened c-bit byte, where $1 \leq c \leq b - 1$, cannot exist.*

Proof Assume that such a perfect code exists with a code length in bits $N = nb + c$ and a check-bit length R, where n is a positive integer. Since $c \leq b - 1$, the $S_{(b-1)/b}$EC-SbED code corrects all error patterns that corrupt the shortened byte with c bits. The total number of different error patterns with $b - 1$ bits corrupted within a single b-bit byte is given by $\sum_{i=1}^{b-1} \binom{b}{i}$. There are n bytes each with b bits; therefore we need a total of $n \times \sum_{i=1}^{b-1} \binom{b}{i}$ different syndromes to correct all these errors. Furthermore $2^c - 1$ syndromes are necessary to correct all diffferent $2^c - 1$ errors that corrupt the shortened byte. Finally we

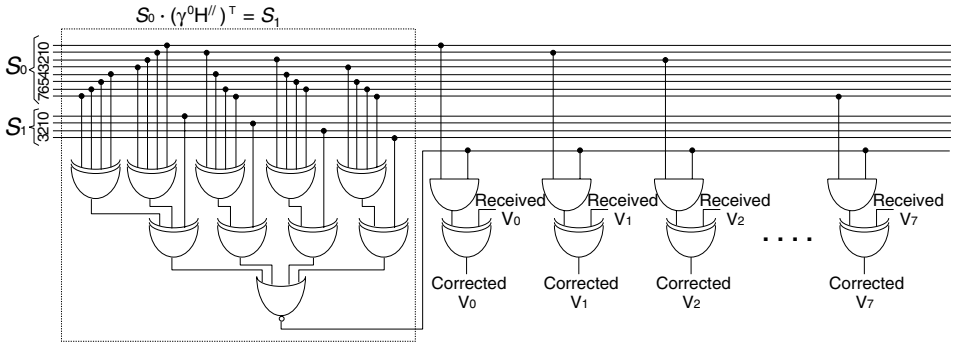


Figure 7.8 *Syndrome decoder and error corrector corresponding to first byte of the $S_{3/8}$EC-S8ED code shown in Figure 7.5.* Source: *[UMAN03b]. © 2003 IEEE.*

need just one syndrome to detect b-bit byte error patterns with all b bits corrupted in it. A code is perfect if and only if it uses all available nonzero syndromes. Therefore we write

$$\begin{aligned} 2^R - 1 &= n \cdot \sum_{i=1}^{b-1} \binom{b}{i} + (2^c - 1) + 1 \\ &= n \cdot (2^b - 2) + (2^c - 1) + 1. \end{aligned}$$

From the above we get the following equation:

$$n = \frac{(2^R - 2) - (2^c - 1)}{2^b - 2}. \tag{7.7}$$

We know that $2^R - 2$ and $2^b - 2$ are even numbers, whereas $2^c - 1$ is an odd number. Therefore the numerator $(2^R - 2) - (2^c - 1)$ of Eq. (7.7) is an odd number and the denominator $2^b - 2$ is an even number. But this case is impossible because n is a natural number. So there is no perfect $S_{(b-1)/b}$EC-SbED code with one shortened byte with c bits, where $1 \leq c \leq b - 1$. Q.E.D.

Theorem 7.5 *A perfect binary linear* $(N,\ N - R)$ $S_{(b-1)/b}$*EC-SbED code exists only if* $R - 1$ *is an integer multiple of* $b - 1$ *and code length in bits* $N = b \cdot (2^{R-1} - 1)/(2^{b-1} - 1)$.

Proof From Lemma 7.2 we know that a perfect $S_{(b-1)/b}$EC-SbED code with one shortened byte cannot exist. Therefore we assume that the code length N is a multiple of byte length b. Then, from the fact that a perfect code uses all available syndromes, we have

$$2^R - 1 = \frac{N}{b} \cdot \sum_{i=1}^{b-1} \binom{b}{i} + 1.$$

From the equation above we obtain the following equation:

$$\frac{N}{b} = \frac{2^R - 2}{2^b - 2} = \frac{2^{R-1} - 1}{2^{b-1} - 1}. \tag{7.8}$$

We know that Eq. (7.8) is true for an integer N/b only when $R - 1$ is an integer multiple of $b - 1$. Subsequently a perfect $(N,\ N - R)$ $S_{(b-1)/b}$EC-SbED codes exists only when $R - 1$ is an integer multiple of $b - 1$, and the code length in bits is given by $N = b \cdot (2^{R-1} - 1)/(2^{b-1} - 1)$. Q.E.D.

The following theorem shows how we can construct a perfect $S_{(b-1)/b}$EC-SbED code by using the $GF(2^{b-1})$ subfield of $GF(2^{R-1})$ whenever $R - 1$ is an integer multiple of $b - 1$.

Theorem 7.6 *Let* α *be a primitive element of* $GF(2^{R-1})$ *such that* $R - 1$ *is an integer multiple of* $b - 1$. *For* $0 \leq i \leq s - 1$, *define the* $(R - 1) \times b$ *binary matrix* $\boldsymbol{H}_i$ *as follows:*

$$\boldsymbol{H}_i = \begin{bmatrix} \alpha^i & \alpha^{i+s} & \alpha^{i+2s} & \cdots & \alpha^{i+(b-2)s} & f(\alpha^i) \end{bmatrix},$$

where $s = (2^{R-1} - 1)/(2^{b-1} - 1)$ *and* $f(\alpha^i) = \sum_{j=0}^{b-2} \alpha^{i+js}$. *The null space of*

$$\boldsymbol{H} = \left[\begin{array}{c|c|c|c|c} 100\cdots00 & 100\cdots00 & 100\cdots00 & \cdots & 100\cdots00 \\ \hline \boldsymbol{H}_0 & \boldsymbol{H}_1 & \boldsymbol{H}_2 & \cdots & \boldsymbol{H}_{s-1} \end{array}\right]$$

is a perfect $S_{(b-1)/b}$*EC-SbED code with a code length in bits* $N = b \times s = b(2^{R-1} - 1)/(2^{b-1} - 1)$ *and a check-bit length R.*

Proof Consider the submatrix $\mathbf{H}_i$ for any $0 \le i \le s-1$. Since $\alpha^i + \alpha^{i+s} + \alpha^{i+2s} + \cdots + \alpha^{i+(b-2)s} = f(\alpha^i)$, the summation of all binary column vectors in $\mathbf{H}_i$ results in a zero vector. This means that any $b-1$ or fewer columns in $\mathbf{H}_i$ are linearly independent because the first $b-1$ elements $\alpha^i,\ \alpha^{i+s},\ \alpha^{i+2s},\ \ldots,\ \alpha^{i+(b-2)s}$ are linearly independent. Further the subspace spanned by the binary column vectors of $\mathbf{H}_i$ in fact represents a multiplicative coset of the subfield $GF(2^{b-1})$. Therefore the subspaces spanned by the binary columns of $\mathbf{H}_i$ and $\mathbf{H}_j$ are disjoint for all $i, j,\ 0 \le i \ne j \le s-1$. This implies that the code has $S_{(b-1)/b}$EC capability. On the other hand, when all the b bits are in error, the resulting syndrome is

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

which is clearly nonzero. Also this syndrome is distinguishable from any $(b-1)/b$-error syndromes because the $(b-1)/b$-errors generate a syndrome of the form

$$\begin{bmatrix} a \\ b \end{bmatrix},$$

where $a \in GF(2),\ b \in GF(2^{R-1}) - \{0\}$. The optimality of the code in Theorem 7.6 can be easily proved by showing that the code length $b \cdot (2^{R-1} - 1)/(2^{b-1} - 1)$ meets the upper bound given by Inequality (7.6). Q.E.D.

By using Theorem 7.6, we can design perfect $S_{(b-1)/b}$EC-S*b*ED codes for any value of $b \ge 2$, and any value of R such that $R-1$ in an integer multiple of $b-1$. Figure 7.9 shows a perfect $S_{(b-1)/b}$EC-S*b*ED code with $b = 4$ and $R = 7$. In this case there are $9 \times \sum_{i=1}^{3} \binom{4}{i} = 126$ different 3-bit in a 4-bit byte error patterns, so we need 126 syndromes to correct them. On the other hand, there are nine different byte error patterns with all four bits are corrupted, and we need just one more syndrome to detect them. Therefore the total number of syndromes required is 127, which is same as $2^7 - 1$. Since the code uses only 7 check bits, it is a perfect code.

7.3.3 $S_{t/B}$EC-S*b*EC-S*B*ED Codes

Today's high-density DRAM chips have a multi-bank architecture where each bank usually has a number of memory subarrays that are almost physically separated from each

$$\left[\begin{array}{ccccccccc}
1000 & 1000 & 1000 & 1000 & 1000 & 1000 & 1000 & 1000 & 1000 \\
\hline
1010 & 0011 & 0011 & 1111 & 1001 & 0011 & 1001 & 0000 & 1100 \\
0011 & 0101 & 0110 & 0110 & 1100 & 0011 & 0110 & 0011 & 0000 \\
0011 & 0011 & 1111 & 1001 & 0011 & 1001 & 0000 & 1100 & 0110 \\
0110 & 0101 & 1001 & 0101 & 0000 & 1111 & 0011 & 1100 & 1100 \\
0101 & 1010 & 0000 & 0110 & 1001 & 1111 & 1111 & 1010 & 0110 \\
0000 & 0011 & 0101 & 0110 & 0110 & 1100 & 0011 & 0110 & 0011
\end{array}\right]$$

Figure 7.9 *Example of a perfect (36, 29) $S_{3/4}$EC-S4ED code.* Source: *[UMAN03b]. © 2003 IEEE.*

other [NUMA89]. In particular, the binary bits stored in a memory subarray are highly independent of bits stored in other memory subarrays. It is therefore advantageous to consider the entire chip output as a B-bit block and subarray output as a b-bit byte [UMAN02]. Figure 7.10 illustrates these concepts in terms of the architecture of a recent 16 Mb high-density DRAM chip along with its corresponding organization of bit, byte, and block in a codeword.

In these memory chips, apart from multiple random bit errors (i.e., t/B-errors), errors caused by subarray data faults (i.e., b-bit byte errors) are a source concern too. Therefore, in addition to correcting multiple random bit errors corrupting a single memory chip, correction of errors caused by single subarray data faults is desired as well. In other words, an $S_{t/B}$EC-SBED code with SbEC capability (i.e., an $S_{t/B}$EC-SbEC-SBED code) is desirable in this situation.

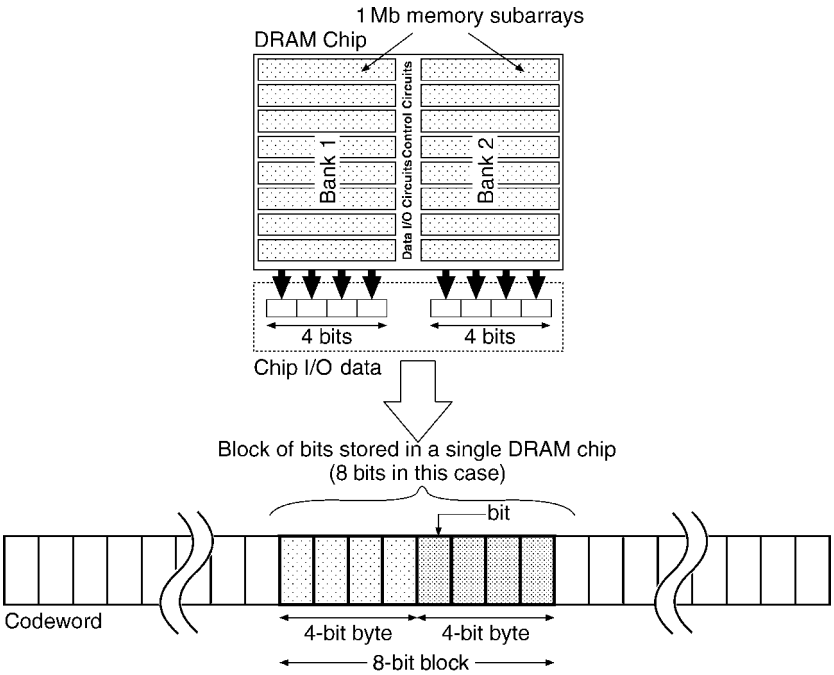


Figure 7.10 *Organization of the bit, byte, and block for a 16 Mb high-density DRAM chip with a nibbled-page architecture.* Source: *[NUMA89], [UMAN03b]. © 1989, and 2003 IEEE.*

Theorem 7.7 *For $t < b$, let $\boldsymbol{H}' = \boldsymbol{I}_B$. Also let $\boldsymbol{H}'' = \left[h''_0\ h''_1\ h''_2\ \cdots\ h''_{B-1}\right]$ represent a binary t-error detecting code such that any b column vectors in $\boldsymbol{H}''$ (i.e., h''_{bi}, h''_{bi+1}, h''_{bi+2}, $\cdots$, h''_{bi+b-1}) are linearly independent for $0 \le i < B/b$. Then the codes described by the parity-check matrices of Theorem 7.2, Lemma 7.1, and Theorem 7.3 are all systematic $S_{t/B}$ EC-SbEC-SBED codes.*

Proof We know that codes capable of correcting single t/B-errors for the case where $t \ge b$ are also capable of correcting *b-bit* byte errors. Hence it suffices to consider the case where $t < b$. Clearly, from Theorem 7.2 the code has $S_{t/B}$EC-S*B*ED capability. An additional condition is that linear independence of any b column vectors of $h''_{bi}, h''_{bi+1}, h''_{bi+2}, \cdots$, and h''_{bi+b-1}, ensures that the second part of the syndrome is nonzero for b-bit byte errors. Since the error pattern is given by the first B-bits of the syndrome, the second nonzero part locates the b-bit byte error position. Hence it is an $S_{t/B}$EC-SbEC-SBED code. Q.E.D.

The example code shown in Figure 7.5 is in fact capable of correcting 4-bit byte errors (i.e., it is an $S_{3/8}$EC-S4EC-S8ED code) because the $\mathbf{H}''$ matrix in this case is given by

$$\mathbf{H}'' = [\,\mathbf{H}_0 \mid \mathbf{H}_1\,] = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \end{array}\right]$$

$$= \left[\,\beta^0\ \beta^4\ \beta^8\ \beta^{14} \mid \beta^{10}\ \beta^{13}\ \beta^{12}\ \beta^7\,\right],$$

where α and β are primitive elements of $GF(2^3)$ and $GF(2^4)$, respectively. Clearly, any 4 column vectors of $\mathbf{H}_0$ and of $\mathbf{H}_1$ are linearly independent, as required by Theorem 7.7. Also note that the decoding method given in the previous section corrects b-bit byte errors. Figure 7.11 shows the check-bit length versus information-bit length relationship of the $S_{3/8}$EC-S4EC-S8ED codes, along with the S8EC codes and the $S_{3/8}$EC-S4EC-S8ED code bound.

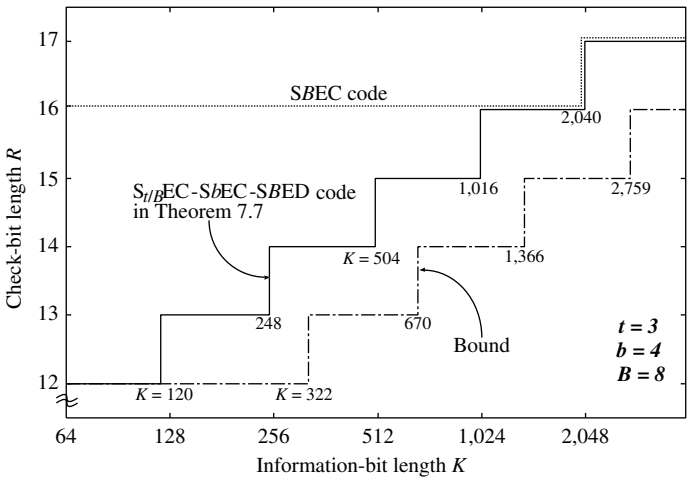


Figure 7.11 *Check-bit lengths compared with information-bit lengths of the $S_{3/8}$EC-S4EC-S8ED codes, along with the S8EC codes and the $S_{3/8}$EC-S4EC-S8ED code bound.* Source: *[UMAN03b]. © 2003 IEEE.*

7.4 SINGLE SPOTTY BYTE ERROR CORRECTING AND DOUBLE SPOTTY BYTE ERROR DETECTING ($S_{t/b}EC\text{-}D_{t/b}ED$) CODES

Here we study the $S_{t/b}EC\text{-}D_{t/b}ED$ codes that correct single t/b-errors and detect double t/b-errors. An $S_{t/b}EC\text{-}D_{t/b}ED$ code is primarily inspired by the architecture of Reed-Solomon SbEC-DbED code, denoted as RS SbEC-DbED code. Since the RS SbEC-DbED code has a strong error control function of single-byte error correction and double-byte error detection, this requires check-bit length equal to three times the length of a byte. In particular, in computer and communication systems that are prone only to a few transient bit errors in a byte, this code function becomes unnecessary.

7.4.1 Code Conditions and Bounds

Theorem 7.8 *Let $\boldsymbol{H}_i$ denote an $r \times b$ binary submatrix for $0 \le i \le n-1$. The null space of $\boldsymbol{H} = [\boldsymbol{H}_0 \boldsymbol{H}_1 \boldsymbol{H}_2 \boldsymbol{H}_3 \cdots \boldsymbol{H}_{n-1}]$ is an $S_{t/b}EC\text{-}D_{t/b}ED$ code, if and only if:*

1. $(E_1 + E_2) \cdot \boldsymbol{H}_i{}^T \neq 0 \quad$ *for* $E_1 \neq E_2$,
2. $E_1 \cdot \boldsymbol{H}_i{}^T \neq E_2 \cdot \boldsymbol{H}_j{}^T \quad$ *for* $i \neq j$,
3. $E_1 \cdot \boldsymbol{H}_i{}^T \neq (E_2 + E_3) \cdot \boldsymbol{H}_j{}^T \quad$ *for* $i \neq j$, $E_2 \neq E_3$,
4. $E_1 \cdot \boldsymbol{H}_i{}^T + E_2 \cdot \boldsymbol{H}_j{}^T \neq E_3 \cdot \boldsymbol{H}_k{}^T \quad$ *for* $i \neq j \neq k \neq i$,

where $0 \le i, j, k \le n-1$, $\forall E_1, E_2, E_3 \in \boldsymbol{E}_{t/b}$, $\boldsymbol{E}_{t/b} = \{E \in GF(2^b) | 1 \le w(E) \le t\}$.

Proof Conditions 1 and 2 confirm that all single t/b-error patterns within a b-bit byte generate unique nonzero syndromes. Hence these satisfy the conditions of an $S_{t/b}EC$ code. On the other hand, condition 3 together with condition 4 confirm that the syndrome generated by a single t/b-error is different from that generated by a double t/b-error. This asserts that double t/b-errors are detectable, and hence the code that satisfies these conditions is an $S_{t/b}EC\text{-}D_{t/b}ED$ code. Q.E.D.

Theorem 7.9 *A linear binary $S_{t/b}EC\text{-}D_{t/b}ED$ code requires at least $3t$ check bits.*

Proof According to condition 3, at least $3t$ binary columns of $\mathbf{H}$ (t columns each corresponding to the three t/b-errors) are linearly independent. Therefore a linear binary $S_{t/b}EC\text{-}D_{t/b}ED$ code requires at least $3t$ check bits. Q.E.D.

Theorem 7.10 *An $(N,\ N-R)$ $S_{t/b}EC\text{-}D_{t/b}ED$ code exists only if*

$$2^R - 1 \ge \frac{N}{b} \cdot \sum_{i=1}^{t} \binom{b}{i} + (2^t - 1) \cdot \left(\frac{N}{b} - 1\right) \cdot \sum_{j=1}^{t} \binom{b}{j}.$$

The proof of this theorem will be given in a generalized form in Subsection 7.5.1.

7.4.2 Design for $S_{t/b}EC\text{-}D_{t/b}ED$ Codes and $S_{t/b}EC\text{-}D_{t/b}ED\text{-}Sb ED$ Codes

Let $\mathbf{H}' = [h'_0\, h'_1 \cdots h'_{b-1}]$ be a $q \times b$ binary matrix ($q \le b$) whose at least $\min(3t, b)$ columns are linearly independent. Here $h'_0,\ h'_1,\ \cdots,\ h'_{b-1}$, are binary column vectors of

$GF(2^q)$. If $\min(3t,\ b) = b$, $\mathbf{H}'$ can be any $b \times b$ nonsingular matrix, including the $b \times b$ identity matrix. On the other hand, if $\min(3t,\ b) = 3t < b$, we consider $\mathbf{H}'$ to be a parity-check matrix of a linear binary $(b,\ b-q)$ code with minimum distance at least $3t+1$, (i.e., it is a parity-check matrix of a binary t-error correcting and $2t$-error detecting code).

Similarly let $\mathbf{H}'' = [h_0''\, h_1'' \cdots h_{b-1}'']$ be an $r \times b$ matrix with at least t columns that are linearly independent. Here $h_0'',\ h_1'',\ \cdots,\ h_{b-1}''$, are binary column vectors of $GF(2^r)$. If $t = b$, the matrix $\mathbf{H}''$ can be any $b \times b$ nonsingular matrix, including the $b \times b$ identity matrix. If $t < b$, we consider $\mathbf{H}''$ to be a parity-check matrix of a linear binary $(b,\ b-r)$ code with the minimum distance being at least $t+1$, which is to say, it is a parity-check matrix of a binary t-error detecting code.

We now use the $\mathbf{H}'$ and $\mathbf{H}''$ matrices defined above in the following theorem to design the $S_{t/b}$EC-$D_{t/b}$ED code.

Theorem 7.11 *Let γ be a primitive element of $GF(2^{(R-q)/2})$, where $R \geq b + 2r$. Let $\boldsymbol{H}_R$ be an $R \times b \cdot 2^{(R-q)/2}$ binary submatrix given by*

$$\boldsymbol{H}_R = \left[\begin{array}{ccccccc|c} \boldsymbol{H}' & \boldsymbol{H}' & \cdots & \boldsymbol{H}' & \cdots & \boldsymbol{H}' & \boldsymbol{H}' \\ \gamma^0\boldsymbol{H}'' & \gamma^1\boldsymbol{H}'' & \cdots & \gamma^i\boldsymbol{H}'' & \cdots & \gamma^{2^{(R-q)/2}-2}\boldsymbol{H}'' & \boldsymbol{O}_{((R-q)/2)\times b} \\ \gamma^0\boldsymbol{H}'' & \gamma^2\boldsymbol{H}'' & \cdots & \gamma^{2i}\boldsymbol{H}'' & \cdots & \gamma^{2(2^{(R-q)/2}-2)}\boldsymbol{H}'' & \boldsymbol{O}_{((R-q)/2)\times b} \end{array}\right],$$

where

$$\gamma^i\boldsymbol{H}'' = \left[\gamma^i\phi(h_0'') \quad \gamma^i\phi(h_1'') \quad \gamma^i\phi(h_2'') \quad \cdots \quad \gamma^i\phi(h_{b-1}'')\right]$$

for $0 \leq i \leq 2^{(R-q)/2} - 2$, and $\phi : GF(2^r) \rightarrow GF(2^{(R-q)/2})$ is a homomorphism of $GF(2^r)$ into $GF(2^{(R-q)/2})$ under addition. Then the null space of

$$\begin{aligned} \boldsymbol{H} &= [\boldsymbol{H}_R \mid \boldsymbol{I}_R] \\ &= \left[\begin{array}{cccccc|cc} \boldsymbol{H}' & \boldsymbol{H}' & \cdots & \boldsymbol{H}' & \cdots & \boldsymbol{H}' & \boldsymbol{H}' & \\ \gamma^0\boldsymbol{H}'' & \gamma^1\boldsymbol{H}'' & \cdots & \gamma^i\boldsymbol{H}'' & \cdots & \gamma^{2^{(R-q)/2}-2}\boldsymbol{H}'' & \boldsymbol{O}_{((R-q)/2)\times b} & \\ \gamma^0\boldsymbol{H}'' & \gamma^2\boldsymbol{H}'' & \cdots & \gamma^{2i}\boldsymbol{H}'' & \cdots & \gamma^{2(2^{(R-q)/2}-2)}\boldsymbol{H}'' & \boldsymbol{O}_{((R-q)/2)\times b} & \\ \multicolumn{6}{c|}{\begin{array}{cc} \boldsymbol{O}_{q\times((R-q)/2)} & \boldsymbol{O}_{q\times((R-q)/2)} \\ \boldsymbol{I}_{(R-q)/2} & \boldsymbol{O}_{((R-q)/2)\times((R-q)/2)} \\ \boldsymbol{O}_{((R-q)/2)\times((R-q)/2)} & \boldsymbol{I}_{(R-q)/2} \end{array}} & \end{array}\right] \end{aligned}$$

is an $S_{t/b}$EC-$D_{t/b}$ED code with check-bit length R and code length in bits $N = b \cdot 2^{(R-q)/2} + (R-q)$. Here $\boldsymbol{I}_x$ and $\boldsymbol{O}_{y\times z}$ denote a binary $x \times x$ identity matrix and a $y \times z$ all-zero matrix, respectively.

Theorem 7.12 *If $\boldsymbol{H}' = \boldsymbol{I}_b$ in Theorem 7.11, then the code is an $S_{t/b}$EC-$D_{t/b}$ED-SbED code with code length in bits $N = b \cdot 2^{(R-b)/2} + (R-b)$.*

It is left to the reader to prove the theorems above.

Example 7.2 $S_{3/8}EC\text{-}D_{3/8}ED$ code and $S_{3/8}EC\text{-}D_{3/8}ED\text{-}S8ED$ code.

We design a practical $S_{t/b}EC\text{-}D_{t/b}ED$ code, where $t = 3$ and $b = 8$, that is, an $S_{3/8}EC\text{-}D_{3/8}ED$ code. Since $\min(3t, b) = 8$, $\mathbf{H}'$ can be an 8×8 identity matrix. The matrix $\mathbf{H}''$ is a parity-check matrix of a distance-4 code, that is, a 3-bit error detecting code. In this example we use the following binary matrix as $\mathbf{H}''$.

$$\mathbf{H}'' = \begin{bmatrix} 11111111 \\ 01001011 \\ 00101110 \\ 00010111 \end{bmatrix}$$

By using these $\mathbf{H}'$ and $\mathbf{H}''$, we have a parity-check matrix of $(136, 120)$ $S_{3/8}EC\text{-}D_{3/8}ED$ code. According to Theorem 7.12, the matrix of $(136, 120)$ $S_{3/8}EC\text{-}D_{3/8}ED$ code, is shown in Figure 7.12. Since $\mathbf{H}' = \mathbf{I}_8$, i.e., 8×8 identity matrix, then the code shown in this figure is a $(136, 120)$ $S_{3/8}EC\text{-}D_{3/8}ED$ code.

It is noteworthy that when $t = 3$ and $b = 8$, the $S_{3/8}EC\text{-}D_{3/8}ED$ code can be designed with check-bit length $R = 15$, as shown in Figure 7.13. In this example, $\mathbf{H}'$ is given by

$$\mathbf{H}' = \begin{bmatrix} 10000001 \\ 01000001 \\ 00100001 \\ 00010001 \\ 00001001 \\ 00000101 \\ 00000011 \end{bmatrix},$$

which is a 7-bit error detecting code. The parity-check matrix given by Figure 7.13 still works as an $S_{3/8}EC\text{-}D_{3/8}ED$ code.

It can be investigated that the code shown in Figure 7.12 can detect double 3/8-errors occurred in any one byte as well as occurred in any different two bytes. The code shown in Figure 7.13, however, can detect double 3/8-errors occurred in different two bytes, but cannot detect these errors in any one byte. The former code is called an m-spotty byte error detecting code, and the latter is called an s-spotty byte error detecting code, which will be defined in the next Section 7.5.

Evaluation It is clear that if any b column vectors in the matrix $\mathbf{H}'$ are linearly independent, the $S_{t/b}EC\text{-}D_{t/b}ED$ code possesses an extra function of single-byte error detection, which is the $S_{t/b}EC\text{-}D_{t/b}ED\text{-}Sb ED$ code. This argument naturally holds for any value of b.

A graphic relationship between the check-bit lengths and the information-bit lengths for the $S_{t/8}EC\text{-}D_{t/8}ED$ codes and $S_{t/8}EC\text{-}D_{t/8}ED\text{-}S8ED$ codes is provided in Figure 7.14. Compared to the 24 check bits required by the S8EC-D8ED code, the $S_{3/8}EC\text{-}D_{3/8}ED$ code and the $S_{3/8}EC\text{-}D_{3/8}ED\text{-}S8ED$ code require only 15 and 16 check bits, respectively, for the practical information length of 64 bits. Furthermore, for other practical information lengths of 128 and 256 bits, these codes require fewer check bits than the S8EC-D8ED code. However the $S_{3/8}EC\text{-}D_{3/8}ED$ codes presented in Theorem 7.11 require a large number of check bits compared to the bound shown in Theorem 7.10, especially for large information-bit lengths.

$$
\left[\begin{array}{c}
10000000\ 10000000\ 10000000\ 10000000\ 10000000\ 10000000\ 10000000\ 10000000\ 10000000\ 10000000\ 10000000\ 10000000\ 10000000\ 10000000\ 10000000\ 10000000\ 00000000 \\
01000000\ 01000000\ 01000000\ 01000000\ 01000000\ 01000000\ 01000000\ 01000000\ 01000000\ 01000000\ 01000000\ 01000000\ 01000000\ 01000000\ 01000000\ 01000000\ 00000000 \\
00100000\ 00100000\ 00100000\ 00100000\ 00100000\ 00100000\ 00100000\ 00100000\ 00100000\ 00100000\ 00100000\ 00100000\ 00100000\ 00100000\ 00100000\ 00100000\ 00000000 \\
00010000\ 00010000\ 00010000\ 00010000\ 00010000\ 00010000\ 00010000\ 00010000\ 00010000\ 00010000\ 00010000\ 00010000\ 00010000\ 00010000\ 00010000\ 00010000\ 00000000 \\
00001000\ 00001000\ 00001000\ 00001000\ 00001000\ 00001000\ 00001000\ 00001000\ 00001000\ 00001000\ 00001000\ 00001000\ 00001000\ 00001000\ 00001000\ 00001000\ 00000000 \\
00000100\ 00000100\ 00000100\ 00000100\ 00000100\ 00000100\ 00000100\ 00000100\ 00000100\ 00000100\ 00000100\ 00000100\ 00000100\ 00000100\ 00000100\ 00000100\ 00000000 \\
00000010\ 00000010\ 00000010\ 00000010\ 00000010\ 00000010\ 00000010\ 00000010\ 00000010\ 00000010\ 00000010\ 00000010\ 00000010\ 00000010\ 00000010\ 00000010\ 00000000 \\
00000001\ 00000001\ 00000001\ 00000001\ 00000001\ 00000001\ 00000001\ 00000001\ 00000001\ 00000001\ 00000001\ 00000001\ 00000001\ 00000001\ 00000001\ 00000001\ 00000000 \\
\hline
11111111\ 00010111\ 00101110\ 01001011\ 11101000\ 00111001\ 01100101\ 10100011\ 11010001\ 01011100\ 11000110\ 01110010\ 10001101\ 10011010\ 10110100\ 00000000\ 10000000 \\
01001011\ 11101000\ 00111001\ 01100101\ 10100011\ 11010001\ 01011100\ 11000110\ 01110010\ 10001101\ 10011010\ 10110100\ 11111111\ 00010111\ 00101110\ 00000000\ 01000000 \\
00101110\ 01001011\ 11101000\ 00111001\ 01100101\ 10100011\ 11010001\ 01011100\ 11000110\ 01110010\ 10001101\ 10011010\ 10110100\ 11111111\ 00010111\ 00000000\ 00100000 \\
00010111\ 00101110\ 01001011\ 11101000\ 00111001\ 01100101\ 10100011\ 11010001\ 01011100\ 11000110\ 01110010\ 10001101\ 10011010\ 10110100\ 11111111\ 00000000\ 00010000 \\
\hline
11111111\ 00101110\ 11101000\ 01100101\ 11010001\ 11000110\ 10001101\ 10110100\ 00010111\ 01001011\ 00111001\ 10100011\ 01011100\ 01110010\ 10011010\ 00000000\ 00001000 \\
01001011\ 00111001\ 10100011\ 01011100\ 01110010\ 10011010\ 11111111\ 00101110\ 11101000\ 01100101\ 11010001\ 11000110\ 10001101\ 10110100\ 00010111\ 00000000\ 00000100 \\
00101110\ 11101000\ 01100101\ 11010001\ 11000110\ 10001101\ 10110100\ 00010111\ 01001011\ 00111001\ 10100011\ 01011100\ 01110010\ 10011010\ 11111111\ 00000000\ 00000010 \\
00010111\ 01001011\ 00111001\ 10100011\ 01011100\ 01110010\ 10011010\ 11111111\ 00101110\ 11101000\ 01100101\ 11010001\ 11000110\ 10001101\ 10110100\ 00000000\ 00000001
\end{array}\right]
$$

Figure 7.12 *(136, 120)$S_{3/8}$EC-$D_{3/8}$ED-S8ED code.*

10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	00000000
01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	00000000
00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00000000
00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00000000
00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00000000
00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000000
00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000000
11111111	00010111	00101110	01001011	11101000	00111001	01100101	10100011	11010001	01011100	11000110	01110010	10001101	10011010	10110100	00000000	10000000
01001011	11101000	00111001	01100101	10100011	11010001	01011100	11000110	01110010	10001101	10011010	10110100	11111111	00010111	00101110	00000000	01000000
00101110	01001011	11101000	00111001	01100101	10100011	11010001	01011100	11000110	01110010	10001101	10011010	10110100	11111111	00010111	00000000	00100000
00010111	00101110	01001011	11101000	00111001	01100101	10100011	11010001	01011100	11000110	01110010	10001101	10011010	10110100	11111111	00000000	00010000
11111111	00101110	11101000	01100101	11010001	11000110	10001101	10110100	00010111	01001011	00111001	10100011	01011100	01110010	10011010	00000000	00001000
01001011	00111001	10100011	01011100	01110010	10011010	11111111	00101110	11101000	01100101	11010001	11000110	10001101	10110100	00010111	00000000	00000100
00101110	11101000	01100101	11010001	11000110	10001101	10110100	00010111	01001011	00111001	10100011	01011100	01110010	10011010	11111111	00000000	00000010
00010111	01001011	00111001	10100011	01011100	01110010	10011010	11111111	00101110	11101000	01100101	11010001	11000110	10001101	10110100	00000000	00000001

Figure 7.13 *(136, 121)$S_{3/8}EC$-$D_{3/8}ED$ code.*

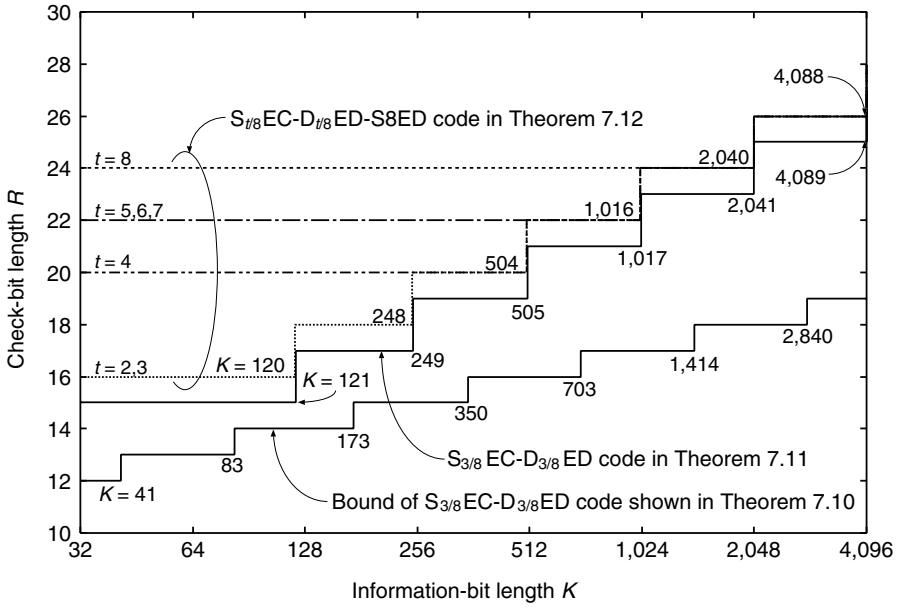


Figure 7.14 *Comparison of check-bit lengths and information-bit lengths of the $S_{t/8}EC\text{-}D_{t/8}ED$ codes and the $S_{t/8}EC\text{-}D_{t/8}ED\text{-}S8ED$ codes.*

The error detection capabilities of the $S_{3/8}EC\text{-}D_{3/8}ED$ code are shown in Table 7.2 for three types of errors: random triple-bit errors, random quadruple-bit errors, and single 4-bit byte plus single-bit errors. These errors are outside the error control capability of the $S_{3/8}EC\text{-}D_{3/8}ED$ code. In the table "byte plus bit errors" means that single 4-bit byte errors and single-bit errors are occurred simultaneously. The error detection capabilities of the $S_{3/8}EC\text{-}D_{3/8}ED\text{-}S8ED$ code are shown in Table 7.3.

TABLE 7.2 Error Detection Capabilities of the $S_{3/8}EC\text{-}D_{3/8}ED$ Code

	Error detection capability (%)		
Errors	$K = 64$ ($R = 15$)	$K = 128$ ($R = 17$)	$K = 256$ ($R = 19$)
Triple-bit errors	97.57	98.34	98.76
Quadruple-bit errors	98.19	99.07	99.53
Byte plus bit errors	94.25	94.03	94.07

TABLE 7.3 Error Detection Capabilities of the $S_{3/8}EC\text{-}D_{3/8}ED\text{-}S8ED$ Code

	Error detection capability (%)		
Errors	$K = 64$ ($R = 16$)	$K = 128$ ($R = 18$)	$K = 256$ ($R = 20$)
Triple-bit errors	97.47	98.22	98.73
Quadruple-bit errors	96.45	97.96	98.65
Byte plus bit errors	88.95	90.14	90.36

7.5 A GENERAL CLASS OF SPOTTY BYTE ERROR CONTROL CODES

In Section 7.1 we saw that when a small number of random bit errors collect in a byte, we have a situation called a *spotty byte error*. From a generalized and theoretical code design standpoint, the single spotty errors in a byte are called *s-spotty byte errors*, and the multiple spotty errors in a byte are called *m-spotty byte errors*. In this section the code design for generalized s-spotty and m-spotty byte error control codes over $GF(2^b)$ is discussed [KASH04, SUZU04, 05a, 05b].

7.5.1 A General Class of Codes for s-Spotty Byte Errors

The s-spotty byte error has been defined as a set of t or fewer bits errors confined to a b-bit byte. In this case the maximum number of erroneous bits in each byte does not exceed $t(< b)$. Below we present a general class of s-spotty byte error control codes [KASH04].

1. s-Spotty Byte Error Control Codes

Preliminaries

Definition 7.1 An error is called an *s-spotty byte error* if a set of random t or fewer bits errors is confined to a byte, meaning the maximum number of erroneous bits in a byte does not exceed t. □

The necessary and sufficient conditions of the s-spotty byte error control codes are presented as follows.

Theorem 7.13 *Let $\boldsymbol{H}_i$ be an $R \times b$ binary submatrix for $0 \le i \le n-1$, and also let $\boldsymbol{E}_{t/b} = \{E \in GF(2^b) \mid 1 \le w(E) \le t\}$ be a set of all t/b-error patterns in a b-bit byte where $w(E)$ denotes the Hamming weight of b-bit vector E. The null space of $\boldsymbol{H} = [\boldsymbol{H}_0 \boldsymbol{H}_1 \boldsymbol{H}_2 \boldsymbol{H}_3 \cdots \boldsymbol{H}_{n-1}]$ is a λ t/b-errors correcting and μ t/b-errors detecting code if and only if*

$$(E_1 + E_2) \cdot \boldsymbol{H}_{i_1}^T + \cdots + (E_{2v-1} + E_{2v}) \cdot \boldsymbol{H}_{i_v}^T + E_{2v+1} \cdot \boldsymbol{H}_{i_{v+1}}^T + \cdots + E_{2v+w} \cdot \boldsymbol{H}_{i_{v+w}}^T \neq 0_R$$
$$\text{for} \quad 2v + w \le \lambda + \mu, 0 \le v \le \lambda,\ 0 \le w \le \lambda + \mu,$$

where $\mu \ge \lambda$, $\forall E_1, E_2, \ldots, E_{2v}, E_{2v+1}, \ldots, E_{2v+w} \in \boldsymbol{E}_{t/b}$, $i_1, i_2, \ldots, i_v, i_{v+1}, \ldots, i_{v+w}$ are distinct integers of i satisfying $0 \le i_1, i_2, \cdots, i_v, i_{v+1}, \ldots, i_{v+w} \le n-1$, 0_R is an R-bit zero vector, and T is a transpose of vector or matrix.

Proof Let two sets having $\rho(\le \lambda)$ and $\sigma(\le \mu)$ s-spotty byte errors be $\mathbf{E}_i = \{E_{i_1}, E_{i_2}, \cdots, E_{i_\rho}\}$ and $\mathbf{E}_j = \{E_{j_1}, E_{j_2}, \cdots, E_{j_\sigma}\}$, respectively. In each set, ρ s-spotty byte errors are assumed to have occurred in the different ρ bytes and σ s-spotty byte errors in the different σ bytes. For λ t/b-errors correcting and μ t/b-errors detecting code, the following relation should be satisfied:

$$E_{i_1} \cdot \mathbf{H}_{i_1}^T + E_{i_2} \cdot \mathbf{H}_{i_2}^T + \cdots + E_{i_\rho} \cdot \mathbf{H}_{i_\rho}^T \neq E_{j_1} \cdot \mathbf{H}_{j_1}^T + E_{j_2} \cdot \mathbf{H}_{j_2}^T + \cdots + E_{j_\sigma} \cdot \mathbf{H}_{j_\sigma}^T.$$

Without loss of generality, say two s-spotty byte errors, occur in the same byte such as in the x-th byte, $E_{i_x} \in \mathbf{E}_i$ and $E_{j_x} \in \mathbf{E}_j$. Now assume that this type of errors occurs in v bytes, where $0 \le v \le \lambda$. Then the following relation holds:

$$\begin{aligned} &E_{i_1} \cdot \mathbf{H}_{i_1}^T + E_{i_2} \cdot \mathbf{H}_{i_2}^T + \cdots + E_{i_v} \cdot \mathbf{H}_{i_v}^T + E_{i_{v+1}} \cdot \mathbf{H}_{i_{v+1}}^T + \cdots + E_{i_\rho} \cdot \mathbf{H}_{i_\rho}^T \\ &\quad \neq E_{j_1} \cdot \mathbf{H}_{i_1}^T + E_{j_2} \cdot \mathbf{H}_{i_2}^T + \cdots + E_{j_v} \cdot \mathbf{H}_{i_v}^T + E_{j_{v+1}} \cdot \mathbf{H}_{j_{v+1}}^T + \cdots + E_{j_\sigma} \cdot \mathbf{H}_{j_\sigma}^T. \end{aligned}$$

Consequently, we have

$$\begin{aligned} &(E_{i_1} + E_{j_1}) \cdot \mathbf{H}_{i_1}^T + (E_{i_2} + E_{j_2}) \cdot \mathbf{H}_{i_2}^T + \cdots + (E_{i_v} + E_{j_v}) \cdot \mathbf{H}_{i_v}^T \\ &\quad + E_{i_{v+1}} \cdot \mathbf{H}_{i_{v+1}}^T + \cdots + E_{i_\rho} \cdot \mathbf{H}_{i_\rho}^T + E_{j_{v+1}} \cdot \mathbf{H}_{j_{v+1}}^T + \cdots + E_{j_\sigma} \cdot \mathbf{H}_{j_\sigma}^T \neq 0. \end{aligned}$$

If $\rho + \sigma - 2v = w$, then the relation in Theorem 7.13 holds. Q.E.D.

Theorem 7.14 *A linear λ t/b-errors correcting and μ t/b-errors detecting code requires at least $(\lambda + \mu)t$ check bits.*

Proof From Theorem 7.13, the $(\lambda + \mu)t$ binary columns of the parity-check matrix $\mathbf{H}$ should be linearly independent. Therefore a linear λ t/b-errors correcting and μ t/b-errors detecting code requires at least $(\lambda + \mu)t$ check bits. Q.E.D.

Theorem 7.15 *If the code length N is a multiple of the byte length b, a linear $(N,\ N - R)\lambda$ t/b-errors correcting code exists only if*

$$2^R - 1 \ge \sum_{i=1}^{\lambda} \left\{ \binom{N/b}{i} \cdot \left\{ \sum_{j=1}^{t} \binom{b}{j} \right\}^i \right\}. \tag{7.9}$$

Proof The total number of t/b-error is given by $\sum_{j=1}^{t} \binom{b}{j}$. There are N/b bytes in a codeword with N bit lengths. Therefore we need $\binom{N/b}{i} \cdot \left\{ \sum_{j=1}^{t} \binom{b}{j} \right\}^i$ different syndrome patterns to correct all i t/b-error patterns. The i can take any value from 1 to λ, and hence the total number of different nonzero syndromes necessary to correct up to λ t/b-errors can be expressed as

$$\sum_{i=1}^{\lambda} \left\{ \binom{N/b}{i} \cdot \left\{ \sum_{j=1}^{t} \binom{b}{j} \right\}^i \right\}.$$

Clearly, the inequality in Theorem 7.15 holds. Q.E.D.

Theorem 7.16 *If a code length N is a multiple of a byte length b, a linear $(N,\ N - R)$ λ t/b-errors correcting and $(\lambda + 1)$ t/b-errors detecting code exists only if*

$$2^R - 1 \ge \sum_{i=1}^{\lambda} \left\{ \binom{N/b}{i} \cdot \left\{ \sum_{j=1}^{t} \binom{b}{j} \right\}^i \right\} + (2^t - 1) \cdot \binom{N/b - 1}{\lambda} \cdot \left\{ \sum_{j=1}^{t} \binom{b}{j} \right\}^{\lambda}. \tag{7.10}$$

Proof The total number of different nonzero syndromes to correct up to λ t/b-errors can be expressed as $\sum_{i=1}^{\lambda}\left\{\binom{N/b}{i}\cdot\left\{\sum_{j=1}^{t}\binom{b}{j}\right\}^{i}\right\}$.

For the number of extra syndromes caused by $(\lambda+1)$ t/b-errors, We can assume without loss of generality that a single t/b-error E_0 has occurred in the first byte. Then λ t/b-errors E_{i_1}, E_{i_2}, $\cdots$, E_{i_λ}, have occurred in the remaining $N/b-1$ bytes. Now we consider $(\lambda+1)$ t/b-errors, including the error E_0 in the first byte, that is, E_0, E_{i_1}, E_{i_2}, $\cdots$, and E_{i_λ}. We also consider another $(\lambda+1)$ t/b-errors including the error E_0' in the first byte, that is, E_0', E_{j_1}, E_{j_2}, $\cdots$, and E_{j_λ}. The syndrome caused by all these errors can be assumed to satisfy the following relation:

$$\begin{aligned}&\left(E_0\cdot\mathbf{H}_0^T+E_{i_1}\cdot\mathbf{H}_{i_1}^T+E_{i_2}\cdot\mathbf{H}_{i_2}^T+\cdots+E_{i_\lambda}\cdot\mathbf{H}_{i_\lambda}^T\right)\\&\quad+\left(E_0'\cdot\mathbf{H}_0^T+E_{j_1}\cdot\mathbf{H}_{j_1}^T+E_{j_2}\cdot\mathbf{H}_{j_2}^T+\cdots+E_{j_\lambda}\cdot\mathbf{H}_{j_\lambda}^T\right)=0_R^T.\end{aligned}$$

That is,

$$\begin{aligned}&(E_0+E_0')\cdot\mathbf{H}_0^T+\left\{\left(E_{i_1}\cdot\mathbf{H}_{i_1}^T+E_{i_2}\cdot\mathbf{H}_{i_2}^T+\cdots+E_{i_\lambda}\cdot\mathbf{H}_{i_\lambda}^T\right)\right.\\&\quad\left.+\left(E_{j_1}\cdot\mathbf{H}_{j_1}^T+E_{j_2}\cdot\mathbf{H}_{j_2}^T+\cdots+E_{j_\lambda}\cdot\mathbf{H}_{j_\lambda}^T\right)\right\}=0_R^T.\end{aligned}$$

This equation says that the total $2\lambda+1$ syndrome sum caused by $(2\lambda+1)$ t/b-errors, including the one in the first byte, is equal to zero. This contradicts the necessary and sufficient condition in Theorem 7.13.

Hence the syndrome sum of $(\lambda+1)$ t/b-errors should not be equal to the sum of any other $(\lambda+1)$ t/b-errors, including the same one t/b-error. The number of distinct t bits errors in the fixed byte is counted as 2^t-1. Therefore the number of the extra syndromes necessary for detecting distinct $(\lambda+1)$ t/b-errors is calculated as

$$(2^t-1)\cdot\binom{N/b-1}{\lambda}\cdot\left\{\sum_{j=1}^{t}\binom{b}{j}\right\}^{\lambda}.$$

Therefore the inequality in Theorem 7.16 holds. Q.E.D.

Code Design First, we design the s-spotty byte error control codes by using the tensor product.

Definition 7.2 Let $\mathbf{H}'=[h_0'\,h_1'\,\cdots\,h_{b-1}']$ be an $r\times b$ binary matrix whose $\min(2t,\ b)$ column vectors are linearly independent, where $h_0',\ h_1',\ \cdots, h_{b-1}'$ are the binary column vectors of $GF(2^r)$. Here $\min(x, y)$ means that if $x<y$, then $\min(x, y)=x$, and if $x\geq y$, then $\min(x, y)=y$. □

If $\min(2t, b)=b$, the matrix $\mathbf{H}'$ can be any $b\times b$ nonsingular matrix, including the $b\times b$ identity matrix. On the other hand, if $\min(2t, b)=2t<b$, the matrix $\mathbf{H}'$ is a parity-

check matrix of a linear binary $(b,\ b-r)$ code with minimum Hamming distance $2t+1$, meaning a parity-check matrix of binary t-error correcting code.

In using the above-defined $\mathbf{H}'$, the following theorem outlines the design of the λ s-spotty byte errors correcting and μ s-spotty byte errors detecting code.

Theorem 7.17 *Let γ be a primitive element in $GF(2^r)$. Let $\boldsymbol{H}''$ be a distance-$(\lambda+\mu+1)$ Reed-Solomon code over $GF(2^r)$. In terms of γ, the matrix $\boldsymbol{H}''$ is expressed as follows:*

$$\boldsymbol{H}'' = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 & 0 \\ \gamma^0 & \gamma^1 & \gamma^2 & \cdots & \gamma^{n-1} & 0 & 0 \\ \gamma^0 & \gamma^2 & \gamma^4 & \cdots & \gamma^{2(n-1)} & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ \gamma^0 & \gamma^{\lambda+\mu-1} & \gamma^{2(\lambda+\mu-1)} & \cdots & \gamma^{(\lambda+\mu-1)(n-1)} & 0 & 1 \end{bmatrix},$$

where $\gamma^0 = 1$ is the unit element of $GF(2^r)$ and 0 is the zero element of $GF(2^r)$. Then the null space of

$$\boldsymbol{H} = \boldsymbol{H}'' \otimes \boldsymbol{H}'$$

is a λ t/b-errors correcting and μ t/b-errors detecting code with a check-bit length $R = r(\lambda+\mu)$ and a code length in bits $N = b(n+2)$, where $\lambda \le \mu$, $n = 2^r - 1$, and

$$\gamma^i \boldsymbol{H}' = \begin{bmatrix} \gamma^i h'_0 & \gamma^i h'_1 & \cdots & \gamma^i h'_{b-1} \end{bmatrix}$$

for $0 \le i \le n-1$. Here $\gamma^i h'_j$ means the product of γ^i and h'_j over $GF(2^r)$.

Proof The following shows how the code presented in this theorem satisfies the condition of Theorem 7.13. Without loss of generality, we assume that Eq. (7.11) holds for $E_1 + E_2 \ne 0,\ \ldots,\ E_{2v-1} + E_{2v} \ne 0$. Here 0_r means an r-bit zero vector

$$(E_1+E_2)\cdot\begin{bmatrix} \mathbf{H}' \\ \gamma^{i_1}\mathbf{H}' \\ \gamma^{2i_1}\mathbf{H}' \\ \vdots \\ \gamma^{(\lambda+\mu-1)i_1}\mathbf{H}' \end{bmatrix}^T + \cdots + (E_{2v-1}+E_{2v})\cdot\begin{bmatrix} \mathbf{H}' \\ \gamma^{i_v}\mathbf{H}' \\ \gamma^{2i_v}\mathbf{H}' \\ \vdots \\ \gamma^{(\lambda+\mu-1)i_v}\mathbf{H}' \end{bmatrix}^T$$

$$+ E_{2v+1}\cdot\begin{bmatrix} \mathbf{H}' \\ \gamma^{i_{v+1}}\mathbf{H}' \\ \gamma^{2i_{v+1}}\mathbf{H}' \\ \vdots \\ \gamma^{(\lambda+\mu-1)i_{v+1}}\mathbf{H}' \end{bmatrix}^T + \cdots + E_{2v+w}\cdot\begin{bmatrix} \mathbf{H}' \\ \gamma^{i_{v+w}}\mathbf{H}' \\ \gamma^{2i_{v+w}}\mathbf{H}' \\ \vdots \\ \gamma^{(\lambda+\mu-1)i_{v+w}}\mathbf{H}' \end{bmatrix}^T = \begin{bmatrix} 0_r^T \\ 0_r^T \\ 0_r^T \\ \vdots \\ 0_r^T \end{bmatrix}. \quad (7.11)$$

This equation does not include the last and second last column of $\mathbf{H}$. Multiplying γ^0 to both sides of first row in Eq. (7.11) comes to

$$\gamma^0\left((E_1+E_2)\cdot\mathbf{H}'^T\right)+\cdots+\gamma^0\left((E_{2v-1}+E_{2v})\cdot\mathbf{H}'^T\right)$$
$$+\gamma^0\left(E_{2v+1}\cdot\mathbf{H}'^T\right)+\cdots+\gamma^0\left(E_{2v+w}\cdot\mathbf{H}'^T\right)=\gamma^0\cdot(0_r^T)=0_{r'}^T. \tag{7.12}$$

Now we let $(E_1+E_2)\cdot\mathbf{H}'^T$, $\cdots$, $(E_{2v-1}+E_{2v})\cdot\mathbf{H}'^T$, $E_{2v+1}\cdot\mathbf{H}'^T$, $\cdots$, and $E_{2v+w}\cdot\mathbf{H}'^T$ be x_1, $\cdots$, x_v, x_{v+1}, $\cdots$, and x_{v+w}, respectively. Since $\mathbf{H}'$ is a $r\times b$ matrix whose $\min(2t, b)$ column vectors are linearly independent, $x_1\neq 0$, $\cdots$, $x_v\neq 0$, $x_{v+1}\neq 0$, $\cdots$, $x_{v+w}\neq 0$. From Eqs. (7.11) and (7.12), the following relations hold:

$$\begin{aligned}
&\gamma^0x_1+\gamma^0x_2+\cdots+\gamma^0x_{v+w}=0,\\
&\gamma^{i_1}x_1+\gamma^{i_2}x_2+\cdots+\gamma^{i_{v+w}}x_{v+w}=0,\\
&\gamma^{2i_1}x_1+\gamma^{2i_2}x_2+\cdots+\gamma^{2i_{v+w}}x_{v+w}=0,\\
&\qquad\qquad\cdots\\
&\gamma^{(\lambda+\mu-1)i_1}x_1+\gamma^{(\lambda+\mu-1)i_2}x_2+\cdots+\gamma^{(\lambda+\mu-1)i_{v+w}}x_{v+w}=0.
\end{aligned}$$

The coefficient matrix of the top $v+w(\leq\lambda+\mu)$ relations is nonsingular because its determinant is a Vandermonde's determinant. Multiplying the inverse matrix of this $(v+w)\times(v+w)$ matrix to the coefficient matrix from the left side yields

$$[x_1\quad x_2\quad x_3\quad\cdots\quad x_{v+w}]^T=[0\quad 0\quad 0\quad\cdots\quad 0]^T,$$

which is a contradiction because $x_1\neq 0, x_2\neq 0,\cdots$, and $x_{v+w}\neq 0$. The condition of Theorem 7.13 is now satisfied, and therefore the matrix $\mathbf{H}$ represents a parity-check matrix of a λ t/b-errors correcting and μ t/b-errors detecting code. It is apparent that the check-bit length takes $R=r(\lambda+\mu)$ and the code length in bits takes $N=b(n+2)=b(2^r+1)$. Q.E.D.

Note that if $b/2\leq t\leq b$, the code shown in Theorem 7.17 is identical to the maximum distance separable (MDS) RS code over $GF(2^b)$ because $\mathbf{H}'$ is equal to the $b\times b$ identity matrix.

Lemma 7.3 *Let $\boldsymbol{H}'=[h'_0\,h'_1\,\cdots\,h'_{b-1}]$ be an $r\times b$ binary matrix whose p or fewer column vectors are linearly independent, and let ϕ be an injective homomorphism of $GF(2^r)$ into $GF(2^{r'})$ under addition. Then any p column vectors in $\phi(\boldsymbol{H}')=[\phi(h'_0)\,\phi(h'_1)\,\cdots\,\phi(h'_{b-1})]$ are linearly independent.*

Proof First, if ϕ is a injective homomorphism of additive group, then $\mathrm{Ker}(\phi)=\{0\}$, where $\mathrm{Ker}(\phi)$ is the *kernel* of ϕ. For all $\beta\in GF(2^r)$, the following relation holds because ϕ is an injective homomorphism under addition:

$$\phi(0)+\phi(\beta)=\phi(0+\beta)=\phi(\beta).$$

Therefore $\phi(0)=0$ is obtained. Since ϕ is injective, then $\mathrm{Ker}(\phi)=\{0\}$.

Next we assume that σ $(\leq p)$ column vectors in $\phi(\mathbf{H}')$ are linearly dependent, that is, that the following relation holds for distinct integers $i_1,\ i_2,\ \cdots,\ i_\sigma$, where $0 \leq i_1,\ i_2,\ \cdots, i_\sigma \leq b-1$:

$$\phi(h_{i_1}) + \phi(h_{i_2}) + \cdots + \phi(h_{i_\sigma}) = 0. \tag{7.13}$$

Since ϕ is an injective homomorphism under addition, we have

$$\phi(h_{i_1}) + \phi(h_{i_2}) + \cdots + \phi(h_{i_\sigma}) = \phi(h_{i_1} + h_{i_2} + \cdots + h_{i_\sigma}). \tag{7.14}$$

From Eqs. (7.13) and (7.14),

$$h_{i_1} + h_{i_2} + \cdots + h_{i_\sigma} = 0.$$

This contradicts that σ column vectors of the matrix $\mathbf{H}'$ are linearly independent. Hence any p column vectors in $\phi(\mathbf{H}')$ are linearly independent. Q.E.D.

Lemma 7.4 *Let $\boldsymbol{H}' = [h'_0\, h'_1\, \cdots\, h'_{b-1}]$ be an $r \times b$ matrix, γ be a primitive element of $GF(2^r)$, ϕ be an injective homomorphism of $GF(2^r)$ into $GF(2^{r'})$ under addition, and $\gamma^i\boldsymbol{H}' = [\gamma^i\phi(h'_0)\,\gamma^i\phi(h'_1)\,\cdots\,\gamma^i\phi(h'_{b-1})]$. Then the following relation holds for a vector $E_j = [E_{j_0}\, E_{j_1}\, \cdots\, E_{j_{b-1}}]$:*

$$E_j \cdot \left(\gamma^i\boldsymbol{H}'\right)^T = \gamma^i\phi\left(E_j \cdot \boldsymbol{H}'^T\right).$$

Proof The term $E_j \cdot (\gamma^i\mathbf{H}')^T$ can be calculated as follows:

$$\begin{aligned} E_j \cdot \left(\gamma^i\mathbf{H}'\right)^T &= E_{j_0}\gamma^i\phi(h_0) + E_{j_1}\gamma^i\phi(h_1) + \cdots + E_{j_{b-1}}\gamma^i\phi(h_{b-1}) \\ &= \gamma^i\left(E_{j_0}\phi(h_0) + E_{j_1}\phi(h_1) + \cdots + E_{j_{b-1}}\phi(h_{b-1})\right), \end{aligned}$$

where $E_{j_p}\phi(h_p)(0 \leq p \leq b-1)$ represents a vector $\phi(h_p)$ multiplied by a scalar $E_{j_p} \in GF(2)$. On the other hand, $\gamma^i\phi\left(E_j \cdot \mathbf{H}'^T\right)$ can be calculated as follows:

$$\gamma^i\phi\left(E_j \cdot \mathbf{H}'^T\right) = \gamma^i\left(\phi(E_{j_0}h_0) + \phi(E_{j_1}h_1) + \cdots + \phi(E_{j_{b-1}}h_{b-1})\right).$$

Here, if $E_{j_p} = 0$, the following relation holds for $E_{j_p}\phi(h_p)$ and $\phi(E_{j_p}h_p)$, where $0 \leq p \leq b-1$:

$$\begin{aligned} E_{j_p}\phi(h_p) &= 0\phi(h_p) = 0_{r'}, \\ \phi(E_{j_p}h_p) &= \phi(0h_p) = \phi(0_r) = 0_{r'}. \end{aligned}$$

If $E_{j_p} = 1$, the following relation holds:

$$\begin{aligned} E_{j_p}\phi(h_p) &= 1\phi(h_p) = \phi(h_p), \\ \phi(E_{j_p}h_p) &= \phi(1h_p) = \phi(h_p). \end{aligned}$$

Therefore $E_{j_p}\phi(h_p) = \phi(E_{j_p}h_p)$ for $E_{j_p} \in GF(2)$.

From the above we end up with

$$E_j \cdot \left(\gamma^i \mathbf{H}'\right)^T = \gamma^i \phi\left(E_j \cdot \mathbf{H}'^T\right),$$

as required by Lemma 7.4.

The following theorem presents the lengthened codes. This can further lengthen the code by taking r' being larger than the original r.

Theorem 7.18 *Let γ be a primitive element in $GF(2^{r'})$, where $r' \geq r$. The null space of*

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{H}' & \boldsymbol{H}' & \cdots & \boldsymbol{H}' & \boldsymbol{H}' & \boldsymbol{O} \\ \gamma^0\boldsymbol{H}' & \gamma^1\boldsymbol{H}' & \cdots & \gamma^{n-1}\boldsymbol{H}' & \boldsymbol{O} & \boldsymbol{O} \\ \gamma^0\boldsymbol{H}' & \gamma^2\boldsymbol{H}' & \cdots & \gamma^{2(n-1)}\boldsymbol{H}' & \boldsymbol{O} & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \gamma^0\boldsymbol{H}' & \gamma^{\lambda+\mu-1}\boldsymbol{H}' & \cdots & \gamma^{(\lambda+\mu-1)(n-1)}\boldsymbol{H}' & \boldsymbol{O} & \gamma^0\boldsymbol{H}' \end{bmatrix}$$

is a lengthened λ t/b-errors correcting and μ t/b-errors detecting code with check-bit length $R = r + r'(\lambda + \mu - 1)$ and code length in bits $N = b(n + 2)$, where $\boldsymbol{H}'$ is an $r \times b$ binary matrix defined in Definition 7.2, $n = 2^{r'} - 1$, $\boldsymbol{O}$ is zero matrix, and

$$\gamma^i\boldsymbol{H}' = \begin{bmatrix} \gamma^i\phi(h'_0) & \gamma^i\phi(h'_1) & \cdots & \gamma^i\phi(h'_{b-1}) \end{bmatrix}$$

for $0 \leq i \leq n - 1$. Here $\phi : GF(2^r) \rightarrow GF(2^{r'})$ is an injective homomorphism of $GF(2^r)$ into $GF(2^{r'})$ under addition.

Theorem 7.18 can be proved in the same way as Theorem 7.17.

Decoding Decoding of the s-spotty byte error control code given by Theorem 7.18 is presented here. Let v, c, and E be the received word, the codeword, and the error vector, respectively. Then the syndrome S is calculated as follows:

$$\begin{aligned} S &= \begin{bmatrix} S_0 & S_1 & S_2 & \ldots & S_{\lambda+\mu-1} \end{bmatrix} \\ &= v \cdot \mathbf{H}^T = (c + E) \cdot \mathbf{H}^T = E \cdot \mathbf{H}^T, \end{aligned}$$

where $S_0 \in$GF(2^r) is an r-bit binary row vector and $S_i \in GF(2^{r'})$ is an r'-bit binary row vector, where $i = 1,\ 2,\ \ldots,\ \lambda + \mu - 1$. If $p(\leq \lambda)$ s-spotty byte errors, $E_0,\ E_1,\ \cdots,\ E_{p-1} \in E_{t/b}$, have occurred in the i_0-th, i_1-th, $\cdots$, i_{p-1}-th bytes, respectively, then the syndrome S is given as follows:

$$\begin{aligned} S &= \begin{bmatrix} S_0 & S_1 & S_2 & \ldots & S_{\lambda+\mu-1} \end{bmatrix} \\ &= \left[\sum_{x=0}^{p-1} E_x \cdot \mathbf{H}'^T \quad \sum_{x=0}^{p-1} E_x \cdot \gamma^{i_x}\mathbf{H}'^T \quad \sum_{x=0}^{p-1} E_x \cdot \gamma^{2i_x}\mathbf{H}'^T \quad \ldots \quad \sum_{x=0}^{p-1} E_x \cdot \gamma^{(\lambda+\mu-1)i_x}\mathbf{H}'^T \right]. \end{aligned} \tag{7.15}$$

From Lemma 7.4 this syndrome can be expressed as follows:

$$\left[\sum_{x=0}^{p-1} E_x \cdot \mathbf{H}'^T \quad \sum_{x=0}^{p-1} \gamma^{i_x} \phi\left(E_x \cdot \mathbf{H}'^T\right) \quad \sum_{x=0}^{p-1} \gamma^{2i_x} \phi\left(E_x \cdot \mathbf{H}'^T\right) \quad \cdots \quad \sum_{x=0}^{p-1} \gamma^{(\lambda+\mu-1)i_x} \phi\left(E_x \cdot \mathbf{H}'^T\right)\right]. \tag{7.16}$$

Here let $\widetilde{S}_0 = \phi(S_0) = \sum_{x=0}^{p-1} \phi(E_x \cdot \mathbf{H}'^T)$. Each $\widetilde{S}_i \in GF(2^{r'})$, $i = 0,\ 1,\ \ldots,\ \lambda + \mu - 1$, is an r'-bit binary row vector. Then $\widetilde{S}$ is written as follows:

$$\widetilde{S} = \left[\widetilde{S}_0 \quad \widetilde{S}_1 \quad \widetilde{S}_2 \quad \ldots \quad \widetilde{S}_{\lambda+\mu-1}\right]. \tag{7.17}$$

Now let $\phi(E_0 \cdot \mathbf{H}'^T),\ \phi(E_1 \cdot \mathbf{H}'^T),\ \cdots,\ \phi(E_{p-1} \cdot \mathbf{H}'^T) \in GF(2^{r'})$ be $\widetilde{E}_0,\ \widetilde{E}_1,\ \cdots,\ \widetilde{E}_{p-1} \in GF(2^{r'})$, respectively. Then $\widetilde{S}$ shown in Eq. (7.17) is expressed as follows:

$$\left[\sum_{x=0}^{p-1} \widetilde{E}_x \quad \sum_{x=0}^{p-1} \gamma^{i_x} \widetilde{E}_x \quad \sum_{x=0}^{p-1} \gamma^{2i_x} \widetilde{E}_x \quad \ldots \quad \sum_{x=0}^{p-1} \gamma^{(\lambda+\mu-1)i_x} \widetilde{E}_x\right]. \tag{7.18}$$

The $\widetilde{S}$ corresponds to the syndrome of the following λ bytes error correcting and μ bytes error detecting RS code over $GF(2^{r'})$ where the errors $\widetilde{E}_0,\ \widetilde{E}_1,\ \cdots,\ \widetilde{E}_{p-1} \in GF(2^{r'})$ have occurred in the i_0-th, i_1-th, $\cdots$, i_{p-1}-th byte, respectively:

$$\widetilde{\mathbf{H}} = \left[\widetilde{\mathbf{H}}_0 \quad \widetilde{\mathbf{H}}_1 \quad \widetilde{\mathbf{H}}_2 \quad \cdots \quad \widetilde{\mathbf{H}}_{n-1}\right] = \begin{bmatrix} \gamma^0 & \gamma^0 & \gamma^0 & \cdots & \gamma^0 \\ \gamma^0 & \gamma^1 & \gamma^2 & \cdots & \gamma^{n-1} \\ \gamma^0 & \gamma^2 & \gamma^4 & \cdots & \gamma^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \gamma^0 & \gamma^{\lambda+\mu-1} & \gamma^{2(\lambda+\mu-1)} & \cdots & \gamma^{(\lambda+\mu-1)(n-1)} \end{bmatrix},$$

where $\gamma \in GF(2^{r'})$ and $n = 2^{r'} - 1$. Therefore from Eq. (7.18) we can obtain the error locations $i_0,\ i_1,\ \cdots,\ i_{p-1}$, and the error patterns $\widetilde{E}_0,\ \widetilde{E}_1,\ \cdots,\ \widetilde{E}_{p-1} \in GF(2^r)$ by using the existing decoding algorithms of the RS codes, such as the Berlekamp-Massey algorithm, and the Euclidean algorithm shown in Subsections 2.3.5 and 2.3.6.

Since $\min(2t,\ b)$ column vectors of $\mathbf{H}'$ are linearly independent, that is, $\mathbf{H}'$ is a parity-check matrix of the t-bit error correcting codes, the error patterns $E_0,\ E_1,\ \cdots,\ E_{p-1} \in GF(2^b)$ can be obtained from the relations $E_0 \cdot \mathbf{H}'^T = \widetilde{E}_0,\ E_1 \cdot \mathbf{H}'^T = \widetilde{E}_1,\ \cdots,\ E_{p-1} \cdot \mathbf{H}'^T = \widetilde{E}_{p-1}$, respectively, by using a lookup table.

It should be clear that the lengthened code given by Theorem 7.18 can be decoded in the same way.

2. s-Spotty Byte Error Control Codes with Byte Error Detection Capability

Preliminaries Using Theorems 7.15 and 7.16, we need to calculate a number of extra syndromes to detect single-byte errors, as shown below:

$$\sum_{j=t+1}^{b} \binom{b}{j} = (2^b - 1) - \sum_{j=1}^{t} \binom{b}{j}. \tag{7.19}$$

From this relation, we have the following theorem:

Theorem 7.19 *If a code length N is a multiple of a byte length b, a linear $(N,\ N-R)$ λ t/b-errors correcting code with a single-byte error detection capability exists only if*

$$2^R - 1 \geq \sum_{i=1}^{\lambda} \left\{ \binom{N/b}{i} \cdot \left\{ \sum_{j=1}^{t} \binom{b}{j} \right\}^i \right\} + (2^b - 1) - \sum_{j=1}^{t} \binom{b}{j}. \tag{7.20}$$

Code Design

Theorem 7.20 *The null space of*

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I}_b & \boldsymbol{I}_b & \boldsymbol{I}_b & \cdots & \boldsymbol{I}_b \\ \gamma^0\boldsymbol{H}' & \gamma^1\boldsymbol{H}' & \gamma^2\boldsymbol{H}' & \cdots & \gamma^{n-1}\boldsymbol{H}' \\ \gamma^0\boldsymbol{H}' & \gamma^2\boldsymbol{H}' & \gamma^4\boldsymbol{H}' & \cdots & \gamma^{2(n-1)}\boldsymbol{H}' \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \gamma^0\boldsymbol{H}' & \gamma^{\lambda+\mu-1}\boldsymbol{H}' & \gamma^{2(\lambda+\mu-1)}\boldsymbol{H}' & \cdots & \gamma^{(\lambda+\mu-1)(n-1)}\boldsymbol{H}' \end{bmatrix}$$

is a λ t/b-errors correcting and μ t/b-errors detecting code with single-byte error detection capability, where $\boldsymbol{H}'$ is an $r \times b$ binary matrix, as defined in Definition 7.2, and $\boldsymbol{I}_b$ is a $b \times b$ identity matrix. This code has a check-bit length $R = b + r'(\lambda + \mu - 1)$ and a code length in bits $N = nb$, where $n = 2^{r'} - 1$.

The lengthened code is obtained by adding two columns in the same way as the code in Theorem 7.18.

Decoding The way to decode the s-spotty byte error control code given by Theorem 7.20 is presented here. The decoding method is similar to that of the previous codes.

First, the syndrome is calculated as $S = [S_0\ S_1\ S_2\ \cdots\ S_{\lambda+\mu-1}]$. If $S_0 \neq 0$ and $S_1 = \cdots = S_{\lambda+\mu-1} = 0$, the error cannot be corrected. In other cases we can obtain the error locations and the error patterns in $GF(2^{r'})$ from the syndrome shown in Eq. (7.17) by using the existing decoding algorithm of the RS codes. If the received word has only one erroneous byte, S_0 represents the error pattern of the erroneous byte. Therefore, if $w_{\mathrm{H}}(S_0) > t$, the error can be detected. Otherwise, the error patterns in $GF(2^b)$ must be determined from the corresponding error patterns in $GF(2^{r'})$.

Examples and Evaluation We proceed here to look at some example codes of Theorems 7.17 and 7.20, and evaluate them from the standpoint of check-bit length, error detection capability, and decoder hardware complexity.

Figure 7.15 gives an example code with parameters $\lambda = \mu = 2$, $b = 8$ bits, $t = 3$ bits, and information-bit length $K = 128$ for a parity-check matrix of the shortened double s-spotty byte

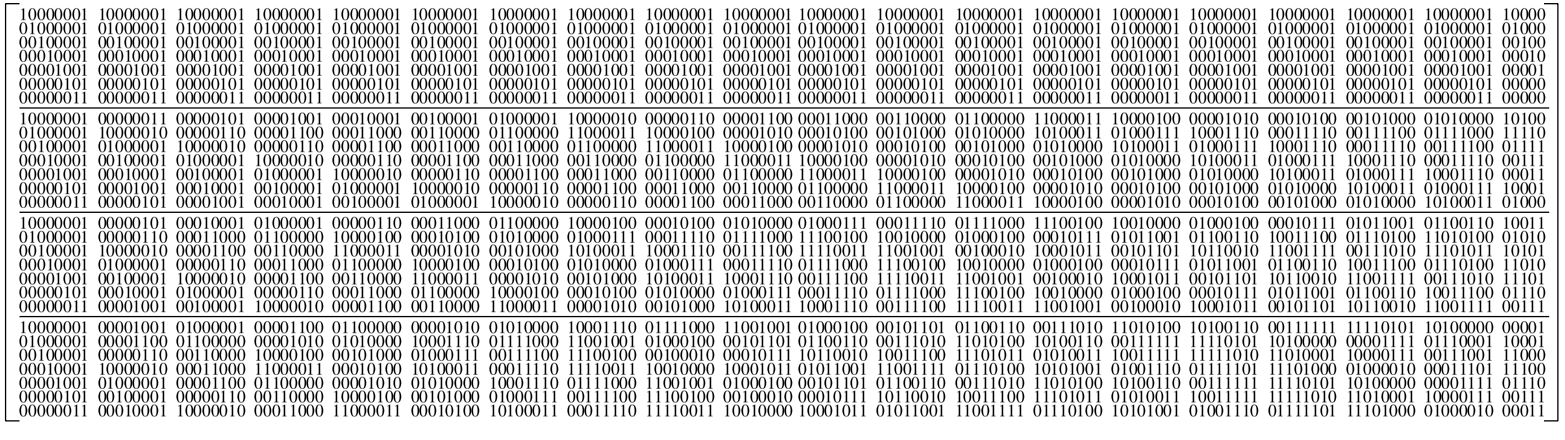


Figure 7.15 *(156, 128) s-spotty $D_{3/8}$EC code.*

TABLE 7.4 Error Detection Capabilities of $D_{3/8}EC$ Codes and $D_{3/8}EC$-S8ED Codes for Three Types of Multiple Errors

	$D_{3/8}EC$ codes (%)			$D_{3/8}EC$-S8ED codes (%)		
Errors	$K = 64$ $(R = 28)$	$K = 128$ $(R = 28)$	$K = 256$ $(R = 28)$	$K = 64$ $(R = 29)$	$K = 128$ $(R = 29)$	$K = 256$ $(R = 29)$
Random triple-bit errors	99.93	99.72	98.81	99.94	99.80	99.13
Single 8-bit byte errors	63.72	63.64	63.59	100	100	100
Single byte plus single bit errors	63.13	63.05	63.00	100	100	100

error correcting code, that is, the (156, 128) s-spotty $D_{3/8}EC$ code. In this case, γ is a primitive element in $GF(2^7)$ defined by the primitive polynomial $\mathbf{g}(x) = x^7 + x + 1$, and $\mathbf{H}'$ is the following 7×8 binary matrix whose any 6 or fewer column vectors are linearly independent:

$$\mathbf{H}' = \begin{bmatrix} 1 & \gamma & \gamma^2 & \gamma^3 & \gamma^4 & \gamma^5 & \gamma^6 & \gamma^{121} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

The error detection capabilities of the $D_{3/8}EC$ code and the $D_{3/8}EC$-S8ED code are presented in Table 7.4 for three types of errors that are beyond the error correction or detection capability of the code. Here, "single 8-bit byte plus single bit errors" means the errors caused by single 8-bit byte errors in a byte and single bit errors that occurred in another byte simultaneously.

Figure 7.16 shows the relations between the information-bit lengths and the check-bit lengths of the lengthened $D_{3/8}EC$ codes and $D_{3/8}EC$-S8ED codes, along with the double 8-bit byte error correcting codes, which are the D8EC codes.

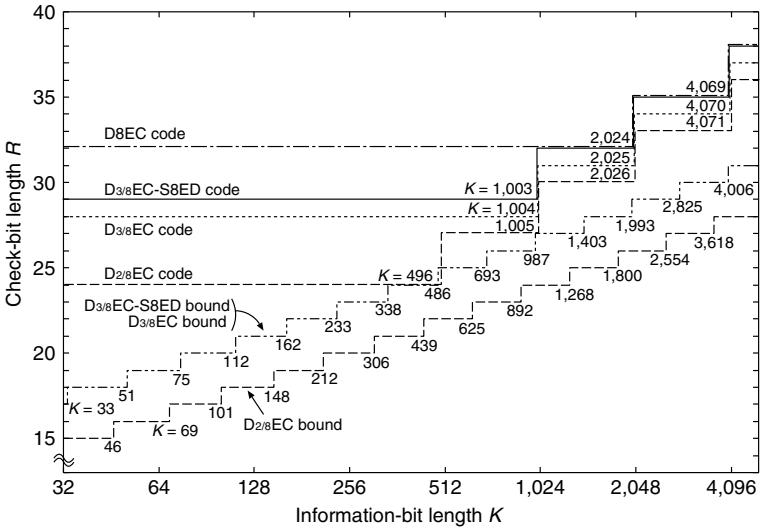


Figure 7.16 *Comparison of information-bit lengths and check-bit lengths of the s-spotty $D_{2/8}EC$ codes, the $D_{3/8}EC$ codes, and the $D_{3/8}EC$-S8ED codes.*

7.5.2 A General Class of Codes for m-Spotty Byte Errors

When high-energy particles strike a particular RAM chip, more than t-bit errors may result in the corresponding byte. Hence we would have to deal with another type of spotty byte error, that is, multiple spotty byte error—called *m-spotty byte error* for short. The m-spotty byte error control codes can correct or detect multiple spotty byte errors that are distributed in multiple bytes or have become concentrated in a few bytes. This code can flexibly correct or detect any size of errors in a byte. The m-spotty byte error control codes are applicable to the devices at one time in the circumstances that change every time. For example, very strong particles can strike the particular device at one time in a cosmic space and so cause multiple errors in a byte, or at another time a shower of particles may strike a number of such devices and cause multiple spotty byte errors distributed over multiple bytes.

Figure 7.17 gives examples of erroneous words with three spotty byte errors, that is, each with triple t/b-errors, where $t = 2$ and $b = 8$. In Figure 7.17(a) note that the word has three erroneous bytes, each of which has a single t/b-error. In this case, the maximum number of erroneous bits in a byte does not exceed $t = 2$ bits, so these are triple s-spotty byte errors. Figure 7.17(b) shows a second byte with 3-bit errors. Because $\lceil 3/t \rceil = \lceil 3/2 \rceil = 2$, these errors are double t/b-error in a byte. Here, $\lceil \delta \rceil$ means the smallest integer larger than or equal to δ. Because there is another erroneous byte that includes a single t/b-error, the total number of t/b-errors is three, which illustrates the case of a triple t/b-error. In Figure 7.17(c) we see a single erroneous byte that includes five bits errors. Because $\lceil 5/t \rceil = \lceil 5/2 \rceil = 3$, the word also includes triple t/b-error. The triple m-spotty byte error control codes with $t = 2$ and $b = 8$ can detect or correct all these erroneous bytes.

This subsection is focused on a generalized m-spotty byte error control codes with a *minimum m-spotty distance d*. The practical design of the m-spotty byte error control codes with $d = 3$ and 4 have been shown in Sections 7.2 through 7.4. Here we have a general class of m-spotty byte error control codes with any value d. We also present a *complex m-spotty byte error control codes* that can control two kinds of spotty errors with different sizes, t and t'. Furthermore, efficient and practical m-spotty byte error control codes are

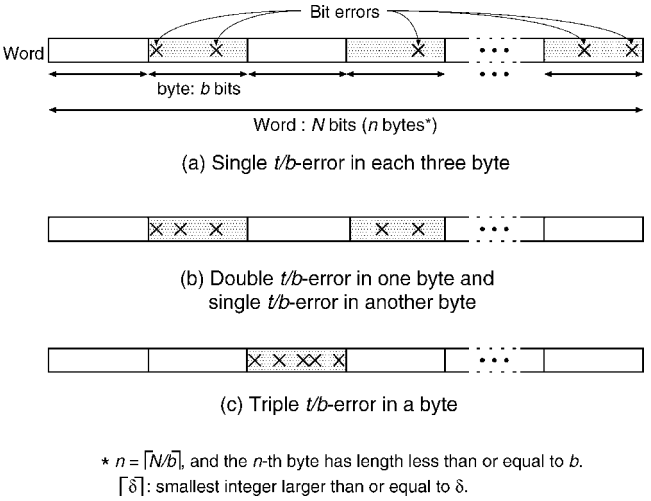


Figure 7.17 *Examples of triple spotty byte errors where $t = 2$ bits and $b = 8$ bits.* Source: *[SUZU04]. © 2004 IEEE.*

presented, where the codes are designed based on the fact that errors usually occur in a small number of RAM chips at most two or three chips simultaneously even in large capacity memory systems. This makes it possible to reduce the number of check bits of the codes.

1. m-Spotty Byte Error Control Codes

Preliminaries

Definition 7.3 Let E be an error vector with length n bytes. And let E_i be the i-th byte of E, where $0 \le i \le n-1$. If the vector E satisfies the relation

$$\sum_{i=0}^{n-1} \left\lceil \frac{w_{\mathrm{H}}(E_i)}{t} \right\rceil = l,$$

then this type of errors is called l *m-spotty byte errors*, where $w_{\mathrm{H}}(E_i)$ is a Hamming weight of vector E_i. In particular, if the total number of t/b-errors l is equal to the number of erroneous bytes, then this type of errors is called l *s-spotty byte errors*. □

Definition 7.4 For codewords x and y with n byte, the *m-spotty distance function* $d_{\mathrm{M}}(x, y)$ of the code $\mathbf{C}$ is defined as follows:

$$d_{\mathrm{M}}(x, y) = \sum_{i=0}^{n-1} \left\lceil \frac{d_{\mathrm{H}}(x_i,\ y_i)}{t} \right\rceil, \tag{7.21}$$

where $d_{\mathrm{H}}(x_i, y_i)$ denotes the Hamming distance between the i-th bytes of x and y. Also *minimum m-spotty distance* d is defined as $d = \min_{\substack{x,\ y \in C \\ x \neq y}} d_{\mathrm{M}}(x, y)$. □

Theorem 7.21 *Let $\boldsymbol{H}_i$ be an $R \times b$ binary submatrix for $0 \le i \le n-1$, and also let $\boldsymbol{E}_{t/b} = \{E \in GF(2^b) \mid 1 \le w_H(E) \le t\}$ be a set of t/b-errors in a byte. The null space of $\boldsymbol{H} = [\boldsymbol{H}_0\ \boldsymbol{H}_1\ \boldsymbol{H}_2\ \boldsymbol{H}_3\ \cdots\ \boldsymbol{H}_{n-1}]$ is an m-spotty byte error control code with minimum m-spotty distance d if and only if the following relation is satisfied:*

$$\begin{aligned}
&E_1 \cdot \boldsymbol{H}_{i_1}^T + \ldots + E_{v_1} \cdot \boldsymbol{H}_{i_{v_1}}^T \\
&+ \underbrace{(E_{v_1+1} + E_{v_1+2})}_{2} \cdot \boldsymbol{H}_{i_{v_1+1}}^T + \ldots + \underbrace{(E_{v_1+2v_2-1} + E_{v_1+2v_2})}_{2} \cdot \boldsymbol{H}_{i_{v_1+v_2}}^T \\
&+ \ldots \\
&+ \underbrace{(E_{v_1+2v_2+\ldots+(c-1)v_{c-1}+1} + \ldots + E_{v_1+2v_2+\ldots+(c-1)v_{c-1}+c})}_{c} \cdot \boldsymbol{H}_{i_{v_1+v_2+\ldots+v_{c-1}+1}}^T \\
&+ \ldots + \underbrace{(E_{v_1+2v_2+\ldots+cv_c-(c-1)} + \ldots + E_{v_1+2v_2+\ldots+cv_c})}_{c} \cdot \boldsymbol{H}_{i_{v_1+v_2+\ldots+v_c}}^T \neq 0 \\
&\qquad \textit{for} \quad 0 < v_1 + 2v_2 + \ldots + cv_c \le d-1,\ 0 \le v_1 \le d-1, \\
&\qquad\quad 0 \le v_2 \le \lfloor (d-1)/2 \rfloor,\ \cdots,\ 0 \le v_c \le \lfloor (d-1)/c \rfloor,
\end{aligned} \tag{7.22}$$

where $\forall E_j \in \boldsymbol{E}_{t/b}\ (j = 1, 2, \cdots,\ v_1 + 2v_2 + \cdots + cv_c)$, $c = \lceil b/t \rceil$ *and* $i_1,\ i_2,\ \cdots,\ i_{v_1+v_2+\ldots+v_c}$ *are distinct integers of i satisfying* $0 \le i_1,\ i_2,\ \cdots,\ i_{v_1+v_2+\ldots+v_c} \le n-1$.

Theorem 7.22 *A linear m-spotty byte error control code with minimum m-spotty distance d requires at least* $(d-1)t$ *check bits.*

Theorem 7.23 *If N is a multiple of b, a linear* $(N,\ N-R)$ *m-spotty byte error control code with minimum m-spotty distance d exists only if*

$$2^R \geq 1 + \sum_{j=1}^{\lfloor (d-1)/2 \rfloor} S_j(n) + ((d-1) \bmod 2) \cdot \Big\{ (2^t - 1) \cdot S_{\lfloor (d-1)/2 \rfloor}(n-1) + \sum_{v=1}^{M} \left\{ \sum_{u=0}^{t-1} \left\{ \binom{t}{t-u} \cdot \sum_{i=u+(v-1)\cdot t+1}^{\min(v\cdot t,\ b-t)} \binom{b-t}{i} \right\} \times S_{\lfloor (d-1)/2 \rfloor - v}(n-1) \right\} \Big\}, \quad (7.23)$$

where $\lfloor \delta \rfloor$ *is the largest integer smaller than or equal to* δ, $M = min(c-1, \lfloor (d-1)/2 \rfloor)$,

$$S_m(n) = \sum_{\substack{p_1, p_2, \cdots, p_c \geq 0 \\ p_1 + 2p_2 + \cdots + cp_c = m}} \left\{ \binom{n}{p_1 + p_2 + \cdots + p_c} \times \binom{p_1 + p_2 + \cdots + p_c}{p_1,\ p_2,\ \ldots,\ p_c} \times \prod_{z=1}^{c} \left\{ \sum_{i=(z-1)\cdot t+1}^{\min(z\cdot t,\ b)} \binom{b}{i} \right\}^{p_z} \right\} \quad (m \geq 1),$$

$S_0(n) = 1$, *and*

$$\binom{p_1 + p_2 + \cdots + p_c}{p_1,\ p_2,\ \ldots,\ p_c} = \frac{(p_1 + p_2 + \cdots + p_c)!}{p_1! \times p_2! \times \cdots \times p_c!}.$$

Code Design

Definition 7.5 Let $\mathbf{H}' = [h'_0\ h'_1\ \cdots\ h'_{b-1}]$ be a $q \times b$ binary matrix whose any $\min((d-1)t, b)$ or fewer column vectors are linearly independent, where $h'_0,\ h'_1,\ \cdots,\ h'_{b-1}$ are binary column vectors of $GF(2^q)$. Also let $\mathbf{H}'' = [h''_0\ h''_1\ \cdots\ h''_{b-1}]$ be an $r \times b$ binary matrix whose any $\min(\lfloor (d-1)/2 \rfloor t, b)$ or fewer column vectors are linearly independent, where $h''_0,\ h''_1,\ \cdots,\ h''_{b-1}$ are binary column vectors of $GF(2^r)$. □

If $\min((d-1)t, b) = b$, the matrix $\mathbf{H}'$ can be any nonsingular $b \times b$ matrix, including $b \times b$ identity matrix. On the other hand, if $\min((d-1)t, b) = (d-1)t < b$, the matrix $\mathbf{H}'$ is a parity-check matrix of a linear binary $(b,\ b-q)$ code with minimum Hamming distance $(d-1)t+1$. And if $\min(\lfloor (d-1)/2 \rfloor t, b) = b$, the matrix $\mathbf{H}''$ can be any nonsingular $b \times b$ matrix, including $b \times b$ identity matrix. Nevertheless, if $\min(\lfloor (d-1)/2 \rfloor t, b) = \lfloor (d-1)/2 \rfloor t < b$, the matrix $\mathbf{H}''$ is a parity-check matrix of a linear binary $(b,\ b-r)$ code with minimum Hamming distance $\lfloor (d-1)/2 \rfloor t + 1$.

We use the above-defined $\mathbf{H}'$ and $\mathbf{H}''$ matrices in the following theorems to design the distance-d m-spotty byte error control codes.

Theorem 7.24 *Let γ be a primitive element of $GF(2^r)$. The null space of*

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{H}' & \boldsymbol{H}' & \cdots & \boldsymbol{H}' \\ \gamma^0\boldsymbol{H}'' & \gamma^1\boldsymbol{H}'' & \cdots & \gamma^{n-1}\boldsymbol{H}'' \\ \gamma^0\boldsymbol{H}'' & \gamma^2\boldsymbol{H}'' & \cdots & \gamma^{2(n-1)}\boldsymbol{H}'' \\ \vdots & \vdots & \ddots & \vdots \\ \gamma^0\boldsymbol{H}'' & \gamma^{(d-2)}\boldsymbol{H}'' & \cdots & \gamma^{(d-2)(n-1)}\boldsymbol{H}'' \end{bmatrix}$$

is a distance-d m-spotty byte error control code with check-bit length $R = q + (d-2)r$ and code length in bits $N = n \cdot b$, where $n = 2^r - 1$, and $\gamma^i\boldsymbol{H}'' = [\gamma^i h_0'' \ \gamma^i h_1'' \ \cdots \ \gamma^i h_{b-1}'']$ for $0 \le i \le n-1$.

Note that for $b/\lfloor (d-1)/2 \rfloor \le t \le b$, the code shown in Theorem 7.24 is identical to the maximum distance separable (MDS) RS code over $GF(2^b)$ because $\mathbf{H}'$ and $\mathbf{H}''$ are equal to $b \times b$ identity matrices.

Next, we consider the lengthened code which is further lengthened by taking r' being larger than the original r.

Theorem 7.25 *Let γ be a primitive element of $GF(2^{r'})$, where $r' = (R-q)/(d-2)$, $r' \ge r$, and $R - q$ is a multiple of $d - 2$. The null space of*

$$\boldsymbol{H} = \left[\begin{array}{cccc|cc} \boldsymbol{H}' & \boldsymbol{H}' & \cdots & \boldsymbol{H}' & \boldsymbol{H}' & \boldsymbol{O} \\ \gamma^0\boldsymbol{H}'' & \gamma^1\boldsymbol{H}'' & \cdots & \gamma^{n-1}\boldsymbol{H}'' & \boldsymbol{O} & \boldsymbol{O} \\ \gamma^0\boldsymbol{H}'' & \gamma^2\boldsymbol{H}'' & \cdots & \gamma^{2(n-1)}\boldsymbol{H}'' & \boldsymbol{O} & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \gamma^0\boldsymbol{H}'' & \gamma^{(d-2)}\boldsymbol{H}'' & \cdots & \gamma^{(d-2)(n-1)}\boldsymbol{H}'' & \boldsymbol{O} & \boldsymbol{I}_{r'} \end{array}\right]$$

is a distance-d m-spotty byte error control code with check-bit length R and code length in bits $N = (n+1) \cdot b + r'$, where $n = 2^{r'} - 1$, $\gamma^i\boldsymbol{H}'' = [\gamma^i\phi(h_0'') \ \gamma^i\phi(h_1'') \ \cdots \ \gamma^i\phi(h_{b-1}'')]$ for $0 \le i \le n-1$, ϕ: $GF(2^r) \to GF(2^{r'})$ is a homomorphism of $GF(2^r)$ into $GF(2^{r'})$ under addition, and $I_{r'}$ is $r' \times r'$ identity matrix.

Example 7.3

Figure 7.18 gives an example of a parity-check matrix of the shortened double m-spotty byte error correcting code, which is a (285, 256) m-spotty $D_{3/8}EC$ code with parameters of $d = 5$, $b = 8$ bits, and $t = 3$ bits, and information-bit length $K = 256$. Original is a lengthened (1031,1002) code. Here γ is a primitive element in $GF(2^7)$ defined by the primitive polynomial $\mathbf{g}(x) = x^7 + x + 1$, $\mathbf{H}'$ is an 8×8 identity matrix, and $\mathbf{H}''$ is the following 7×8 binary matrix whose any 6 or fewer column vectors are linearly independent:

$$\mathbf{H}'' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} \gamma^0 & \gamma^1 & \gamma^2 & \gamma^3 & \gamma^4 & \gamma^5 & \gamma^6 & \gamma^{121} \end{bmatrix}.$$

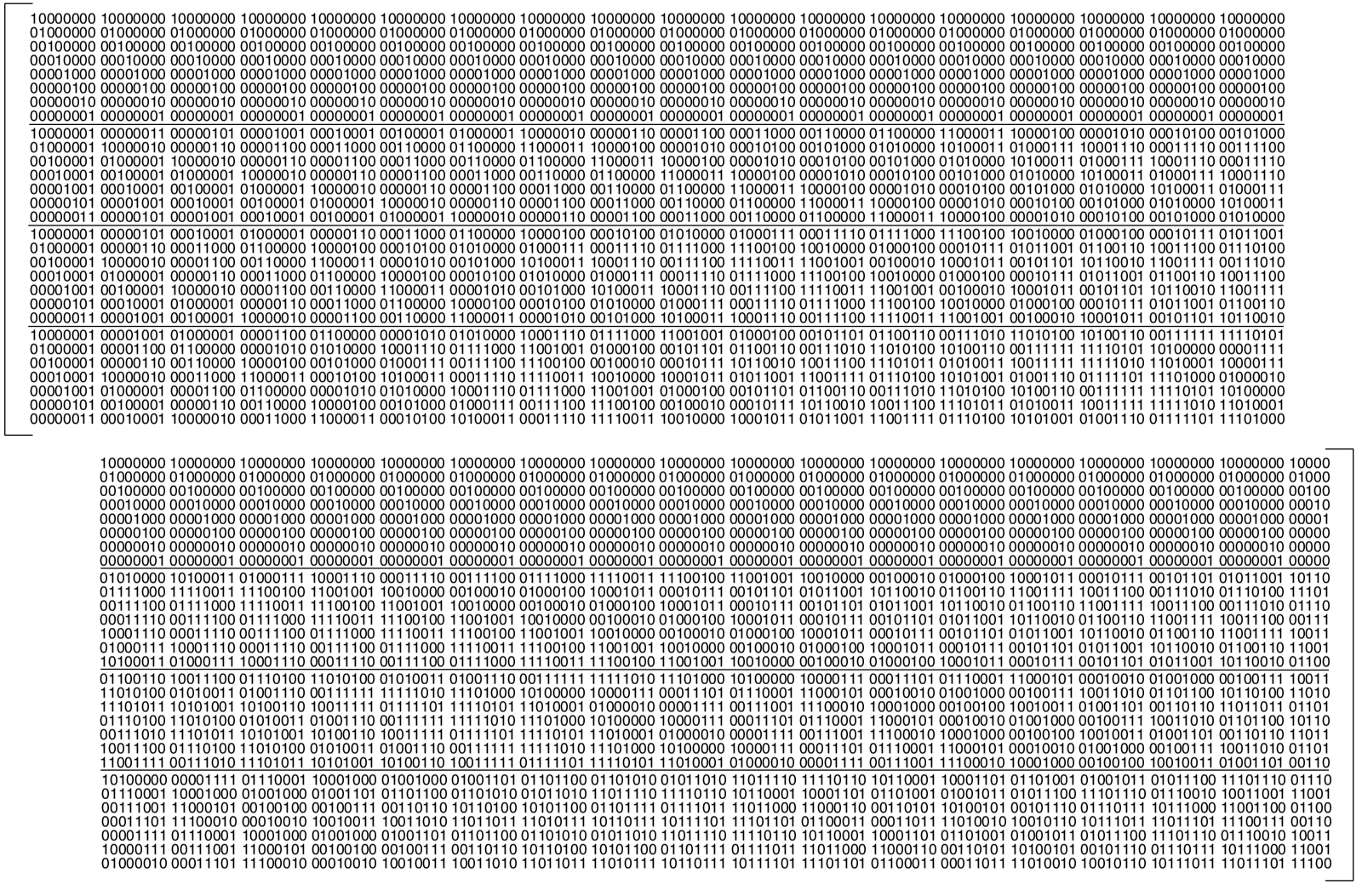


Figure 7.18 *Shortened (285, 256) m-spotty $D_{3/8}EC$ code.*

Decoding The m-spotty byte error control code given by Theorem 7.24 is decoded here. Let v, c, and E be the received word, codeword, and error vector, respectively. Then the syndrome S is calculated as follows:

$$\begin{aligned} S &= [S_0\ S_1\ S_2\ \ldots\ S_{d-2}] \\ &= v \cdot \mathbf{H}^T = (c + E) \cdot \mathbf{H}^T = E \cdot \mathbf{H}^T, \end{aligned}$$

where $S_0 \in GF(2^q)$ is an q-bit binary row vector and $S_i \in GF(2^r)$ is an r-bit binary row vector for $i = 1,\ 2,\ \ldots,\ d-2$. If $p(\leq \lceil (d-1)/2 \rceil)$ m-spotty byte errors, $E_0,\ E_1,\ \ldots,\ E_{p-1} \in \boldsymbol{E}_{t/b}$ have occurred in the i_0-th, i_1-th, ..., i_{p-1}-th byte, respectively, then the syndrome S is given by Eq. (7.24):

$$\begin{aligned} S &= \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ \vdots \\ S_{d-2} \end{bmatrix}^T \\ &= \begin{bmatrix} E_0 \cdot \mathbf{H}'^T + E_1 \cdot \mathbf{H}'^T + \cdots + E_{p-1} \cdot \mathbf{H}'^T \\ \gamma^{i_0} E_0 \cdot \mathbf{H}''^T + \gamma^{i_1} E_1 \cdot \mathbf{H}''^T + \cdots + \gamma^{i_{p-1}} E_{p-1} \cdot \mathbf{H}''^T \\ \gamma^{2i_0} E_0 \cdot \mathbf{H}''^T + \gamma^{2i_1} E_1 \cdot \mathbf{H}''^T + \cdots + \gamma^{2i_{p-1}} E_{p-1} \cdot \mathbf{H}''^T \\ \vdots \\ \gamma^{(d-2)i_0} E_0 \cdot \mathbf{H}''^T + \gamma^{(d-2)i_1} E_1 \cdot \mathbf{H}''^T + \cdots + \gamma^{(d-2)i_{p-1}} E_{p-1} \cdot \mathbf{H}''^T \end{bmatrix}^T \end{aligned} \tag{7.24}$$

We let $\sum_{x=0}^{p-1} E_x$ be expressed as E^*. The relation $S_0 = E^* \cdot \mathbf{H}'^T$ leads to E^* because $\mathbf{H}'$ is a parity-check matrix of $\lfloor (d-1)/2 \rfloor$ t/b-error correcting and $\lceil (d-1)/2 \rceil$ t/b-error detecting code. Multiplying this by $\mathbf{H}''^T$ from the right gives $E^* \cdot \mathbf{H}''^T$. We also let $E_0 \cdot \mathbf{H}''^T, E_1 \cdot \mathbf{H}''^T, \ldots, E_{p-1} \cdot \mathbf{H}''^T \in GF(2^r)$ be $\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_{p-1}$, respectively, and then the syndrome S' is given by Eq. (7.25):

$$S' = \begin{bmatrix} S'_0 \\ S_1 \\ S_2 \\ \vdots \\ S_{d-2} \end{bmatrix}^T = \begin{bmatrix} \varepsilon_0 + \varepsilon_1 + \cdots + \varepsilon_{p-1} \\ \gamma^{i_0}\varepsilon_0 + \gamma^{i_1}\varepsilon_1 + \cdots + \gamma^{i_{p-1}}\varepsilon_{p-1} \\ \gamma^{2i_0}\varepsilon_0 + \gamma^{2i_1}\varepsilon_1 + \cdots + \gamma^{2i_{p-1}}\varepsilon_{p-1} \\ \vdots \\ \gamma^{(d-2)i_0}\varepsilon_0 + \gamma^{(d-2)i_1}\varepsilon_1 + \cdots + \gamma^{(d-2)i_{p-1}}\varepsilon_{p-1} \end{bmatrix}^T \tag{7.25}$$

This syndrome is identical to that of the RS code with a minimum Hamming distance d over $GF(2^r)$. The error patterns of $GF(2^r)$ and error locations are determined next by using a decoding algorithm of the RS code such as the Berlekamp-Massey algorithm mentioned in Subsections 2.3.5 and 2.3.6.

In the final step of the decoding, the error patterns $\widehat{E}_x \in \mathrm{GF}(2^b)$, where $x = 0,\ 1,\ \ldots,\ p-1$, are transformed from the corresponding error patterns $\varepsilon_x \in \mathrm{GF}(2^r)$ to $\widehat{E}_x$ by one-to-one mapping for $x = 0,\ 1,\ \ldots,\ p-1$. In this case at most one of $\widehat{E}_x$'s may

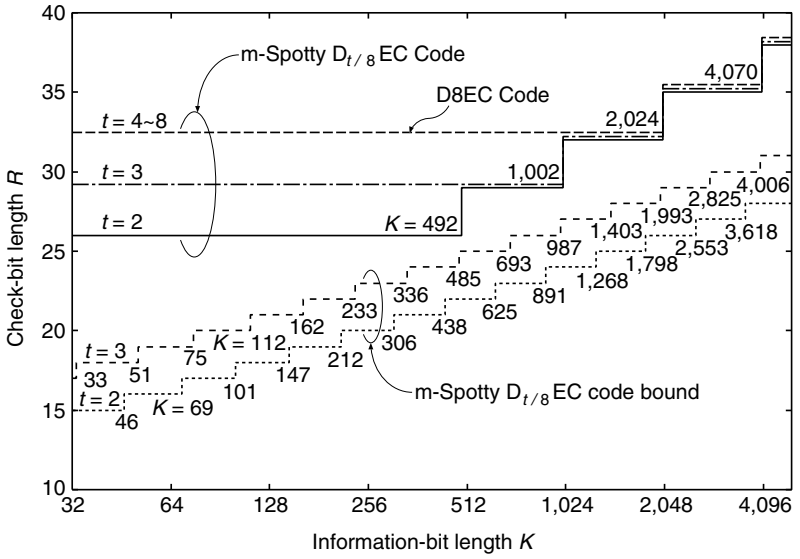


Figure 7.19 *Comparison of information-bit lengths and check-bit lengths of the m-spotty $D_{t/8}$EC codes.* Source: *[SUZU04] © 2004 IEEE.*

be miscorrected, that is, $\widehat{E_x} \neq E_x$. The following relation determines whether or not $\widehat{E_x}$ is identical to E_x. That is, if $\widehat{E_x}$ satisfies the relation (7.26), then $\widehat{E_x}$ is equal to E_x. Otherwise, $\widehat{E_x} \neq E_x$.

$$w_{\mathrm{M}}(\widehat{E_x} + E^*) \leq \lfloor (d-1)/2 \rfloor - w_{\mathrm{M}}(\widehat{E_x}). \tag{7.26}$$

This relation can be proved based on the fact that $w_{\mathrm{M}}(E)$ satisfies the triangle inequality [IMAI79].

Evaluation Figure 7.19 shows the relation between the information-bit length K and the check-bit length R for the lengthened m-spotty double byte error correcting codes with $d = 5$, $b = 8$, $t = 2$ and 3, and for the double byte error correcting codes, namely the D8EC codes. Here bound is the one given by Theorems 7.22 and 7.23. The error detection capabilities of the m-spotty $D_{2/8}$EC code are presented in Table 7.5 for four types of errors that are beyond the error correction capability of the code.

TABLE 7.5 Error Detection Capabilities of m-Spotty $D_{2/8}$EC Codes for Four Types of Errors

Errors	Error detection capability (%)		
	$K = 64$ ($R = 26$)	$K = 128$ ($R = 26$)	$K = 256$ ($R = 26$)
Random triple-bit errors	99.97	99.64	98.45
Single-byte errors	100	100	100
Single-byte plus single-bit errors	99.51	99.51	99.51
Double-byte errors	92.54	92.53	92.53

2. Complex m-Spotty Byte Error Control Codes

We now come to a new error model of two spotty errors with different lengths; that is we are interested in the case of t/b-error and t'/b-error, where $t' \neq t$, called *complex spotty byte errors*. When these two errors occur in one byte at a time, they are included in the m-spotty byte errors, meaning $t + t' \leq b$. Here we consider two types of *complex m-spotty byte error control codes*. One is the complex spotty byte error detecting code type, and the other is the complex spotty byte error correcting code type [SUZU05a].

(1) $S_{t/b}$EC-($S_{t/b} + S_{t'/b}$)ED Codes

We start our discussion with distance-4 type complex m-spotty byte error control codes. The following theorem shows the necessary and sufficient condition of the single t/b-error correcting and single t/b-error plus single t'/b-error detecting codes, called m-spotty $S_{t/b}$EC-($S_{t/b}$+ $S_{t'/b}$)ED codes. This class of codes can detect both single t/b-error and single t'/b-error in a byte that have occurred simultaneously for $t + t' \leq b$ and it can detect these errors simultaneously in the different two bytes.

Code Conditions and Bounds

Theorem 7.26 *Let $\boldsymbol{H}_i$ be an $R \times b$ binary submatrix for $0 \leq i \leq n-1$, and let $\boldsymbol{E}_{t/b} = \{E \in GF(2^b) \mid 1 \leq w_H(E) \leq t\}$, $\boldsymbol{E}_{t'/b} = \{E' \in GF(2^b) \mid 1 \leq w_H(E') \leq t'\}$, and $t \neq t'$. Here $w_H(E)$ means the Hamming weight over $GF(2)$ of E. The null space of $\boldsymbol{H} = [\boldsymbol{H}_0\ \boldsymbol{H}_1\ \boldsymbol{H}_2\ \boldsymbol{H}_3\ \cdots\ \boldsymbol{H}_{n-1}]$ is an $S_{t/b}$EC-($S_{t/b}$+$S_{t'/b}$)ED code if and only if*

1. $(E_1 + E_2 + E'_3) \cdot \boldsymbol{H}_i^T \neq 0$ *for* $E_1 + E_2 + E'_3 \neq 0$,
2a. $(E_1 + E_2) \cdot \boldsymbol{H}_i^T + E'_3 \cdot \boldsymbol{H}_j^T \neq 0$ *for* $E_1 + E_2 \neq 0$,
2b. $(E_1 + E'_3) \cdot \boldsymbol{H}_i^T + E_2 \cdot \boldsymbol{H}_j^T \neq 0$ *for* $E_1 + E'_3 \neq 0$,
3. $E_1 \cdot \boldsymbol{H}_i^T + E_2 \cdot \boldsymbol{H}_j^T + E'_3 \cdot \boldsymbol{H}_k^T \neq 0$,

where $\forall E_1,\ E_2 \in \boldsymbol{E}_{t/b}$, $\forall E'_3 \in \boldsymbol{E}_{t'/b}$, and $i,\ j,\ k$ are mutually distinct integers, satisfying $0 \leq i,\ j,\ k \leq n-1$.

Proof Condition 1 of this theorem ensures that single t/b-errors generate a nonzero syndrome. Condition 1 also says that a syndrome generated by a single t/b-error is different from that generated by other single t/b-errors and single t'/b-errors occurred in the same byte. Condition 2a includes the condition $E_1 \cdot \mathbf{H}_i^T + E'_3 \cdot \mathbf{H}_j^T \neq 0$, where the syndrome caused by a single t/b-error plus single t'/b-error should be a nonzero. Condition 2a, together with conditions 2b and 3, says that the syndrome caused by a single t/b-error should be different from that caused by a single t/b-error plus single t'/b-error. So these double spotty byte errors of a single t/b-error plus single t'/b-error are detectable. Condition 2b includes the condition $E_1 \cdot \mathbf{H}_i^T + E_2 \cdot \mathbf{H}_j^T \neq 0$, which ensures that the syndromes caused by different single t/b-errors are distinguishable. Therefore the code that satisfies the conditions above is capable of correcting single t/b-errors and detecting a single t/b-error plus single t'/b-error.

Theorem 7.27 *A linear $S_{t/b}$EC-($S_{t/b}$+$S_{t'/b}$)ED code requires at least $2t + t'$ check bits.*

Proof According to the conditions of the previous theorem, the $2t + t'$ binary columns of the parity-check matrix of this code are linearly independent. Therefore this code requires at least $2t + t'$ check bits. Q.E.D.

Theorem 7.28 *If N is a multiple of b, a linear $(N,\ N-R)$ $S_{t/b}EC$-$(S_{t/b}+S_{t'/b})ED$ code exists only if*

$$2^R - 1 \geq \frac{N}{b} \cdot \sum_{i=1}^{t} \binom{b}{i} + (2^{t'} - 1) \cdot \left(\frac{N}{b} - 1\right) \cdot \sum_{i=1}^{t} \binom{b}{i} + (2^{t'} - 1) \cdot \binom{b - t'}{t}. \tag{2.27}$$

Proof The total number of t/b-errors that can corrupt a single b-bit byte is given by $\sum_{i=1}^{t} \binom{b}{i}$. There are N/b bytes in a codeword with length N bits. Therefore we need $N/b \times \sum_{i=1}^{t} \binom{b}{i}$ different syndrome patterns to correct all single t/b-errors. Next, we consider the number of extra syndromes for error detection of single t/b-errors plus single t'/b-errors. The syndromes of a single t'/b-error plus single t/b-error that corrupts both a certain t'-bit in the fixed byte and a t-bit in another byte are all different and not equal to zero. The syndromes of this type of errors that corrupts both the t'-bit in the fixed byte and another t-bit in the same byte are also different and not equal to zero. These syndromes are further different from those of single t/b-errors. Therefore the number of extra syndromes necessary for detecting distinct single t/b-errors plus single t'/b-errors is calculated as

$$(2^{t'} - 1) \cdot \left(\frac{N}{b} - 1\right) \cdot \sum_{i=1}^{t} \binom{b}{i} + (2^{t'} - 1) \cdot \binom{b - t'}{t}.$$

Q.E.D.

Code Design

Definition 7.6 Let $\mathbf{H}' = \left[h'_0\ h'_1\ \cdots\ h'_{b-1}\right]$ be a $q \times b$ matrix whose $\min(2t + t',\ b)$ column vectors are linearly independent, where $h'_0,\ h'_1,\ \ldots,\ h'_{b-1}$ are binary column vectors of $GF(2^q)$. Also let $\mathbf{H}'' = \left[h''_0\ h''_1\ \cdots\ h''_{b-1}\right]$ be an $r \times b$ binary matrix whose $\max(t,\ t')$ column vectors are linearly independent, where $h''_0, h''_1, \ldots, h''_{b-1}$ are binary column vectors of $GF(2^r)$. □

The matrix $\mathbf{H}'$ is a $b \times b$ nonsingular matrix that includes a $b \times b$ identity matrix for $\min(2t + t',\ b) = b$. On the other hand, for $\min(2t + t',\ b) = 2t + t' < b$, the matrix $\mathbf{H}'$ is a parity-check matrix of a linear binary $(b,\ b - q)$ code with minimum Hamming distance $2t + t' + 1$. For $\max(t,\ t') = t$, the matrix $\mathbf{H}''$ is a parity-check matrix of a linear binary $(b,\ b - r)$ code with minimum Hamming distance $t + 1$. For $\max(t,\ t') = t'$, the matrix $\mathbf{H}''$ is a parity-check matrix of a linear binary $(b,\ b - r)$ code with minimum Hamming distance $t' + 1$.

From the above-defined $\mathbf{H}'$ and $\mathbf{H}''$ we can design the $S_{t/b}EC$-$(S_{t/b}+S_{t'/b})ED$ code in the following theorem.

Theorem 7.29 *Let γ be a primitive element of $GF(2^{r'})$, where $r \leq r'$. The null space of*

$$\boldsymbol{H} = \left[\begin{array}{ccccc|ccc} \mathbf{H}' & \boldsymbol{H}' & \boldsymbol{H}' & \cdots & \boldsymbol{H}' & \boldsymbol{H}' & \boldsymbol{O}_{q\times r'} & \boldsymbol{O}_{q\times r'} \\ \gamma^0\boldsymbol{H}'' & \gamma^1\boldsymbol{H}'' & \gamma^2\boldsymbol{H}'' & \cdots & \gamma^{(n-1)}\boldsymbol{H}'' & \boldsymbol{O}_{r'\times b} & \boldsymbol{I}_{r'} & \boldsymbol{O}_{r'\times r'} \\ \gamma^0\boldsymbol{H}'' & \gamma^2\boldsymbol{H}'' & \gamma^4\boldsymbol{H}'' & \cdots & \gamma^{2(n-1)}\boldsymbol{H}'' & \boldsymbol{O}_{r'\times b} & \boldsymbol{O}_{r'\times r'} & \boldsymbol{I}_{r'} \end{array}\right]$$

is an $S_{t/b}$EC-($S_{t/b}$+$S_{t'/b}$)ED code with check-bit length $R = q + 2r'$ and code length in bits $N = (n+1)\cdot b + 2r'$, where $n = 2^{r'} - 1$, $\boldsymbol{O}_{q\times r'}$ is a $q \times r'$ zero matrix, $\boldsymbol{O}_{r'\times b}$ is an $r' \times b$ zero matrix, $\boldsymbol{O}_{r'\times r'}$ is an $r' \times r'$ zero matrix, $\boldsymbol{I}_{r'}$ is an $r' \times r'$ identity matrix, $\gamma^i \boldsymbol{H}'' = \left[\gamma^i\phi(h''_0) \quad \gamma^i\phi(h''_1) \quad \cdots \quad \gamma^i\phi(h''_{b-1})\right]$ for $0 \le i \le n-1$, and $\phi : GF(2^r) \to GF(2^{r'})$ is an injective homomorphism of $GF(2^r)$ into $GF(2^{r'})$ under addition.

Proof The following shows how the code indicated in this theorem satisfies the conditions in Theorem 7.26.

Condition 1: Since $\mathbf{H}'$ is a $q \times b$ binary matrix whose $\min(2t + t', b)$ column vectors are linearly independent, $(E_1 + E_2 + E'_3)\cdot \mathbf{H}'^T \neq 0$ for $\forall E_1,\ E_2 \in \boldsymbol{E}_{t/b}$, $\forall E'_3 \in \boldsymbol{E}_{t'/b}$, and $E_1 + E_2 + E'_3 \neq 0$.

Condition 2a: Without loss of generality, we assume that the following equation holds for $(E_1 + E_2) \neq 0$:

$$(E_1 + E_2)\cdot \begin{bmatrix} \mathbf{H}' \\ \gamma^i\mathbf{H}'' \\ \gamma^{2i}\mathbf{H}'' \end{bmatrix}^T + E'_3 \cdot \begin{bmatrix} \mathbf{H}' \\ \gamma^j\mathbf{H}'' \\ \gamma^{2j}\mathbf{H}'' \end{bmatrix}^T = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

The relation $(E_1 + E_2 + E'_3)\cdot \mathbf{H}'^T = 0$ leads to $E_1 + E_2 + E'_3 = 0$ because $\mathbf{H}'$ is a $q \times b$ binary matrix whose $\min(2t + t',\ b)$ column vectors are linearly independent. Multiplying $(E_1 + E_2 + E'_3)$ by $\mathbf{H}''^T$ from the right gives $(E_1 + E_2 + E'_3)\cdot \mathbf{H}''^T = 0$. Let $(E_1 + E_2)\cdot \mathbf{H}''^T$ and $E'_3 \cdot \mathbf{H}''^T$ be expressed by x and y, respectively. Then the following relations hold:

$$\begin{cases} x + y = 0, \\ \gamma^i x + \gamma^j y = 0, \\ \gamma^{2i} x + \gamma^{2j} y = 0, \end{cases}$$

where $y \neq 0$. The top two relations can be expressed in the following matrix form:

$$\begin{bmatrix} 1 & 1 \\ \gamma^i & \gamma^j \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{7.28}$$

The 2×2 matrix in this equation is nonsingular because its determinant is a Vandermonde's determinant. Multiplying Eq. (7.28) by the inverse matrix of this 2×2 matrix from the left comes to $x = y = 0$, which is a contradiction because $y \neq 0$.

For other columns of $\mathbf{H}$, we assume that the following equation holds for $(E_1 + E_2) \neq 0$:

$$(E_1 + E_2)\cdot \begin{bmatrix} \mathbf{H}' \\ \gamma^i\mathbf{H}'' \\ \gamma^{2i}\mathbf{H}'' \end{bmatrix}^T + E'_3 \cdot \begin{bmatrix} \mathbf{H}' \\ \mathbf{O} \\ \mathbf{O} \end{bmatrix}^T = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

The relation $(E_1 + E_2 + E'_3)\cdot \mathbf{H}'^T = 0$ leads to $E_1 + E_2 + E'_3 = 0$; that is, $E_1 + E_2 = E'_3$. Multiplying $(E_1 + E_2)$ by $\mathbf{H}''^T$ from the right gives $(E_1 + E_2)\cdot \mathbf{H}''^T = E'_3 \cdot \mathbf{H}''^T \neq 0$, which contradicts to $(E_1 + E_2)\cdot \mathbf{H}''^T = 0$.

It can be easily proved that the following equation holds because $E_3' \cdot \mathbf{H}''^T \neq 0$.

$$E_3' \cdot \begin{bmatrix} \mathbf{H}' \\ \gamma^i \mathbf{H}'' \\ \gamma^{2i} \mathbf{H}'' \end{bmatrix}^T + (E_1 + E_2) \cdot \begin{bmatrix} \mathbf{H}' \\ \mathbf{O} \\ \mathbf{O} \end{bmatrix}^T \neq \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

For all the other combinations of columns of $\mathbf{H}$, condition 2a is proved to be satisfied.

Conditions 2b and 3: These can be proved in the same way as condition 2a. Q.E.D.

The code length is almost doubled every time we add an additional two check bits. In the case where $\max(t, t') = b$, the code given by Theorem 7.29 is identical to the maximum distance separable (MDS) RS code over $GF(2^b)$ with a minimum distance 4 because $\mathbf{H}'$ and $\mathbf{H}''$ are equal to the $b \times b$ identity matrix.

Example 7.4 [SUZU05a]

Figure 7.20 shows an example of a parity-check matrix of the $(79, 64)$ $S_{3/8}EC$-$(S_{3/8}+S)ED$ code given in Theorem 7.29, with parameters of $b = 8$ bits, $t = 3$ bits, $t' = 1$ bit, and information-bit length $K = 64$. The maximum code length of the original code is $N = 136$ bits. Here $\mathbf{H}'$ is a 7×8 matrix whose seven column vectors are linearly independent, and $\mathbf{H}''$ is the 4×8 matrix with $r' = 4$ whose three column vectors are linearly independent. These $\mathbf{H}'$ and $\mathbf{H}''$ matrices are given below in binary form:

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

Let γ be a primitive element of $GF(2^4)$ defined by the primitive polynomial $\mathbf{p}(x) = x^4 + x + 1$. Then $\mathbf{H}''$ is expressed as $[\gamma^0 \quad \gamma^4 \quad \gamma^8 \quad \gamma^{14} \quad \gamma^{10} \quad \gamma^7 \quad \gamma^{13} \quad \gamma^{12}]$, and therefore $\gamma^i \mathbf{H}'' = [\gamma^i \quad \gamma^{i+4} \quad \gamma^{i+8} \quad \gamma^{i+14} \quad \gamma^{i+10} \quad \gamma^{i+7} \quad \gamma^{i+13} \quad \gamma^{i+12}]$, where $0 \leq i \leq 14$.

Decoding Decoding of the $S_{t/b}EC$-$(S_{t/b}+S_{t'/b})ED$ code given by Theorem 7.29 is presented here. Let v, c, and $E = (E_0, E_1, \ldots, E_{n-1})$ be a received word, a codeword,

10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	10000001	1000000
01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	01000001	0100000
00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	00100001	0010000
00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	00010001	0001000
00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	00001001	0000100
00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	00000101	0000010
00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	00000011	0000001
11111111	00010111	00101011	01001101	11101000	00111100	01100110	10100101	11010100	0101101
01001101	11101000	00111100	01100110	10100101	11010100	01011010	11000011	01110001	1000111
00101011	01001101	11101000	00111100	01100110	10100101	11010100	01011010	11000011	0111000
00010111	00101011	01001101	11101000	00111100	01100110	10100101	11010100	01011010	1100001
11111111	00101011	11101000	01100110	11010100	11000011	10001110	10110010	00010111	0100110
01001101	00111100	10100101	01011010	01110001	10011001	11111111	00101011	11101000	0110011
00101011	11101000	01100110	11010100	11000011	10001110	10110010	00010111	01001101	0011110
00010111	01001101	00111100	10100101	01011010	01110001	10011001	11111111	00101011	1110100

Figure 7.20 *Parity-check matrix of the shortened (79, 64) $S_{3/8}EC$-$(S_{3/8}+S)$ ED code.* Source: *[SUZU05a].*

and an error vector, respectively. E_i shows the i-th byte of E for $0 \le i \le n-1$. The syndrome S is calculated by using the matrix $\mathbf{H}$ in Theorem 7.29 such that

$$\begin{aligned} S &= [\, S_0 \quad S_1 \quad S_2 \,] \\ &= v \cdot \mathbf{H}^T = (c+E) \cdot \mathbf{H}^T = E \cdot \mathbf{H}^T, \end{aligned}$$

where $S_0 \in GF(2^q)$ is a q-bit binary row vector and $S_1,\ S_2 \in GF(2^{r'})$ are r'-bit binary row vectors. The decoding is performed for the following syndrome cases:

1. *$S_0 = 0$ and $S_1 = 0$ and $S_2 = 0$. There exist no errors, and hence the received word is correct.*
2. *$S_0 = 0$ and ($S_1 \neq 0$ or $S_2 \neq 0$).* The errors cannot be corrected. In this case uncorrectable errors are detected.
3. *$S_0 \neq 0$ and ($S_1 = 0$ or $S_2 = 0$).* The errors cannot be corrected. In this case uncorrectable errors are detected.
4. *$S_0 \neq 0$ and $S_1 \neq 0$, and $S_2 \neq 0$.* The first element S_0 of S is expressed as follows: $S_0 = \sum_{k=0}^{n-1}\left(E_k \cdot \mathbf{H}'^T\right) = \left(\sum_{k=0}^{n-1} E_k\right) \cdot \mathbf{H}'^T$. Let $\sum_{k=0}^{n-1} E_k$ be E^*. If there exists an E^* that satisfies the relation $E^* \cdot \mathbf{H}'^T = S_0$ uniquely, the error location is found from the error pattern E^* and the syndromes S_1 and S_2. These syndromes must satisfy the relations $E^* \cdot (\gamma^i \mathbf{H}''^T) = S_1$ and $E^* \cdot (\gamma^{2i} \mathbf{H}''^T) = S_2$ for $i = 0, 1, \ldots, n-1$. That is, for $i = 0, 1, \ldots, n-1$, if at particular i, (i.e., $i = j$), both relations $E^* \cdot (\gamma^j \mathbf{H}''^T) = S_1$ and $E^* \cdot (\gamma^{2j} \mathbf{H}''^T) = S_2$ are satisfied, then the error location is determined such that the j-th byte is an erroneous byte. In this case j-th error pattern $E_j (= E^*)$ is also determined, and therefore errors in the j-th byte can be corrected. If the error location is not determined in the above, then uncorrectable errors are detected.

Figure 7.21 shows the decoding flowchart of the code.

(2) ($S_{t/b} + S_{t'/b}$)EC Codes

We move our discusssion here to the distance-5 type complex m-spotty byte error control codes, that is, single t/b-error plus single t'/b-error correcting codes, called m-spotty ($S_{t/b} + S_{t'/b}$)EC codes. These codes correct both single t/b-errors and t'/b-errors that have occurred in two distinct bytes simultaneously, or they can correct single $(t+t')/b$-errors in just one byte.

Code Conditions and Bounds

Theorem 7.30 *Let $\boldsymbol{H}_i$, $i = 0, 1, \ldots, n-1$, be an $R \times b$ submatrix of $\boldsymbol{H} = [\boldsymbol{H}_0\ \boldsymbol{H}_1\ \boldsymbol{H}_2\ \ldots\ \boldsymbol{H}_{n-1}]$. The null space of $\boldsymbol{H}$ is an ($S_{t/b} + S_{t'/b}$)EC code if and only if*

1. $(E_1 + E_2 + E_3 + E_4) \cdot \boldsymbol{H}_i^T \neq 0$ *for* $E_1 + E_2 + E_3 + E_4 \neq 0$,
2a. $(E_1 + E_2) \cdot \boldsymbol{H}_i^T + (E'_3 + E'_4) \cdot \boldsymbol{H}_j^T \neq 0$ *for* $E_1 \neq E_2,\ E'_3 \neq E'_4$,
2b. $(E_1 + E'_3) \cdot \boldsymbol{H}_i^T + (E_2 + E'_4) \cdot \boldsymbol{H}_j^T \neq 0$ *for* $E_1 \neq E'_3,\ E_2 \neq E'_4$,
2c. $(E_1 + E_2 + E'_3) \cdot \boldsymbol{H}_i^T + E'_4 \cdot \boldsymbol{H}_j^T \neq 0$ *for* $E_1 + E_2 + E'_3 \neq 0$,
2d. $(E_1 + E'_3 + E'_4) \cdot \boldsymbol{H}_i^T + E_2 \cdot \boldsymbol{H}_j^T \neq 0$ *for* $E_1 + E'_3 + E'_4 \neq 0$,
3a. $E_1 \cdot \boldsymbol{H}_i^T + E_2 \cdot \boldsymbol{H}_j^T + (E'_3 + E'_4) \cdot \boldsymbol{H}_k^T \neq 0$ *for* $E'_3 \neq E'_4$,
3b. $E_1 \cdot \boldsymbol{H}_i^T + E'_3 \cdot \boldsymbol{H}_j^T + (E_2 + E'_4) \cdot \boldsymbol{H}_k^T \neq 0$ *for* $E_2 \neq E'_4$,

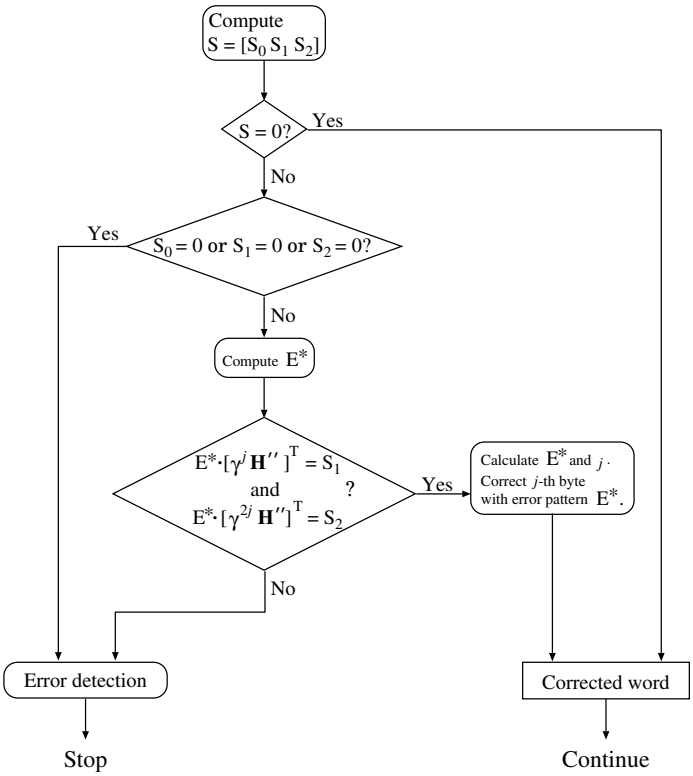


Figure 7.21 *Decoding flowchart of the $S_{t/b}$EC-($S_{t/b} + S_{t'/b}$)ED code.*

3c. $(E_1 + E_2) \cdot \boldsymbol{H}_i^T + E_3' \cdot \boldsymbol{H}_j^T + E_4' \cdot \boldsymbol{H}_k^T \neq 0 \quad$ *for* $E_1 \neq E_2$,

4. $E_1 \cdot \boldsymbol{H}_i^T + E_2 \cdot \boldsymbol{H}_j^T + E_3' \cdot \boldsymbol{H}_k^T + E_4' \cdot \boldsymbol{H}_l^T \neq 0$,

where $\forall E_1,\ E_2 \in \boldsymbol{E}_{t/b},\ \forall E_3',\ E_4' \in \boldsymbol{E}_{t'/b}$, *and* $i,\ j,\ k,\ l$ *are mutually distinct integers satisfying* $0 \leq i,\ j,\ k,\ l \leq n-1$.

The conditions of this theorem ensure that the syndromes created by any single t/b-errors plus single t'/b-errors are all different and not equal to zero. Since the codes are m-spotty byte error correcting codes, it should be noted that such t/b-errors and t'/b-errors that occur in one byte occur as well as in two different bytes.

Theorem 7.31 *A linear* $(N,\ N-R)$ *($S_{t/b} + S_{t'/b}$)EC code requires at least* $2t + 2t'$ *check bits.*

This theorem can be proved in the same way as Theorem 7.27 is proved.

Theorem 7.32 *If N is a multiple of b, a linear $(N,\ N-R)$ ($S_{t/b} + S_{t'/b}$)EC code exists only if*

$$2^R - 1 \geq \frac{N}{b} \cdot \sum_{i=1}^{t+t'} \binom{b}{i} + \frac{N}{b}\left(\frac{N}{b} - 1\right)\left\{\sum_{i=1}^{t} \binom{b}{i}\right\}\left\{\sum_{i=1}^{t'} \binom{b}{i}\right\} - \binom{N/b}{2}\left\{\sum_{i=1}^{\min(t,t')} \binom{b}{i}\right\}^2. \tag{7.29}$$

Proof The total number of single $(t+t')/b$-errors that corrupt a single b-bit byte is given by $\sum_{i=1}^{t+t'} \binom{b}{i}$. There exist N/b bytes in a codeword, and therefore the $(N/b) \cdot \sum_{i=1}^{t+t'} \binom{b}{i}$ syndromes should be distinct.

On the other hand, the total number of single t/b-errors plus single t'/b-errors corrupting two different bytes is given by

$$\binom{N/b}{2}\left\{\sum_{i=1}^{t}\binom{b}{i}\right\}\left\{\sum_{i=1}^{t'}\binom{b}{i}\right\} + \binom{N/b}{2}\left\{\sum_{i=1}^{\min(t,\,t')}\binom{b}{i}\right\}\left\{\sum_{i=\min(t,\,t')+1}^{\max(t,\,t')}\binom{b}{i}\right\}.$$

This leads to

$$\frac{N}{b}\left(\frac{N}{b}-1\right)\left\{\sum_{i=1}^{t}\binom{b}{i}\right\}\left\{\sum_{i=1}^{t'}\binom{b}{i}\right\} - \binom{N/b}{2}\left\{\sum_{i=1}^{\min(t,\,t')}\binom{b}{i}\right\}^2.$$

Consequently the relation in this theorem holds. Q.E.D.

Code Design

Definition 7.7 Let $\mathbf{H}' = \left[h'_0\ h'_1\ \cdots\ h'_{b-1}\right]$ be a $q \times b$ binary matrix whose $2t+2t'$ column vectors are linearly independent, where $h'_0,\ h'_1,\ \ldots,\ h'_{b-1}$ are binary column vectors of $GF(2^q)$. Also let $\mathbf{H}'' = \left[h''_0\ h''_1\ \cdots\ h''_{b-1}\right]$ be an $r \times b$ binary matrix whose $t+t'$ column vectors are linearly independent, where $h''_0,\ h''_1,\ \ldots,\ h''_{b-1}$ are binary column vectors of $GF(2^r)$. □

The matrix $\mathbf{H}'$ is a $b \times b$ nonsingular matrix including a $b \times b$ identity matrix for $\min(2t+2t',\ b) = b$. On the other hand, for $\min(2t+2t',\ b) = 2t+2t' < b$, the matrix $\mathbf{H}'$ is a parity-check matrix of a linear binary $(b,\ b-q)$ code with minimum Hamming distance $2t+2t'+1$. For $\min(t+t',\ b) = b$, the matrix $\mathbf{H}''$ is a $b \times b$ nonsingular matrix that includes a $b \times b$ identity matrix. For $\min(t+t',\ b) = t+t' < b$, the matrix $\mathbf{H}''$ is a parity-check matrix of a linear binary $(b,\ b-r)$ code with minimum Hamming distance $t+t'+1$.

We use the above-defined $\mathbf{H}'$ and $\mathbf{H}''$ in the following theorem to design the $(S_{t/b}+S_{t'/b})$EC codes.

Theorem 7.33 *Let γ be a primitive element of $GF(2^{r'})$, where $r' \geq r$. The null space of*

$$\boldsymbol{H} = \left[\begin{array}{ccccc|cc} \boldsymbol{H}' & \boldsymbol{H}' & \boldsymbol{H}' & \cdots & \boldsymbol{H}' & \boldsymbol{H}' & \boldsymbol{O}_{q\times r'} \\ \gamma^0\boldsymbol{H}'' & \gamma^1\boldsymbol{H}'' & \gamma^2\boldsymbol{H}'' & \cdots & \gamma^{(n-1)}\boldsymbol{H}'' & \boldsymbol{O}_{r'\times b} & \boldsymbol{O}_{r'\times r'} \\ \gamma^0\boldsymbol{H}'' & \gamma^2\boldsymbol{H}'' & \gamma^4\boldsymbol{H}'' & \cdots & \gamma^{2(n-1)}\boldsymbol{H}'' & \boldsymbol{O}_{r'\times b} & \boldsymbol{O}_{r'\times r'} \\ \gamma^0\boldsymbol{H}'' & \gamma^3\boldsymbol{H}'' & \gamma^6\boldsymbol{H}'' & \cdots & \gamma^{3(n-1)}\boldsymbol{H}'' & \boldsymbol{O}_{r'\times b} & \boldsymbol{I}_{r'} \end{array}\right]$$

is an $(S_{t/b} + S_{t'/b})$EC code with check-bit length $R = q + 3r'$ and code length in bits $N = (n+1) \cdot b + r'$, where $n = 2^{r'} - 1$, $\boldsymbol{O}_{q\times r'}$ is a $q \times r'$ zero matrix, $\boldsymbol{O}_{r'\times b}$ is an $r' \times b$ zero matrix, $\boldsymbol{O}_{r'\times r'}$ is an $r' \times r'$ zero matrix, $\boldsymbol{I}_{r'}$ is an $r' \times r'$ identity matrix,

$\gamma^i \boldsymbol{H}'' = \left[\gamma^i \phi(h''_0) \quad \gamma^i \phi(h''_1) \quad \cdots \quad \gamma^i \phi(h''_{b-1})\right]$ *for* $0 \le i \le n-1$, *and* $\phi : GF(2^r) \to GF(2^{r'})$ *is an injective homomorphism of* $GF(2^r)$ *into* $GF(2^{r'})$ *under addition.*

This theorem can be proved in the same way as Theorem 7.29 is proved.

Example 7.5 [SUZU05a]

Shown in Figure 7.22 is an example of a parity-check matrix of the (90, 64) $(S_{3/8} + S)$EC code given in Theorem 7.33 with parameters of $b = 8$ bits, $t = 3$ bits, $t' = 1$ bit, and information-bit length $K = 64$. The maximum code length of the original code is $N = 518$ bits. In the figure $\mathbf{H}'$ is an 8×8 identity matrix, and $\mathbf{H}''$ is the following 6×8 matrix with $r' = 6$, whose four column vectors are linearly independent:

$$\mathbf{H}'' = \left[\gamma^0 \quad \gamma^1 \quad \gamma^2 \quad \gamma^3 \quad \gamma^4 \quad \gamma^5 \quad \gamma^{18} \quad \gamma^{20}\right]$$
$$= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

Here γ is a primitive element of $GF(2^6)$ defined by the primitive polynomial $\mathbf{p}(x) = x^6 + x + 1$. Then $\gamma^i \mathbf{H}'' = [\gamma^i \ \gamma^{i+1} \ \gamma^{i+2} \ \gamma^{i+3} \ \gamma^{i+4} \ \gamma^{i+5} \ \gamma^{i+18} \ \gamma^{i+20}]$, where $0 \le i \le 62$.

Decoding To decode the $(S_{t/b}+S_{t'/b})$EC code given by Theorem 7.33 we proceed as follows: Let v, c, and $E = (E_0 \ E_1 \ \ldots \ E_{n-1})$ be the received word, the codeword, and the error vector, respectively. E_i shows the i-th byte of E for $0 \le i \le n-1$. Then the syndrome S is calculated as

$$S = [S_0 \quad S_1 \quad S_2 \quad S_3]$$
$$= v \cdot \mathbf{H}^T = (c + E) \cdot \mathbf{H}^T = E \cdot \mathbf{H}^T,$$

```
10000000 10000000 10000000 10000000 10000000 10000000 10000000 10000000 10000000 10000000 10000000 10
01000000 01000000 01000000 01000000 01000000 01000000 01000000 01000000 01000000 01000000 01000000 01
00100000 00100000 00100000 00100000 00100000 00100000 00100000 00100000 00100000 00100000 00100000 00
00010000 00010000 00010000 00010000 00010000 00010000 00010000 00010000 00010000 00010000 00010000 00
00001000 00001000 00001000 00001000 00001000 00001000 00001000 00001000 00001000 00001000 00001000 00
00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00
00000010 00000010 00000010 00000010 00000010 00000010 00000010 00000010 00000010 00000010 00000010 00
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001 00
10000010 00000101 00001001 00010011 00100011 01000010 10000111 00001100 00011010 00110000 01100001 11
01000010 10000111 00001100 00011010 00110000 01100001 11000101 10001011 00010110 00101010 01010001 10
00100011 01000010 10000111 00001100 00011010 00110000 01100001 11000101 10001011 00010110 00101010 01
00010011 00100011 01000010 10000111 00001100 00011010 00110000 01100001 11000101 10001011 00010110 00
00001001 00010011 00100011 01000010 10000111 00001100 00011010 00110000 01100001 11000101 10001011 00
00000101 00001001 00010011 00100011 01000010 10000111 00001100 00011010 00110000 01100001 11000101 10
10000010 00001001 00100011 10000111 00011010 01100001 10001011 00101010 10100100 10011101 01111011 11
01000010 00001100 00110000 11000101 00010110 01010001 01001110 00111100 11110101 11010011 01000111 00
00100011 10000111 00011010 01100001 10001011 00101010 10100100 10011101 01111011 11101010 10100001 10
00010011 01000010 00001100 00110000 11000101 00010110 01010001 01001110 00111100 11110101 11010011 01
00001001 00100011 10000111 00011010 01100001 10001011 00101010 10100100 10011101 01111011 11101010 10
00000101 00010011 01000010 00001100 00110000 11000101 00010110 01010001 01001110 00111100 11110101 11
10000010 00010011 10000111 00110000 10001011 01010001 10011101 11110101 10100001 00011111 11100110 00
01000010 00011010 11000101 00101010 01001110 01111011 11010011 10001110 01110010 10010001 10010100 10
00100011 00001100 01100001 00010110 10100100 00111100 11101010 01000111 00111001 11001001 01001011 01
00010011 10000111 00110000 10001011 01010001 10011101 11110101 10100001 00011111 11100110 00100110 00
00001001 01000010 00011010 11000101 00101010 01001110 01111011 11010011 10001110 01110010 10010001 10
00000101 00100011 00001100 01100001 00010110 10100100 00111100 11101010 01000111 00111001 11001001 01
```

Figure 7.22 *Parity-check matrix of the shortened (90, 64)* $(S_{3/8}+S)$*EC code.* Source: *[SUZU05a].* © *2005 IEEE.*

where $S_0 \in GF(2^q)$ is a q-bit binary row vector and each $S_1, S_2, S_3 \in GF(2^{r'})$ is an r'-bit binary row vector:

$$\begin{aligned} S &= [\, S_0 \quad S_1 \quad S_2 \quad S_3 \,] \\ &= \left[\sum_{i=0}^{n-1} \left(E_i \cdot \mathbf{H}'^T \right) \quad \sum_{i=0}^{n-1} \left(E_i \cdot \gamma^i \mathbf{H}''^T \right) \quad \sum_{i=0}^{n-1} \left(E_i \cdot \gamma^{2i} \mathbf{H}''^T \right) \quad \sum_{i=0}^{n-1} \left(E_i \cdot \gamma^{3i} \mathbf{H}''^T \right) \right] \end{aligned} \tag{7.30}$$

The syndrome S_0 in S can be expressed as

$$S_0 = \sum_{i=0}^{n-1} \left(E_i \cdot \mathbf{H}'^T \right) = \left(\sum_{i=0}^{n-1} E_i \right) \cdot \mathbf{H}'^T.$$

Next we determine $\sum_{i=0}^{n-1} E_i$ by using the syndrome S_0 because $\mathbf{H}'$ is a parity-check matrix of $(b,\ b-q)\ (t+t')$-bit error correcting code. Let $\sum_{i=0}^{n-1} E_i$ be E^*. Multiplying this by $\gamma^0 \mathbf{H}''^T$ from the right gives $S_0' = E^* \cdot (\gamma^0 \mathbf{H}''^T)$. Also we have $E_i \cdot (\gamma^i \mathbf{H}''^T) = \gamma^i (E_i \cdot \mathbf{H}''^T)$. Then the syndrome is newly represented by $S' = [S_0'\ S_1\ S_2\ S_3]$. We let $\varepsilon_i = E_i \cdot \mathbf{H}''^T$ for $i = 1, 2, \ldots, n-1$. Then

$$\begin{aligned} S' &= [\, S_0' \quad S_1 \quad S_2 \quad S_3 \,] \\ &= \left[\sum_{i=0}^{n-1} \varepsilon_i \quad \sum_{i=0}^{n-1} (\gamma^i \varepsilon_i) \quad \sum_{i=0}^{n-1} (\gamma^{2i} \varepsilon_i) \quad \sum_{i=0}^{n-1} (\gamma^{3i} \varepsilon_i) \right]. \end{aligned} \tag{7.31}$$

This syndrome is identical to that of the RS code with distance 5 over $GF(2^{r'})$. Then the error patterns of $GF(2^{r'})$ and the error locations are determined by using the existing decoding algorithm of the RS code. If the number of erroneous bytes is one, the byte error pattern is $E^* \in GF(2^b)$, as was given above. If the number of erroneous bytes is two, one byte has t or fewer bits in error and another has t' or fewer bits in error. The error patterns E_x and E_y of $GF(2^b)$, where $0 \le x,\ y \le n-1$, and $x \ne y$, are transformed by the corresponding error patterns ε_x and ε_y of $GF(2^{r'})$. At least one of the error patterns can be determined because $\mathbf{H}''$ is a parity-check matrix of $\min(t,\ t')$ bits error correcting and $\max(t,\ t')$ bits error detecting code. If one of these two error patterns is determined and another is not determined but is detected, then the detected error pattern can be obtained by sum of E^* and the determined error pattern.

Figure 7.23 shows the decoding flowchart of the code.

Evaluation Figure 7.24 shows the relation between the information-bit length and the check-bit length of the $S_{t/b}$EC-($S_{t/b}$+$S_{t'/b}$)ED code and its bound at $b = 8$ bits, $t = 3$ bits, and $t' = 1$ bit, along with the single-byte error correcting and double-byte error detecting code (i.e., the S8EC-D8ED code) and the single t/b-error correcting and double t/b-error detecting code (i.e., the $S_{3/8}$EC-$D_{3/8}$ED code) [SUZU04]. The bound in the figure is as given by Theorems 7.27 and 7.28. Figure 7.25 further shows the relation of the ($S_{t/b}$+$S_{t'/b}$)EC code and its bound at $b = 8$ bits, $t = 3$ bits, and $t' = 1$ bit, along with the D8EC code and the m-spotty $D_{3/8}$EC code [SUZU04]. The bound in the figure is as given by Theorems 7.31 and 7.32.

Tables 7.6 and 7.7 show the error detection capabilities of the $S_{3/8}$EC-($S_{3/8}$+S)ED codes and the ($S_{3/8}$+S)EC codes for three types of errors that are beyond the error control

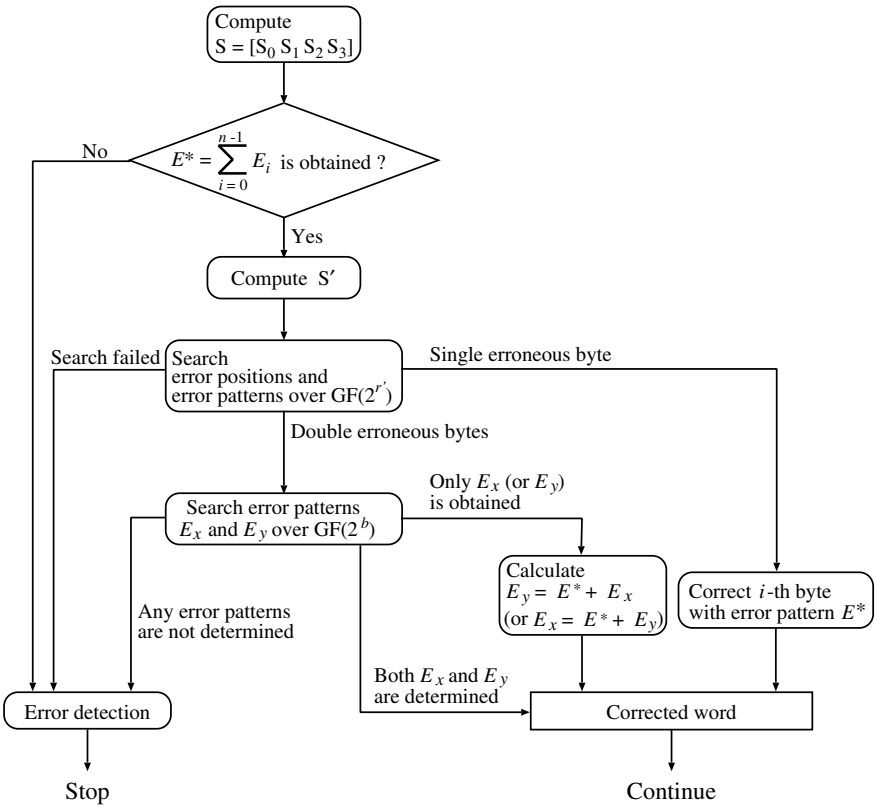


Figure 7.23 *Decoding flowchart of the* $(S_{t/b}+S_{t'/b})$*EC code.*

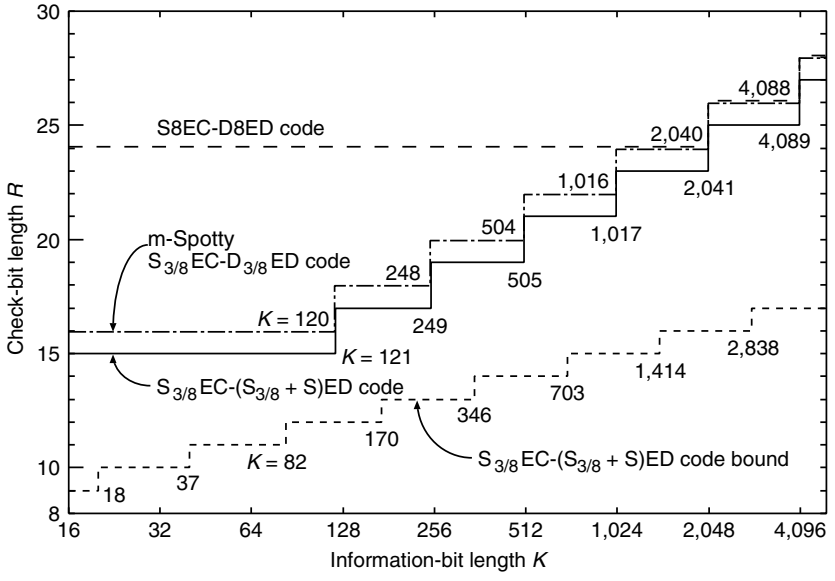


Figure 7.24 *Check-bit lengths compared with information-bit lengths of the* $S_{3/8}$*EC-*$(S_{3/8}+S)$*ED codes, along with those of the m-spotty* $S_{3/8}$*EC-*$D_{3/8}$*ED codes and the S8EC-D8ED codes.* Source: *[SUZU05a].*

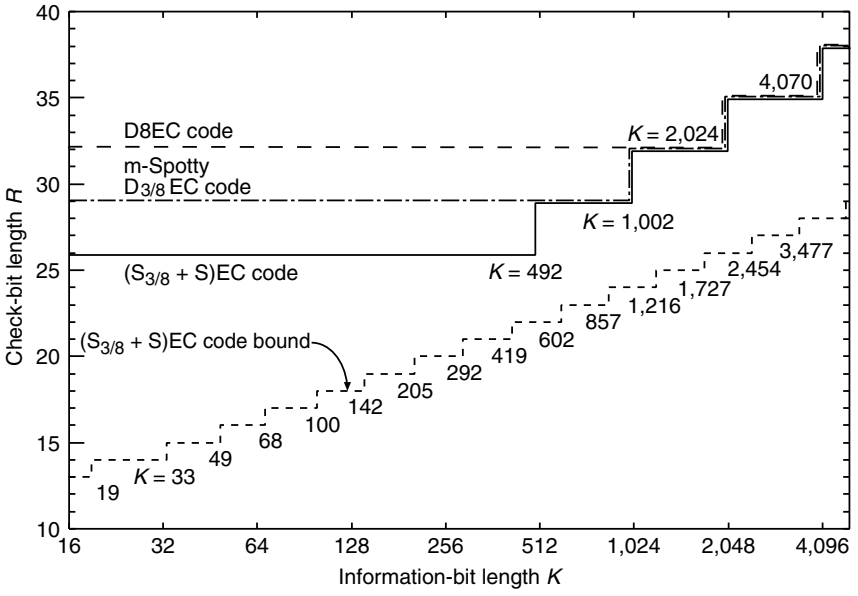


Figure 7.25 *Check-bit lengths compared with information-bit lengths of the ($S_{3/8}$+S)EC codes, along with those of the $D_{3/8}$EC codes and the D8EC codes.* Source: *[SUZU05a]. © 2005 IEEE.*

TABLE 7.6 Error-Detection Capabilities of the $S_{3/8}$EC-($S_{3/8}$+S)ED Codes for Three Types of Errors

	Error detection capability (%)		
Errors	$K = 64$ ($R = 15$)	$K = 128$ ($R = 17$)	$K = 256$ ($R = 19$)
Triple-bit errors	97.57	98.34	98.76
Double-byte errors	91.64	91.47	91.48
Byte plus bit errors	94.26	94.07	94.09

TABLE 7.7 Error Detection Capabilities of the ($S_{3/8}$+S)EC Codes for Three Types of Errors

	Error detection capability (%)		
Errors	$K = 64$ ($R = 26$)	$K = 128$ ($R = 26$)	$K = 256$ ($R = 26$)
Triple-bit errors	99.97	99.63	98.45
Double-byte errors	92.02	92.02	92.01
Byte plus bit errors	99.51	99.51	99.51

TABLE 7.8 Check-Bit Lengths of the $S_{t/16}EC$-$(S_{t/16}+S_{t'/16})ED$ Codes and the $(S_{t/16}+S_{t'/16})EC$ Codes $(t' < t)$ for $K = 256$ Bits

Codes	t'	t						
		2	3	4	5	6	7	8
$S_{t/16}EC$-$(S_{t/16}+S_{t'/16})ED$	1	19	22	30	33	37	39	48
	2	—	24	31	33	37	40	48
	3	—	—	31	33	37	40	48
	4	—	—	—	33	38	40	49
m-Spotty $S_{t/16}EC$-$D_{t/16}ED$		21	24	31	33	38	40	48
$(S_{t/16}+S_{t'/16})EC$	1	26	38	42	48	51	58	58
	2	—	42	48	51	58	58	61
	3	—	—	51	58	58	61	61
	4	—	—	—	58	61	61	61
	5	—	—	—	—	61	61	61
	6	—	—	—	—	—	61	61
m-Spotty $D_{t/16}EC$		32	44	56	60	60	60	64

capabilities of these codes. Since these codes are m-spotty byte error control codes, it is noted that the $S_{3/8}EC$-$(S_{3/8}+S)ED$ code can detect any 4-bit errors in a byte as well as simultaneously detect any single $3/8$-error in a byte and single-bit error in another byte. Also the $(S_{3/8}+S)EC$ code can correct any 4 bits errors in a byte as well as simultaneously correct any single $3/8$-error in a byte and single-bit error in another byte.

Table 7.8 shows the check-bit lengths of the $S_{t/16}EC$-$(S_{t/16}+S_{t'/16})$ ED codes and the $(S_{t/16}+S_{t'/16})$ EC codes, compared to the corresponding m-spotty byte error control codes for $b = 16$ bits and $K = 256$ bits. Filled in are the check-bit lengths of these complex spotty byte error control codes that are smaller than those of the corresponding m-spotty byte error control codes.

3. A Class of Codes for m-Spotty Byte Errors Occurred in a Limited Number of Bytes

In large-capacity semiconductor memory systems, errors occur usually in a limited number of RAM chips at a time, for example, at most two or three chips at the same address [VAID92]. For such systems we must consider a different class of m-spotty byte error control codes where the errors are confined to a small number of bytes. Fewer number of check bits can be expected than the preceding discussion's m-spotty byte error control codes [SUZU05b].

Preliminaries We denote the codes correcting μ_1 m-spotty byte errors in p_1 bytes and detecting μ_2 m-spotty byte errors in p_2 bytes as $[\mu_1\ _{t/b}EC]_{p_1}$-$[\mu_2\ _{t/b}ED]_{p_2}$ codes, where $\mu_1 \geq p_1$, and $\mu_2 \geq p_2$. More precisely, the codes correct μ_1 t/b-errors that occur in less than or equal to p_1 bytes, and detect μ_2 t/b-errors that occur in less than or equal to p_2 bytes. If $\mu_1 = \mu_2$ and $p_1 = p_2$, then these codes correct μ_1 m-spotty byte errors in p_1 bytes, and are simply expressed by $[\mu_{t/b}EC]_p$ codes, where $\mu = \mu_1$ and $p = p_1$. And if $\mu_1 = 0$ and $p_1 = 0$, then these codes detect μ_2 m-spotty byte errors in p_2 bytes, and are expressed by $[\mu_{t/b}ED]_p$ codes where $\mu = \mu_2$ and $p = p_2$. Unless otherwise noted, the codes are denoted as code $\boldsymbol{C}$ in the rest of this chapter.

Theorem 7.34 *Let $\boldsymbol{H}_i$ be an $R \times b$ binary submatrix for $0 \le i \le n-1$. The null space of $\boldsymbol{H} = [\boldsymbol{H}_0\ \boldsymbol{H}_1\ \boldsymbol{H}_2\ \boldsymbol{H}_3\ \cdots\ \boldsymbol{H}_{n-1}]$ is a code $\boldsymbol{C}$ if and only if the following relation is satisfied:*

$$
\begin{aligned}
&E_1 \cdot \boldsymbol{H}_{i_1}^T + \cdots + E_{v_1} \cdot \boldsymbol{H}_{i_{v_1}}^T + E_{v_1+1} \cdot \boldsymbol{H}_{i_{v_1+1}}^T + \cdots + E_{v_1+v_2} \cdot \boldsymbol{H}_{i_{v_1+v_2}}^T + \cdots \\
&\quad + E_{v_1+v_2+\cdots+v_{\lambda-1}+1} \cdot \boldsymbol{H}_{i_{v_1+v_2+\ldots+v_{\lambda-1}+1}}^T + \cdots + E_{v_1+v_2+\cdots+v_\lambda} \cdot \boldsymbol{H}_{i_{v_1+v_2+\cdots+v_\lambda}}^T \neq 0 \qquad (7.32) \\
&\textit{for } w_{\mathrm{M}}(E_1) = \cdots = w_{\mathrm{M}}(E_{v_1}) = 1,\ w_{\mathrm{M}}(E_{v_1+1}) = \cdots = w_{\mathrm{M}}(E_{v_1+v_2}) = 2, \\
&\qquad \ldots,\ w_{\mathrm{M}}(E_{v_1+v_2+\ldots+v_{\lambda-1}+1}) = \cdots = w_{\mathrm{M}}(E_{v_1+v_2+\ldots+v_\lambda}) = \lambda, \\
&\qquad 0 < \sum_{x=1}^{v_1+\cdots+v_\lambda} w_{\mathrm{M}}(E_x) \le \mu_1 + \mu_2, \quad 0 < \sum_{x=1}^{\lambda} v_x \le p_1 + p_2, \text{and} \\
&\qquad 0 \le v_1 \le p_1 + p_2, \quad 0 \le v_2 \le \min(\lfloor (\mu_1 + \mu_2)/2 \rfloor,\ p_1 + p_2), \\
&\qquad\qquad \ldots, 0 \le v_\lambda \le \min(\lfloor (\mu_1 + \mu_2)/\lambda \rfloor,\ p_1 + p_2),
\end{aligned}
$$

where E_j is the j-th byte error in error vector E, meaning $E_j \in \boldsymbol{E}_{t/b} = \{E \in GF(2^b) \mid 1 \le w_H(E) \le t\}$, $w_M(E_j) = \sum_{j=0}^{n-1} \lceil w_H(E_j)/t \rceil (j = 1, 2, \ldots, v_1, v_1+1, \ldots, v_1+v_2+\ldots+v_\lambda)$, $\lambda = \lceil b/t \rceil$, and $i_1,\ i_2,\ \cdots,\ i_{v_1+v_2+\ldots+v_\lambda}$ are mutually distinct integers of i satisfying $0 \le i_1, i_2,\ \cdots,\ i_{v_1+v_2+\ldots+v_\lambda} \le n-1$.

Proof Let an error set having $l_1 (\le \mu_1)$ spotty byte errors that occur in $\rho_1 (\le p_1)$ bytes be $\mathbf{E}_i = \{E_{i_1},\ E_{i_2},\ \ldots,\ E_{i_{\rho_1}}\}$, where $w_{\mathrm{M}}(E_{i_x}) \le \lambda$, and $\sum_{x=1}^{\rho_1} w_{\mathrm{M}}(E_{i_x}) = l_1$. Also let an error set having $l_2 (\le \mu_2)$ spotty byte errors that occur in $\rho_2 (\le p_2)$ bytes be $\mathbf{E}_j = \{E_{j_1}, E_{j_2},\ \ldots,\ E_{j_{\rho_2}}\}$, where $w_{\mathrm{M}}(E_{j_x}) \le \lambda$, and $\sum_{x=1}^{\rho_2} w_{\mathrm{M}}(E_{j_x}) = l_2$. The equation $\sum_{x=1}^{\rho_1} w_{\mathrm{M}}(E_{i_x}) = l_1$ shows that the total number of spotty byte errors occurring in different $\rho_1 (\le l_1)$ bytes is l_1. The same holds for $\sum_{x=1}^{\rho_2} w_{\mathrm{M}}(E_{j_x}) = l_2$; that is, this equation shows that the total number of spotty byte errors occurring in different $\rho_2 (\le l_2)$ bytes is l_2. The code $\boldsymbol{C}$ should satisfy the following relation between the error patterns in $\mathbf{E}_i$ and those in $\mathbf{E}_j$:

$$
E_{i_1} \cdot \mathbf{H}_{i_1}^T + E_{i_2} \cdot \mathbf{H}_{i_2}^T + \cdots + E_{i_{\rho_1}} \cdot \mathbf{H}_{i_{\rho_1}}^T \neq E_{j_1} \cdot \mathbf{H}_{j_1}^T + E_{j_2} \cdot \mathbf{H}_{j_2}^T + \cdots + E_{j_{\rho_2}} \cdot \mathbf{H}_{j_{\rho_2}}^T.
$$

Spotty byte errors in $\mathbf{E}_i$ and $\mathbf{E}_j$ can occur in the same byte. To see this, assume that these errors occur in V bytes, where $0 \le V \le \lfloor (p_1 + p_2)/2 \rfloor$. Then the following relation holds:

$$
\begin{aligned}
&E_{i_1} \cdot \mathbf{H}_{i_1}^T + E_{i_2} \cdot \mathbf{H}_{i_2}^T + \cdots + E_{i_V} \cdot \mathbf{H}_{i_V}^T + E_{i_{V+1}} \cdot \mathbf{H}_{i_{V+1}}^T + \cdots + E_{i_{\rho_1}} \cdot \mathbf{H}_{i_{\rho_1}}^T \\
&\neq E_{j_1} \cdot \mathbf{H}_{i_1}^T + E_{j_2} \cdot \mathbf{H}_{i_2}^T + \cdots + E_{j_V} \cdot \mathbf{H}_{i_V}^T + E_{j_{V+1}} \cdot \mathbf{H}_{i_{V+1}}^T + \cdots + E_{j_{\rho_2}} \cdot \mathbf{H}_{j_{\rho_2}}^T.
\end{aligned}
$$

From this relation we have

$$
\begin{aligned}
&(E_{i_1} + E_{j_1}) \cdot \mathbf{H}_{i_1}^T + (E_{i_2} + E_{j_2}) \cdot \mathbf{H}_{i_2}^T + \cdots + (E_{i_V} + E_{j_V}) \cdot \mathbf{H}_{i_V}^T \\
&+ E_{i_{V+1}} \cdot \mathbf{H}_{i_{V+1}}^T + \cdots + E_{i_{\rho_1}} \cdot \mathbf{H}_{i_{\rho_1}}^T + E_{j_{V+1}} \cdot \mathbf{H}_{j_{V+1}}^T + \cdots + E_{j_{\rho_2}} \cdot \mathbf{H}_{j_{\rho_2}}^T \neq 0,
\end{aligned}
$$

where $w_{\mathrm{M}}(E_{i_x} + E_{j_x}) \le \lambda$ for $x = 1,\ 2,\ \ldots,\ V$. Equation (7.32) is obtained by replacing $(E_{i_x} + E_{j_x})$ above by E_{i_x} for $x = 1,\ 2,\ \ldots,\ V$. In this case, $\rho_1 + \rho_2 - V = v_1 + v_2 + \cdots + v_\lambda$, so the relation in Theorem 7.34 holds. Q.E.D.

Theorem 7.35 *A linear code* $\boldsymbol{C}$ *requires at least* $(\mu_1 + \mu_2)t$ *check bits.*

Proof According to Theorem 7.34, $(\mu_1 + \mu_2)t$ binary columns of the parity-check matrix of this code are linearly independent. Therefore this code requires at least $(\mu_1 + \mu_2)t$ check bits. Q.E.D.

Theorem 7.36 *If* N *is a multiple of* b*, a linear* $(N,\ N - R)$ $[\mu_{t/b}EC]_p$ *code exists only if*

$$2^R - 1 \geq \sum_{j=1}^{\mu} S_j^p\left(\frac{N}{b}\right),$$

where $S_j^p(N/b)$ *is the total number of* j *spotty byte errors occurred in less than or equal to* p *bytes, and is expressed by*

$$S_j^p(n) = \sum_{\substack{\delta_1,\ \delta_2,\ \cdots,\ \delta_\lambda \geq 0 \\ \delta_1+\delta_2+\cdots+\delta_\lambda \leq p \\ \delta_1+2\delta_2+\cdots+\lambda\delta_\lambda = j}} \left\{ \binom{n}{\delta_1 + \delta_2 + \cdots + \delta_\lambda} \times \binom{\delta_1 + \delta_2 + \cdots + \delta_\lambda}{\delta_1,\ \delta_2,\ \ldots,\ \delta_\lambda} \times \prod_{z=1}^{\lambda} \left\{ \sum_{i=(z-1)\cdot t+1}^{\min(z\cdot t,\ b)} \binom{b}{i} \right\}^{\delta_z} \right\},$$

$N/b = n$, *and*

$$\binom{\delta_1 + \delta_2 + \cdots + \delta_\lambda}{\delta_1,\ \delta_2,\ \ldots,\ \delta_\lambda} = \frac{(\delta_1 + \delta_2 + \cdots + \delta_\lambda)!}{\delta_1! \times \delta_2! \times \cdots \times \delta_\lambda!}.$$

Code Design

Definition 7.8 Let $\mathbf{H}' = \left[h'_0\ h'_1\ \cdots\ h'_{b-1}\right]$ be a $q \times b$ binary matrix whose $\min((\mu_1 + \mu_2)t,\ b)$ column vectors are linearly independent, where $h'_0,\ h'_1,\ \ldots,\ h'_{b-1}$ are binary column vectors of $GF(2^q)$. Also let $\mathbf{H}'' = \left[h''_0\ h''_1\ \cdots\ h''_{b-1}\right]$ be an $r \times b$ binary matrix whose $\min(\lfloor(\mu_1 + \mu_2)/2\rfloor t,\ b)$ column vectors are linearly independent, where $h''_0,\ h''_1,\ \ldots,\ h''_{b-1}$ are binary column vectors of $GF(2^r)$. □

The matrix $\mathbf{H}'$ is a $b \times b$ nonsingular matrix including a $b \times b$ identity matrix for $\min((\mu_1 + \mu_2)t,\ b) = b$. On the other hand, for $\min((\mu_1 + \mu_2)t,\ b) = (\mu_1 + \mu_2)t < b$, the matrix $\mathbf{H}'$ is a parity-check matrix of a linear binary $(b,\ b - q)$ code with minimum Hamming distance $(\mu_1 + \mu_2)t + 1$. The matrix $\mathbf{H}''$ is also a $b \times b$ nonsingular matrix including a $b \times b$ identity matrix for $\min(\lfloor(\mu_1 + \mu_2)/2\rfloor t,\ b) = b$. On the other hand, for $\min(\lfloor(\mu_1 + \mu_2)/2\rfloor t,\ b) = \lfloor(\mu_1 + \mu_2)/2\rfloor t < b$, the matrix $\mathbf{H}''$ is a parity-check matrix of a linear binary $(b,\ b - r)$ code with minimum Hamming distance $\lfloor(\mu_1 + \mu_2)/2\rfloor t + 1$.

We use the above-defined $\mathbf{H}'$ and $\mathbf{H}''$ in the following theorems to design the $[\mu_{1\ t/b}\mathrm{EC}]_{p_1}$-$[\mu_{2\ t/b}\mathrm{ED}]_{p_2}$ code $\boldsymbol{C}$.

Theorem 7.37 *Let γ be a primitive element of $GF(2^r)$. The null space of*

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{H}' & \boldsymbol{H}' & \cdots & \boldsymbol{H}' \\ \gamma^0\boldsymbol{H}'' & \gamma^1\boldsymbol{H}'' & \cdots & \gamma^{n-1}\boldsymbol{H}'' \\ \gamma^0\boldsymbol{H}'' & \gamma^2\boldsymbol{H}'' & \cdots & \gamma^{2(n-1)}\boldsymbol{H}'' \\ \vdots & \vdots & \ddots & \vdots \\ \gamma^0\boldsymbol{H}'' & \gamma^{(p_1+p_2-1)}\boldsymbol{H}'' & \cdots & \gamma^{(p_1+p_2-1)(n-1)}\boldsymbol{H}'' \end{bmatrix}$$

is a code $\boldsymbol{C}$ with check-bit length $R = q + (p_1 + p_2 - 1)r$ and code length in bits $N = n \cdot b$, where $n = 2^r - 1$ and $\gamma^i\boldsymbol{H}'' = \begin{bmatrix} \gamma^i h''_0 & \gamma^i h''_1 & \cdots & \gamma^i h''_{b-1} \end{bmatrix}$ for $0 \le i \le n-1$.

Proof Without loss of generality, we assume that Eq. (7.33) holds for $E_{v_1+1} \neq 0$, $E_{v_1+2} \neq 0$, ..., and $E_{v_1+v_2+\cdots+v_\lambda} \neq 0$.

$$E_1 \cdot \begin{bmatrix} \mathbf{H}' \\ \gamma^{i_1}\mathbf{H}'' \\ \gamma^{2i_1}\mathbf{H}'' \\ \vdots \\ \gamma^{(p_1+p_2-1)i_1}\mathbf{H}'' \end{bmatrix}^T + \cdots + E_{v_1+1} \cdot \begin{bmatrix} \mathbf{H}' \\ \gamma^{i_{v_1+1}}\mathbf{H}'' \\ \gamma^{2i_{v_1+1}}\mathbf{H}'' \\ \vdots \\ \gamma^{(p_1+p_2-1)i_{v_1+1}}\mathbf{H}'' \end{bmatrix}^T$$

$$+ \cdots + E_{v_1+v_2+\cdots+v_\lambda} \cdot \begin{bmatrix} \mathbf{H}' \\ \gamma^{i_{v_1+v_2+\cdots+v_{\lambda-1}+v_\lambda}}\mathbf{H}'' \\ \gamma^{2i_{v_1+v_2+\cdots+v_{\lambda-1}+v_\lambda}}\mathbf{H}'' \\ \vdots \\ \gamma^{(p_1+p_2-1)i_{v_1+v_2+\cdots+v_{\lambda-1}+v_\lambda}}\mathbf{H}'' \end{bmatrix}^T = \begin{bmatrix} 0^T \\ 0^T \\ 0^T \\ \vdots \\ 0^T \end{bmatrix}. \qquad (7.33)$$

The relation $\left(\sum_{x=1}^{v_1+v_2+\cdots+v_\lambda} E_x\right) \cdot \mathbf{H}'^T = 0$ leads to $\sum_{x=1}^{v_1+v_2+\cdots+v_\lambda} E_x = 0$ because $\mathbf{H}'$ is a $q \times b$ binary matrix whose $\min((\mu_1 + \mu_2)t, b)$ column vectors are linearly independent. Multiplying this relation by $\mathbf{H}''^T$ from the right gives $\left(\sum_{x=1}^{v_1+v_2+\cdots+v_\lambda} E_x\right) \cdot \mathbf{H}''^T = 0$. Next we let $E_1 \cdot \mathbf{H}''^T$, $E_2 \cdot \mathbf{H}''^T$, ..., and $(E_{v_1+v_2+\cdots+v_\lambda}) \cdot \mathbf{H}''^T$ be expressed as x_1, x_2, ..., and $x_{v_1+v_2+\cdots+v_\lambda}$, respectively. Since $\mathbf{H}''$ is a parity-check matrix of a linear binary $(b,\ b-r)$ code with minimum Hamming distance $\lfloor(\mu_1+\mu_2)/2\rfloor t + 1$, then $x_1 \neq 0$, $x_2 \neq 0$, ..., $x_{v_1+v_2+\cdots+v_{\lfloor(\mu_1+\mu_2)/2\rfloor}} \neq 0$. So the following relations hold:

$$\begin{cases} x_1 + x_2 + \cdots + x_{v_1+v_2+\cdots+v_\lambda} = 0 \\ \gamma^{i_1}x_1 + \gamma^{i_2}x_2 + \cdots + \gamma^{i_{v_1+v_2+\cdots+v_\lambda}}x_{v_1+v_2+\cdots+v_\lambda} = 0 \\ \qquad\qquad \cdots \\ \gamma^{(p_1+p_2-1)i_1}x_1 + \gamma^{(p_1+p_2-1)i_2}x_2 + \cdots + \gamma^{(p_1+p_2-1)i_{v_1+v_2+\cdots+v_\lambda}}x_{v_1+v_2+\cdots+v_\lambda} = 0. \end{cases}$$

The coefficient matrix of the top $v_1 + v_2 + \cdots + v_\lambda$ $(\leq p_1 + p_2)$ relations is nonsingular because its determinant is a Vandermonde's determinant. Therefore these relations have a solution of $x_1 = x_2 = \cdots = x_{v_1+\cdots+v_{\lfloor (p_1+p_2)/2 \rfloor}} = 0$. This contradicts the assumption. Consequently this code satisfies the condition in Theorem 7.34. Q.E.D.

Example 7.6

Presented in Figure 7.26 is an example of a parity-check matrix of the code correcting triple m-spotty byte errors in two bytes, as the (157, 128) $[T_{2/8}EC]_2$ code, with parameters of $\mu = 3, p = 2, t = 2$ bits, $b = 8$ bits, and information-bit length $K = 128$. The maximum code length of the code has originally $N = 1,016$ bits (not lengthened). Here γ is a primitive element of $GF(2^7)$ defined by the primitive polynomial $\mathbf{g}(x) = x^7 + x + 1$. $\mathbf{H}'$ is an 8×8 identity matrix, and $\mathbf{H}''$ is the following 7×8 binary matrix whose column 7 vectors are linearly independent:

$$\mathbf{H}'' = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} = [\gamma^0 \quad \gamma^1 \quad \gamma^2 \quad \gamma^3 \quad \gamma^4 \quad \gamma^5 \quad \gamma^6 \quad \gamma^{121}].$$

This code requires $R = 29$ check bits, whereas the conventional triple m-spotty byte error correcting ($T_{2/8}EC$) code requires $R = 43$ bits.

The following theorem gives the lengthened code.

Theorem 7.38 *Let γ be a primitive element of $GF(2^{r'})$, where $r' \geq r$. The null space of*

$$\boldsymbol{H} = \left[\begin{array}{cccc|cc} \boldsymbol{H}' & \boldsymbol{H}' & \cdots & \boldsymbol{H}' & \boldsymbol{H}' & \boldsymbol{O} \\ \gamma^0\boldsymbol{H}'' & \gamma^1\boldsymbol{H}'' & \cdots & \gamma^{n-1}\boldsymbol{H}'' & \boldsymbol{O} & \boldsymbol{O} \\ \gamma^0\boldsymbol{H}'' & \gamma^2\boldsymbol{H}'' & \cdots & \gamma^{2(n-1)}\boldsymbol{H}'' & \boldsymbol{O} & \boldsymbol{O} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \gamma^0\boldsymbol{H}'' & \gamma^{(p_1+p_2-1)}\boldsymbol{H}'' & \cdots & \gamma^{(p_1+p_2-1)(n-1)}\boldsymbol{H}'' & \boldsymbol{O} & \boldsymbol{I}_{r'} \end{array}\right]$$

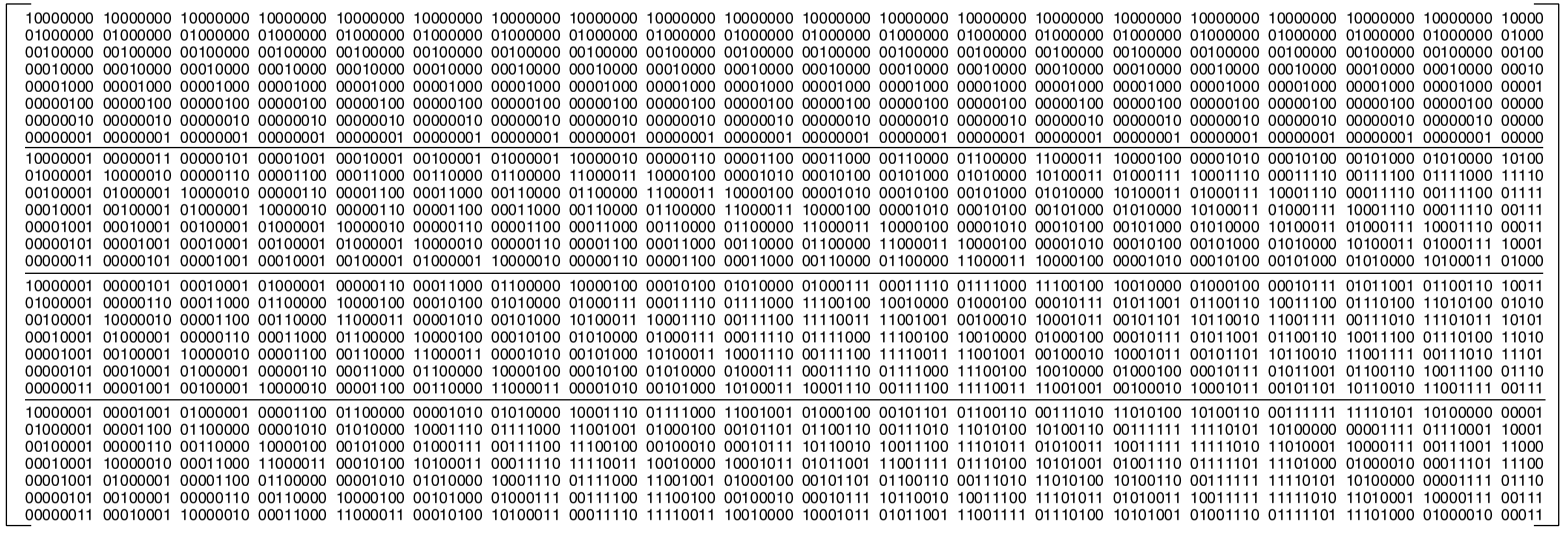
is a code $\boldsymbol{C}$ with check-bit length $R = q + (p_1 + p_2 - 1)r'$ and code length in bits $N = (n + 1) \cdot b + r'$, where $n = 2^{r'} - 1$, $\gamma^i\boldsymbol{H}'' = [\gamma^i\phi(h_0'') \quad \gamma^i\phi(h_1'') \quad \cdots \quad \gamma^i\phi(h_{b-1}'')]$ for $0 \leq i \leq n - 1$, $\phi : GF(2^r) \rightarrow GF(2^{r'})$ is a homomorphism of $GF(2^r)$ into $GF(2^{r'})$ under addition, and $\boldsymbol{I}_{r'}$ is an $r' \times r'$ identity matrix.

This theorem is easily proved, and therefore omitted.

Decoding We decode here the code in Theorem 7.37.

Let $\boldsymbol{C}'$ be a $[\mu_{t/b}EC]_p$ code. Also let c, v, and E be a codeword of $\boldsymbol{C}'$, a received word, and an error vector, respectively. The syndrome S is calculated by using the matrix $\mathbf{H}$ in Theorem 7.37 such that

$$\begin{aligned} S &= [S_0 \; S_1 \; S_2 \; \cdots \; S_{2p-1}] \\ &= v \cdot \mathbf{H}^T = (c + E) \cdot \mathbf{H}^T = E \cdot \mathbf{H}^T, \end{aligned}$$



10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000000	10000
01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000000	01000
00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100000	00100
00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010000	00010
00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001000	00001
00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000100	00000
00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000010	00000
00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000001	00000
10000001	00000011	00000101	00001001	00010001	00100001	01000001	10000010	00000110	00001100	00011000	00110000	01100000	11000011	10000100	00001010	00010100	00101000	01010000	10100			
01000001	10000010	00000110	00001100	00011000	00110000	01100000	11000011	10000100	00001010	00010100	00101000	01010000	10100011	01000111	10001110	00011110	00111100	01111000	11110			
00100001	01000001	10000010	00000110	00001100	00011000	00110000	01100000	11000011	10000100	00001010	00010100	00101000	01010000	10100011	01000111	10001110	00011110	00111100	01111			
00010001	00100001	01000001	10000010	00000110	00001100	00011000	00110000	01100000	11000011	10000100	00001010	00010100	00101000	01010000	10100011	01000111	10001110	00011110	00111			
00001001	00010001	00100001	01000001	10000010	00000110	00001100	00011000	00110000	01100000	11000011	10000100	00001010	00010100	00101000	01010000	10100011	01000111	10001110	00011			
00000101	00001001	00010001	00100001	01000001	10000010	00000110	00001100	00011000	00110000	01100000	11000011	10000100	00001010	00010100	00101000	01010000	10100011	01000111	10001			
00000011	00000101	00001001	00010001	00100001	01000001	10000010	00000110	00001100	00011000	00110000	01100000	11000011	10000100	00001010	00010100	00101000	01010000	10100011	01000			
10000001	00000101	00010001	01000001	00000110	00011000	01100000	10000100	00010100	01010000	01000111	00011110	01111000	11100100	10010000	01000100	00010111	01011001	01100110	10011			
01000001	00000110	00011000	01100000	10000100	00010100	01010000	01000111	00011110	01111000	11100100	10010000	01000100	00010111	01011001	01100110	10011100	01110100	11010100	01010			
00100001	10000010	00001100	00110000	11000011	00001010	00101000	10100011	10001110	00111100	11110011	11001001	00100010	10001011	00101101	10110010	11001111	00111010	11101011	10101			
00010001	01000001	00000110	00011000	01100000	10000100	00010100	01010000	01000111	00011110	01111000	11100100	10010000	01000100	00010111	01011001	01100110	10011100	01110100	11010			
00001001	00100001	10000010	00001100	00110000	11000011	00001010	00101000	10100011	10001110	00111100	11110011	11001001	00100010	10001011	00101101	10110010	11001111	00111010	11101			
00000101	00010001	01000001	00000110	00011000	01100000	10000100	00010100	01010000	01000111	00011110	01111000	11100100	10010000	01000100	00010111	01011001	01100110	10011100	01110			
00000011	00001001	00100001	10000010	00001100	00110000	11000011	00001010	00101000	10100011	10001110	00111100	11110011	11001001	00100010	10001011	00101101	10110010	11001111	00111			
10000001	00001001	01000001	00001100	01100000	00001010	01010000	10001110	01111000	11001001	01000100	00101101	01100110	00111010	11010100	10100110	00111111	11110101	10100000	00001			
01000001	00001100	01100000	00001010	01010000	10001110	01111000	11001001	01000100	00101101	01100110	00111010	11010100	10100110	00111111	11110101	10100000	00001111	01110001	10001			
00100001	00000110	00110000	10000100	00101000	01000111	00111100	11100100	00100010	00010111	10110010	10011100	11101011	01010011	10011111	11111010	11010001	10000111	00111001	11000			
00010001	10000010	00011000	11000011	00010100	10100011	00011110	11110011	10010000	10001011	01011001	11001111	01110100	10101001	01001110	01111101	11101000	01000010	00011101	11100			
00001001	01000001	00001100	01100000	00001010	01010000	10001110	01111000	11001001	01000100	00101101	01100110	00111010	11010100	10100110	00111111	11110101	10100000	00001111	01110			
00000101	00100001	00000110	00110000	10000100	00101000	01000111	00111100	11100100	00100010	00010111	10110010	10011100	11101011	01010011	10011111	11111010	11010001	10000111	00111			
00000011	00010001	10000010	00011000	11000011	00010100	10100011	00011110	11110011	10010000	10001011	01011001	11001111	01110100	10101001	01001110	01111101	11101000	01000010	00011			

where $S_0 \in GF(2^q)$ is a q-bit binary row vector and $S_j \in GF(2^r),\ j = 1,\ 2,\ \ldots,\ 2p-1$, is an r-bit binary row vector. If μ or fewer spotty byte errors $E_{i_1},\ E_{i_2},\ \ldots,\ E_{i_\rho}$, that satisfy $\sum_{x=1}^{\rho} w_{\mathrm{M}}(E_{i_x}) \le \mu$ have occurred in the i_1-th, i_2-th, ..., i_ρ-th byte, respectively, then the syndrome S is given by Eq. (7.34), where $\rho \le p \le \mu$:

$$S = \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ \vdots \\ S_{2p-1} \end{bmatrix}^T = \begin{bmatrix} E_1 \cdot \mathbf{H}'^T + E_2 \cdot \mathbf{H}'^T + \cdots + E_\rho \cdot \mathbf{H}'^T \\ \gamma^{i_1} E_1 \cdot \mathbf{H}''^T + \gamma^{i_2} E_2 \cdot \mathbf{H}''^T + \cdots + \gamma^{i_\rho} E_\rho \cdot \mathbf{H}''^T \\ \gamma^{2i_1} E_1 \cdot \mathbf{H}''^T + \gamma^{2i_2} E_2 \cdot \mathbf{H}''^T + \cdots + \gamma^{2i_\rho} E_\rho \cdot \mathbf{H}''^T \\ \vdots \\ \gamma^{(2p-1)i_1} E_1 \cdot \mathbf{H}''^T + \gamma^{(2p-1)i_2} E_2 \cdot \mathbf{H}''^T + \cdots + \gamma^{(2p-1)i_\rho} E_\rho \cdot \mathbf{H}''^T \end{bmatrix}^T, \quad (7.34)$$

$$S' = \begin{bmatrix} S'_0 \\ S_1 \\ S_2 \\ \vdots \\ S_{2p-1} \end{bmatrix}^T = \begin{bmatrix} \varepsilon_{i_1} + \varepsilon_{i_2} + \cdots + \varepsilon_{i_\rho} \\ \gamma^{i_1}\varepsilon_{i_1} + \gamma^{i_2}\varepsilon_{i_2} + \cdots + \gamma^{i_\rho}\varepsilon_{i_\rho} \\ \gamma^{2i_1}\varepsilon_{i_1} + \gamma^{2i_2}\varepsilon_{i_2} + \cdots + \gamma^{2i_\rho}\varepsilon_{i_\rho} \\ \vdots \\ \gamma^{(2p-1)i_1}\varepsilon_{i_1} + \gamma^{(2p-1)i_2}\varepsilon_{i_2} + \cdots + \gamma^{(2p-1)i_\rho}\varepsilon_{i_\rho} \end{bmatrix}^T. \quad (7.35)$$

Next, let $\sum_{x=1}^{\rho} E_{i_x}$ be expressed as E^*. The relation $S_0 = E^* \cdot \mathbf{H}'^T$ leads to E^*, since $\mathbf{H}'$ is a parity-check matrix of μ t/b-error correcting code. Multiplying this by $\mathbf{H}''^T$ from the right gives $E^* \cdot \mathbf{H}''^T$. Let $E_{i_1} \cdot \mathbf{H}''^T,\ E_{i_2} \cdot \mathbf{H}''^T,\ \ldots,\ E_{i_\rho} \cdot \mathbf{H}''^T \in GF(2^r)$ be $\varepsilon_{i_1},\ \varepsilon_{i_2},\ \ldots,\ \varepsilon_{i_\rho}$, respectively. Then the syndrome S' is as given by Eq. (7.35). This syndrome is identical to that of the RS code with distance $2p+1$ over $GF(2^r)$. The error patterns over $GF(2^r)$ and error locations are determined by using the decoding algorithms of the RS code such as the Berlekamp-Massey algorithm and the Euclidean algorithm.

In the final step of the decoding, the error patterns $\widehat{E_{i_x}} \in GF(2^b)$, where $x = 1,\ 2,\ \ldots,\ \rho$, are transformed from the corresponding error patterns $\varepsilon_{i_x} \in GF(2^r)$ according to one-to-one mapping from ε_{i_x} to $\widehat{E_{i_x}}$ for $x = 1,\ 2,\ \ldots,\ \rho$. This mapping is implemented by the table. Here, at most, one of the $\widehat{E_{i_x}}$'s may be miscorrected, that is, $\widehat{E_{i_x}} \neq E_{i_x}$. The following relation determines whether or not $\widehat{E_{i_x}}$ is identical to E_{i_x}. That is, if $\widehat{E_{i_x}}$ satisfies the relation (7.36), then $\widehat{E_{i_x}}$ is equal to E_{i_x}. Otherwise, $\widehat{E_{i_x}} \neq E_{i_x}$.

$$w_{\mathrm{M}}(\widehat{E_{i_x}} + E^*) \le \mu - w_{\mathrm{M}}(\widehat{E_{i_x}}). \quad (7.36)$$

This relation can be proved based on the fact that $w_{\mathrm{M}}(E)$ satisfies the triangle inequality [IMAI79].

In sum, from the discussion above, the decoding is performed according to the following algorithm:

Step 1. The first element S_0 in S is transformed to $S'_0 \in GF(2^r)$ by the operation $S'_0 = E^* \cdot \mathbf{H}''^T$.

Step 2. Error locations $i_1,\ i_2,\ \ldots,\ i_\rho$, and error patterns $\varepsilon_{i_1},\ \varepsilon_{i_2},\ \ldots,\ \varepsilon_{i_\rho}$, are obtained from the syndrome S' by the decoding algorithm of the RS codes over $GF(2^r)$.

Step 3. The error pattern $\widehat{E_{i_x}}$ is obtained from ε_{i_x} according to the mapping table.

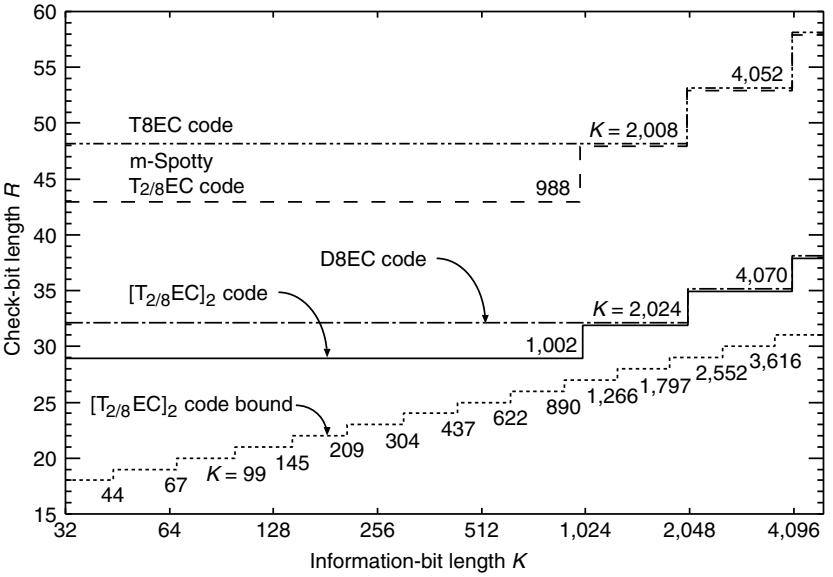


Figure 7.27 *Check-bit lengths compared with information-bit lengths of the $[T_{2/8}EC]_2$ codes, along with the $T_{2/8}EC$ codes, D8EC codes, and T8EC codes.* Source: *[SUZU05b]. © 2005 IEEE.*

Step 4. The error patterns $\widehat{E_{i_x}}$, $x = 1, 2, \ldots, \rho$, obtained in the previous step are checked whether or not they satisfy the relation (7.36). If satisfied, then $\widehat{E_{i_x}} = E_{i_x}$.

Step 5. If $\widehat{E_{i_y}}$ does not satisfy the relation (7.36) or cannot be transformed from ε_{i_x} in the mapping table, the error pattern E_{i_y} is recovered from the other error patterns obtained in step 4, meaning $E_{i_y} = e^* + E_{i_1} + \cdots + E_{i_{y-1}} + E_{i_{y+1}} + \cdots + E_{i_\rho}$.

Evaluation Figure 7.27 shows the relation between the information-bit length and the check-bit length of the lengthened $[T_{2/8}EC]_2$ code and its bound where $\mu = 3$, $p = 2$, t = 2 bits, and b = 8 bits, along with the conventional triple m-spotty byte error correcting ($T_{2/8}EC$) codes, the double-byte error correcting (D8EC) codes, and the triple-byte error correcting (T8EC) codes. In this case the bound is the one given by Theorems 7.35 and 7.36.

EXERCISES

7.1 Prove that for any $(N, N-R)$ $S_{t/b}EC$ codes the following inequality holds:

$$2^R \geq \frac{N}{b} \cdot \sum_{i=1}^{t} \binom{b}{i} + 1.$$

In your proof show that the equality holds when $t = b$ and R is an integer multiple of b. Also show how perfect SbEC codes can be constructed by using Theorem 7.2.

7.2 Investigate: if $\mathbf{H}''$ denotes a parity-check matrix of a perfect single symbol error correcting code over $GF(2^b)$, and $\mathbf{H}' = \mathbf{I}_b$, the binary code defined as the null space of the tensor product of $\mathbf{H}''$ and $\mathbf{H}'$ in that order (i.e., $\mathbf{H}'' \otimes \mathbf{H}'$) is a perfect single b-bit byte error correcting code. Provide simple examples or counter-examples to validate your investigation.

7.3 Let $\mathbf{H}' = [\mathbf{I}_{2t} \mid h]$, where h denotes the all-1 binary column vector of $GF(2^{2t})$. Show that any $2t$ or fewer binary columns of $\mathbf{H}'$ are linearly independent, and convince yourself that $\mathbf{H}'$ can be regarded as a $(2t+1, 1)$ perfect t-error correcting binary code. Design a perfect $S_{t/2t+1}EC$ code by taking tensor product of $\mathbf{H}'$.

7.4 Designing an $S_{2/5}EC$ code: Consider the following $\mathbf{H}'$ and $\mathbf{H}''$ matrices:

$$\mathbf{H}' = \begin{bmatrix} 10001 \\ 01001 \\ 00101 \\ 00011 \end{bmatrix} = \begin{bmatrix} | & | & | & | & | \\ \gamma^0 & \gamma^1 & \gamma^2 & \gamma^3 & (\sum_{i=0}^{3} \gamma^i) \\ | & | & | & | & | \end{bmatrix},$$

$$\mathbf{H}'' = \left[\begin{array}{ccc|cc} \gamma^0 & \gamma^0 & \gamma^0 \cdots \gamma^0 & \gamma^0 & 0 \\ \gamma^0 & \gamma^1 & \gamma^2 \cdots \gamma^{14} & 0 & \gamma^0 \end{array}\right],$$

where γ is a primitive element of $GF(2^4)$. Write down the $\mathbf{H}$ matrix defined by $\mathbf{H}'' \otimes \mathbf{H}'$. Why this $\mathbf{H}$ matrix represents a perfect $S_{2/5}EC$ code?

7.5 Design a perfect $S_{3/7}EC$ code. Clearly illustrate your design procedure. What is the code length of your code? Explain why your code is perfect?

7.6 Figure 7.2 shows a $(132, 121)$ $S_{3/8}EC$ code. Using the illustrations provided in Subsection 7.2.3, show how this code was generated?

7.7 Decoding the $(132, 121)$ $S_{3/8}EC$ code: As shown below, let S_0 and S_1 be the first 7-bit and the second 4-bit vectors of the syndrome, respectively:

$$\left[\begin{array}{cccc|c} \mathbf{H}' & \mathbf{H}' & \cdots & \mathbf{H}' & \mathbf{O}_{7\times 4} \\ \gamma^0\mathbf{H}'' & \gamma^1\mathbf{H}'' & \cdots & \gamma^{15}\mathbf{H}'' & \mathbf{I}_4 \end{array}\right] \begin{array}{l} \rightarrow S_0 \\ \rightarrow S_1 \end{array}$$

Explain the following four steps, and convince yourself that these steps basically provide the decoding algorithm for the $S_{3/8}EC$ code.

Step 1. *If $S_0 = 0$ and $S_1 = 0$, there are no errors.*

Step 2. *If $S_0 = 0$ and $S_1 \neq 0$, the last byte is corrupted by error pattern S_1.*

Step 3. *If $S_0 \neq 0$ and $S_1 = 0$, the second last byte is corrupted by an 8-bit error pattern.*

Step 4. *If $S_0 \neq 0$ and $S_1 \neq 0$, one of the first 15 bytes is corrupted by an 8-bit error pattern.*

In steps 3 and 4 we need to determine the 8-bit error pattern. The $S_{3/8}EC$ code uses the following $\mathbf{H}'$ matrix:

$$\mathbf{H}' = \begin{bmatrix} 10000001 \\ 01000001 \\ 00100001 \\ 00010001 \\ 00001001 \\ 00000101 \\ 00000011 \end{bmatrix}.$$

Explain how the 8-bit error pattern can be obtained from S_0. (Hint: How many one's would be there in S_0 if it is a 3/8-error.) Explain how to determine the error location in step 4. Do you need to know the error pattern in advance in order to determine the error location?

7.8 The $(272, 256)$ $S_{3/16}EC$ code shown in Figure 7.1 is a systematic code obtained by performing row operations on the original code. Can this code be decoded by the method mentioned in Exercise 7.7? If not, can you think of another method for decoding this code? (Hint: Read Chapter 8 on parallel decoding for burst / byte error control codes.)

7.9 A memory system with a 256-bit information length uses 16-bit RAM chips. It is believed that the random double-bit errors occurring within a single RAM chip are the most significant errors. Design an efficient $S_{2/16}EC$ code for this memory system. How many check bits are required by your code? Estimate the decoding hardware complexity of your code.

7.10 Is it possible to construct a $(74, 64)$ $S_{2/8}EC$ code with 10 check bits? If it is possible, design this code. (Hint: See Figure 7.4.)

7.11 A memory system with a 64-bit information length uses 8-bit RAM chips. Design a $(74, 64)$ $S_{2/8}EC$ code for this memory system. Estimate the decoding hardware complexity of your code.

7.12 Let γ be a primitive element of $GF(2^6)$ determined by the primitive polynomial $\mathbf{g}(x) = x^6 + x + 1$. Define $\mathbf{H}_i = [\gamma^i \ \gamma^{i+21} \ \gamma^{i+42}]$, where $0 \leq i \leq 20$. Explain why the code represented by the $\mathbf{H}$ matrix below is a perfect $S_{2/3}EC$-S3ED code.

$$\mathbf{H} = \left[\begin{array}{c|c|c|c|c} 100 & 100 & 100 & \cdots & 100 \\ \hline \mathbf{H}_0 & \mathbf{H}_1 & \mathbf{H}_2 & \cdots & \mathbf{H}_{20} \end{array}\right]$$

7.13 Write down the $GF(2^3)$ subfield elements of the field $GF(2^9)$. Using these subfield elements, demonstrate how a perfect $S_{3/4}EC$-S4ED code can be constructed.

7.14 Prove that inequalities (7.5) and (7.6) hold for an $S_{t/b}EC$-SbED code.

7.15 Explain what type of $\mathbf{H}''$ matrix you would use in Theorem 7.4 to obtain the Hong-Patel type of codes presented in Theorem 7.3.

7.16 Design an $S_{4/8}EC\text{-}S8ED$ code with an information length of 256 bits. How many check bits are required by your code?

7.17 A memory system with a 64-bit information length uses 8-bit RAM chips. The RAM chips have the multi-bank architecture shown in Figure 7.10. Design an $S_{2/8}EC\text{-}S4EC\text{-}S8ED$ code for this memory system. Does your $S_{2/8}EC\text{-}S4EC\text{-}S8ED$ code correct all or some single 3/8-errors?

7.18 Prove Theorem 7.11.

7.19 Prove Theorem 7.12.

7.20 Find the bound on code length of an $(N, N-R)$ $S_{t/b}EC\text{-}Sb EC\text{-}SB ED$ code.

7.21 Explain that the code presented in Figure 7.12 works as an $S_{3/8}EC\text{-}D_{3/8}ED$ code.

7.22 Design the $S_{t/b}EC\text{-}D_{t/b}ED$ codes with code parameters $b = 6$ bits and $t = 2, 3, 4, 5$ bits.

7.23 Show that the code in Theorem 7.11 can also detect double spotty byte errors in a byte as well as detect double spotty byte errors in distinct two bytes.

7.24 Find the decoding method of the $S_{t/b}EC\text{-}D_{t/b}ED$ codes, especially how to detect double t/b-errors.

7.25 Find the matrix $\mathbf{H}'$ in s-spotty byte error control codes with $b = 6$ and $t = 2$.

7.26 Design a binary $(504,\ 480)$ s-spotty $D_{2/8}EC$ code by using the primitive element γ over $GF(2^6)$ defined by the primitive polynomial $\boldsymbol{g}(x) = x^6 + x^5 + 1$ and by

$$\mathbf{H}' = \begin{bmatrix} 1 & & & & 1 & 0 \\ & 1 & & 0 & 1 & 0 \\ & & 1 & & 1 & 1 \\ & & & 1 & 1 & 1 \\ & & 1 & & 0 & 1 \\ & 0 & & & 1 & 0 \\ & & & 1 & 0 & 1 \end{bmatrix}_{6\times 8} = \begin{bmatrix} \gamma^0 & \gamma^1 & \gamma^2 & \gamma^3 & \gamma^4 & \gamma^5 & \gamma^{48} & \gamma^{50} \end{bmatrix}.$$

If the errors occurred in the first byte and the 60th byte, written as $E_1 = (0\ 0\ 1\ 0\ 0\ 0\ 1\ 0)$ and $E_{60} = (0\ 0\ 0\ 1\ 0\ 0\ 1\ 0)$, respectively, indicate the decoding procedure to obtain the error positions and the error values by using the Berlekamp-Massey algorithm.

7.27 Design the lengthened s-spotty $D_{t/b}EC\text{-}Sb ED$ code. Find the code in binary form with $t = 2$ bits and $b = 5$ bits.

7.28 Show that $\mathbf{H}$ matrices in Theorems 7.24 and 7.25 having $\mathbf{H}' = \mathbf{I}$, i.e., $q \times q$ identity matrix, are the distance-d m-spotty byte error control codes with $Sb ED$ capability.

7.29 Design the complex m-spotty $S_{2/8}EC\text{-}(S_{2/8}+S)ED$ code with information-bit length $K = 64$, and 128.

7.30 Design the complex m-spotty $(S_{2/8}+S)EC$ code with $K = 64$ and 128 bits.

7.31 Let's consider the single t/b-error correcting and double-bit error detecting ($S_{t/b}$EC-DED) code. For the given $\mathbf{H}$ matrix shown below, answer the following questions.

$$\mathbf{H} = \left[\begin{array}{cccccc:ccc} \mathbf{H}' & \mathbf{H}' & \mathbf{H}' & \cdots & \mathbf{H}' & \mathbf{H}' & \mathbf{O} & \mathbf{O} \\ \gamma^0\mathbf{H}'' & \gamma^1\mathbf{H}'' & \gamma^2\mathbf{H}'' & \cdots & \gamma^{n'''-1}\mathbf{H}'' & \mathbf{O} & \mathbf{I}_{r''} & \mathbf{O} \\ \delta^0\mathbf{1} & \delta^1\mathbf{1} & \delta^2\mathbf{1} & \cdots & \delta^{n'''-1}\mathbf{1} & \mathbf{O} & \mathbf{O} & \mathbf{I}_{r'''} \end{array}\right] \begin{matrix} \updownarrow r' \\ \updownarrow r'' \\ \updownarrow r''' \end{matrix},$$

where $t \geq 2$,

$\mathbf{H}' = [h'_0\, h'_1 \cdots h'_{b-1}]_{r' \times b}$, $h'_i \in GF(2^{r'})$, $0 \leq i \leq b-1$, whose $\min(2t,\ b)$ column vectors are linearly independent,

$\mathbf{H}'' = [h''_0\, h''_1 \cdots h''_{b-1}]_{r'' \times b}$, $h''_i \in GF(2^{r''})$, $0 \leq i \leq b-1$, whose t column vectors are linearly independent,

γ : primitive element of $GF(2^{r''})$,

$\mathbf{1} = [1\ 1 \cdots 1]_{1 \times b}$,

δ : primitive element of $GF(2^{r'''})$,

$\mathbf{I}_{r''}$: $r'' \times r''$ identity matrix,

$\mathbf{I}_{r'''}$: $r''' \times r'''$ identity matrix, and

the code length in bits $N = b(n''' + 1) + r'' + r'''$,

$$n'' = 2^{r''} - 1 \geq n''' = 2^{r'''} - 1.$$

(a) Show that the null space of the matrix $\mathbf{H}$ above is an $S_{t/b}$EC-DED code.

(b) Design the (21, 10) $S_{2/8}$EC-DED code with parameters of $b = 8$, $t = 2$ and $r''' = 1$, and the (38, 26) $S_{2/8}$EC-DED code with $b = 8$, $t = 2$, and $r''' = 2$.

(c) Verify that the code indicated here has smaller check-bit length by 1 to 3 bits for information-bit length $K \leq 58$, compared to the complex m-spotty $S_{t/b}$EC-$D_{t'/b}$ED code with $t' = 1$ for $b = 8$ bits and $t = 2$ bits.

7.32 Design the $[T_{3/8}EC]_2$ code with $K = 128$ bits.

7.33 Design the $D_{2/8}$EC-$[T_{2/8}ED]_2$ code with $K = 128$ bits.

7.34 Prove Theorems 7.18 through 7.29

7.35 Design the complex s-spotty byte error control codes of the $S_{t/b}$EC-($S_{t/b}$+$S_{t'/b}$)ED code, and the ($S_{t/b}$+$S_{t'/b}$)EC code.

REFERENCES

[BLAH83] R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley (1983).

[DAVY89] A. A. Davydov and A. Yu. Drozhzhina-Labinskaya, "Length 4 byte error and double independent error correction by BCH code in semiconductor memories," *Autom. Remote Contr.*, 50 (November 1989): 1570–1579.

[FUJI04] E. Fujiwara, K. Namba, and M. Kitakami, ''Parallel Decoding for Burst Error Control Codes,'' *Electron. Commun. Japan*, 87 (January 2004): 38–48.

[HONG72] S.J. Hong and A.M. Patel, ''A General Class of Maximal Codes for Computer Applications,'' *IEEE Trans. Comput.*, 21 (December 1972): 1322–1331.

[IMAI79] H. Imai and H. Fujiya, ''A Construction Method for Simply-Decodable Error-Correcting Codes'' (in Japanese), *Trans. IECE Japan*, 62-A (May 1979): 271–277.

[KANE05] H. Kaneko, ''Error Control Coding for Semiconductor Memory Systems in the Space Radiation Environment,'' *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems,* (October 2005): 93–101.

[KASH04] T. Kashiyama and E. Fujiwara, ''A General Class of Byte Error Control Codes for S-Spotty Byte Errors,'' *Proc. IEEE Int. Symp. on Information Theory Applications* (October 2004): 1297–1302.

[MASS96] L. W. Massengil. ''Cosmic and Terrestrial Single Event Radian Effects in Dynamic Random Access Memories,'' *IEEE Trans. Nucl. Sci.*, 43 (April 1996): 576–593.

[NUMA89] K. Numata, Y. Oowaki, et al, ''New Nibbled-Page Architecture for High-Density DRAM's,'' *IEEE J. Solid-State Circ.*, 24 (August 1989): 900–904.

[OGOR96] T. J. O'Gorman et al, ''Field Testing for Cosmic Ray Soft Errors in Semiconductor Memories,'' *IBM J. Res. Dev.*, 40 (January 1996): 41–50.

[SAEK96] T. Saeki, Y. Nakaoka et al, ''A 2.5-ns Clock Access, 256MHz, 256 Mb SDRAM with Synchronous Mirror Delay,'' *IEEE J. Solid-State Circ.*, 31 (November 1996): 1656–1668.

[SHU83] Shu Lin, and D. J. Costello, Jr., *Error Control Coding*, Prentice Hall (1983).

[SRIN96] G. R. Srinivasan, ''Modeling the Cosmic Ray Induced Soft Error Rate in Integrated Circuits: An Overview,'' *IBM J. Res. Dev.*, 40 (January 1996): 77–89.

[SUGI93] T. Sugibayashi, T. Takeshima, et al., ''A 30-ns 256 Mb DRAM With Multidivided Array Structure,'' *IEEE J. Solid-State Circ.*, 28 (November 1993): 1092–1098.

[SUNA95] T. Sunaga, K. Hosokawa, et al., ''A Full Bit Perfect Architecture for Synchronous DRAM's,'' *IEEE J. Solid-State Circ.*, 30 (November 1995): 998–1005.

[SUZU04] K. Suzuki, T. Kashiyama, and E. Fujiwara, ''A General Class of M-Spotty Byte Error Control Codes,'' *Proc. 4th Asia–Europe Workshop on Information Theory* (October 2004): 24–26.

[SUZU05a] K. Suzuki, T. Kashiyama, and E. Fujiwara, ''Complex M-spotty Byte Error Control Codes,'' *Proc. IEEE Information Theory Workshop* (September 2005): 211–215.

[SUZU05b] K. Suzuki, M. Shimizu, T. Kashiyama, and E. Fujiwara, ''A Class of Error Control Codes for M-Spotty Byte Errors Occurred in a Limited Number of Bytes,'' *Proc. IEEE Int. Symp. on Information Theory* (September 2005): 2109–2113.

[TANI92] K. Tanigawa, Y. Kinoshita, and M. Morii, ''Low-Density Byte Error-Control Codes'' (in Japanese), *IEICE Technical Report*, IT 92-18 (1992): 41–46.

[UMAN02a] G. Umanesan and E. Fujiwara, ''Random Double Bit Error Correcting–Single *b*-bit Byte Error Correcting (DEC-S*b*EC) Codes for Memory Systems,'' *IEICE Trans. Fundamentals*, E85-A (January 2002): 273–276.

[UMAN02b] G. Umanesan and E. Fujiwara, ''Single Byte Error Correcting Codes with Double Bit Within a Block Error Correcting Capability for Memory Systems,'' *IEICE Trans. Fundamentals*, E85-A (February 2002): 513–517.

[UMAN03a] G. Umanesan and E. Fujiwara, ''A Class of Codes for Correcting Single Spotty Byte Errors,'' *IEICE Trans. Fundamentals*, E86-A (March 2003): 704–714.

[UMAN03b] G. Umanesan and E. Fujiwara, ''A Class of Random Multiple Bits in a Byte Error Correcting and Single Byte Error Detecting ($S_{t/b}$EC-S_bED) Codes,'' *IEEE Trans. Comput.*, 52 (July 2003): 835–847.

[VAID92] N. H. Vaidya and D. K. Pradhan. "A New Class of Bit and Byte Error Control Codes," *IEEE Trans. Info. Theory*, 38 (September 1992): 1617–1623.

[WICK94] S. B. Wicker and V. K. Bhargava, *Reed-Solomon Codes and Their Applications*, IEEE Press (1994).

[WATA96] Y. Watanabe, H. Wong et al, "A 286 mm^2 256 Mb DRAM with $\times 32$ Both-Ends DQ," *IEEE J. Solid-State Circ.*, 31 (April 1996): 567–574.

[WOLF65] J. K. Wolf, "On Codes Derivable from the Tensor Product of Check Matrices," *IEEE Trans. Info. Theory*, 11 (April 1965): 281–284.

[ZIEG96] J. F. Ziegler, et al., "IBM Experiments in Soft Fails in Computer Electronics (1978–1994)," *IBM J. Res. Dev.*, 40 (January 1996): 3–18.

CONTENTS

8

Parallel Decoding Burst / Byte Error Control Codes

Optical and magnetic recording systems, and communication systems usually read / write (or receive / transmit) the data serially bit by bit. Therefore sequential decoding methods implemented by linear feedback shift registers (LFSRs) are popularly used for error correction and detection [MEGG61, CHIE69]. It is known that two-dimensional burst errors occur in ultra–large capacity holographic memories [NISH97] in which a large amount of data are sometimes readout at once. Therefore parallel decoding implemented only by combinational logic is required for high-speed burst error correction. A parallel encoding / decoding can be easily converted to a serial encoding / decoding by using serial to / from parallel transformation of the data.

An interleaving method for bit or byte error control codes has been popularly used for burst error correction and detection [PETE72] because parallel decoding of the interleaved codes can be easily implemented. However, longer burst error correction requires interleaving with higher degree, subsequently increasing the number of check bits to unacceptable levels for practical applications. On the other hand, Fire codes are well known as efficient burst error control codes [PETE72, ELSP62, KASA62a, KASA62b]. The Fire code has been discussed in Subsection 2.3.7.

The parallel decoding method we deal with here is applicable to any linear burst error control code, including the Fire code. As we explain below, this decoding treats byte errors as a special case of burst errors, so it requires less hardware than the existing methods. The parallel decoding method can therefore be applied to any type of linear burst / byte / bit error correcting code. It is very general in the sense that this decoding not only completely includes the conventional parallel decoding of the linear bit / byte error correcting codes but also applies to the multiple burst / byte error correcting codes [FUJI02].

Code Design for Dependable Systems: Theory and Practical Applications, by Eiji Fujiwara

The last section of this chapter addresses the important problem of *glitches*, meaning the logical noises that occur in parallel decoding circuits. Parallel decoding circuits depend heavily on large exclusive-OR (XOR) tree circuits that are well known to produce glitches readily. This section clarifies why glitches are generated, how they are propagated and accumulated in the circuits, and how to reduce these undesirable effects.

8.1 PARALLEL DECODING BURST ERROR CONTROL CODES

8.1.1 Error Pattern Generation by Inverse Matrices

Let $\mathbf{H}$ be an $R \times N$ parity-check matrix of a linear burst error correcting code capable of correcting single l-bit burst errors. Let E be the error vector denoting an l-bit burst error pattern starting from the i-th bit $(0 \leq i < N)$ of the N-bit word. Let $\mathbf{H}_i$ be the $R \times L$ submatrix of length $L(\geq l)$ starting from the i-th column of the matrix $\mathbf{H}$. Figure 8.1 shows an error E with length L in an error vector E^* that includes the l-bit burst error. It also shows the corresponding submatrix $\mathbf{H}_i$. The syndrome S in this case can be calculated as

$$\begin{aligned} S &= E^* \cdot \mathbf{H}^T \\ &= E \cdot \mathbf{H}_i^T. \end{aligned} \tag{8.1}$$

Let the column vectors of the submatrix $\mathbf{H}_i$ be linearly independent. Let $\mathbf{A}_i$ be an $R \times R$ nonsingular matrix obtained by appending an $R \times (R - L)$ matrix $\mathbf{B}_i$ to the submatrix $\mathbf{H}_i$ $\left(\text{i.e., } \mathbf{A}_i = [\mathbf{H}_i \mathbf{B}_i]\right)$. Again, let $\mathbf{A}_i^{-1} = \begin{bmatrix} \mathbf{H}_i^\dagger \\ \mathbf{B}_i^\dagger \end{bmatrix}$ be the inverse matrix of $\mathbf{A}_i$. Here $\mathbf{H}_i^\dagger$ and $\mathbf{B}_i^\dagger$ are $L \times R$ and $(R - L) \times R$ matrices, respectively. The error E can be obtained from the matrices $\mathbf{H}_i^\dagger$ and $\mathbf{B}_i^\dagger$ and the syndrome S as shown by the following theorem.

Theorem 8.1 *If there exists a burst error E starting from the i-th bit of the received word, the following holds for syndrome S:*

$$\begin{aligned} S \cdot \boldsymbol{H}_i^{\dagger T} &= E, \\ S \cdot \boldsymbol{B}_i^{\dagger T} &= \boldsymbol{0}. \end{aligned} \tag{8.2}$$

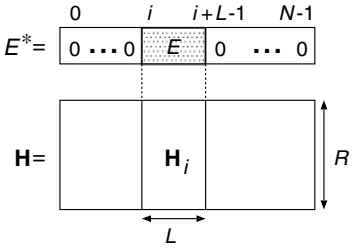


Figure 8.1 *Error E in an error vector E^* and the corresponding submatrix $\boldsymbol{H}_i$ in a parity-check matrix $\boldsymbol{H}$.* Source: *[FUJI02]. © 2002 IEICE Japan.*

Proof Since $\mathbf{A}_i^{-1}$ is the inverse matrix of $\mathbf{A}_i$, we have

$$\mathbf{A}_i^{-1} \cdot \mathbf{A}_i = \begin{bmatrix} \mathbf{H}_i^{\dagger} \\ \mathbf{B}_i^{\dagger} \end{bmatrix} \cdot [\, \mathbf{H}_i \quad \mathbf{B}_i \,] = \begin{bmatrix} \mathbf{H}_i^{\dagger} \cdot \mathbf{H}_i & \mathbf{H}_i^{\dagger} \cdot \mathbf{B}_i \\ \mathbf{B}_i^{\dagger} \cdot \mathbf{H}_i & \mathbf{B}_i^{\dagger} \cdot \mathbf{B}_i \end{bmatrix} = \boldsymbol{I}.$$

Subsequently the following holds:

$$\begin{aligned} \mathbf{H}_i^{\dagger} \cdot \mathbf{H}_i &= \mathbf{I}_{L\times L}, \\ \mathbf{B}_i^{\dagger} \cdot \mathbf{H}_i &= \mathbf{O}_{(R-L)\times L}, \\ \mathbf{H}_i^{\dagger} \cdot \mathbf{B}_i &= \mathbf{O}_{L\times(R-L)}, \\ \mathbf{B}_i^{\dagger} \cdot \mathbf{B}_i &= \mathbf{I}_{(R-L)\times(R-L)}. \end{aligned}$$

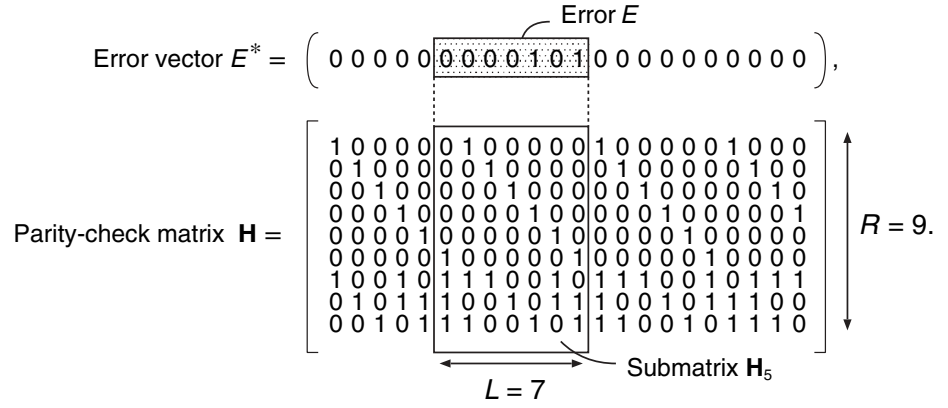
Therefore the following two equations hold for the syndrome given by Eq. (8.1):

$$\begin{aligned} S \cdot \mathbf{H}_i^{\dagger T} &= (E \cdot \mathbf{H}_i{}^T) \cdot \mathbf{H}_i^{\dagger T} = E \cdot (\mathbf{H}_i^{\dagger} \cdot \mathbf{H}_i)^T = E \cdot \mathbf{I} = E, \\ S \cdot \mathbf{B}_i^{\dagger T} &= (E \cdot \mathbf{H}_i{}^T) \cdot \mathbf{B}_i^{\dagger T} = E \cdot (\mathbf{B}_i^{\dagger} \cdot \mathbf{H}_i)^T = \mathbf{0}. \end{aligned}$$

Q.E.D.

Example 8.1 [FUJI02]

Consider the error pattern generation of the (22, 13) 3-bit burst error correcting Fire code shown below. Note that the 9×22 parity-check matrix $\mathbf{H}$ includes a 9×7 submatrix $\mathbf{H}_5$ representing binary columns starting from $i = 5$. The error vector E represents a 3-bit burst error starting from the 9-th bit of the word.



The syndrome is therefore written as follows:

$$S = E \cdot \mathbf{H}_5^T = (000101010).$$

An example of $\mathbf{B}_5$ is given below. Matrix $\mathbf{B}_5$ can be appended to $\mathbf{H}_5$ to obtain a 9×9 nonsingular matrix $\mathbf{A}_5$.

$$\mathbf{B}_5 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}, \qquad R - L = 2$$

$$\text{Nonsingular matrix } \mathbf{A}_5 = \left[\mathbf{H}_5 \,\middle|\, \mathbf{B}_5\right] = \left[\begin{array}{ccccccc|cc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right], \quad R = 9.$$

$R = 9$

$\mathbf{H}_5^{\dagger}$ and $\mathbf{B}_5^{\dagger}$ are then obtained as follows:

$$\text{Inverse matrix } \mathbf{A}_5^{-1} = \left[\begin{array}{c} \mathbf{H}_5^{\dagger} \\ \hline \mathbf{B}_5^{\dagger} \end{array}\right] = \left[\begin{array}{ccccccccc} 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ \hline 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \end{array}\right]$$

$\mathbf{H}_5^{\dagger}$: $L = 7$; $R - L = 2$; $R = 9$; $\mathbf{B}_5^{\dagger}$

Finally we have

$$S \cdot \mathbf{H}_5^{\dagger T} = (0000101) = E,$$
$$S \cdot \mathbf{B}_5^{\dagger T} = (00).$$

8.1.2 Frames for Burst Error Location

We will discuss the decoding of l-bit burst error correcting and $d(\geq l)$-bit burst error detecting codes. For burst error correction, the burst error pattern and the burst error location can be extracted from the syndrome. To calculate the burst error location, we then consider *frames* of length $L(\geq l)$ that completely include any l-bit burst error occurring in any position in the received word. As shown in Figure 8.2, the codeword is divided by a number of frames where adjacent frames overlap by z bits. The first

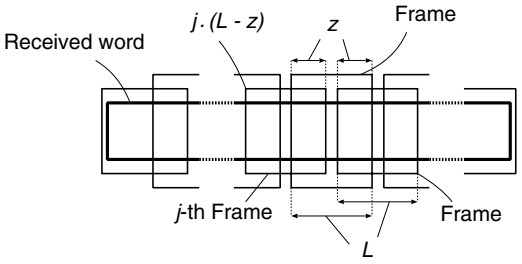


Figure 8.2 *Frame with length L bits overlapping at adjacent frames by z bits.* Source: *[FUJI02]. © 2002 IEICE Japan.*

and the last frames overlap only at one side. The length of the last frame is less than or equal to L bits. For a received word of length N, the total number of frames can then be given by

$$m = \left\lceil \frac{N - z}{L - z} \right\rceil, \tag{8.3}$$

where $\lceil x \rceil$ represents the smallest integer greater than or equal to x. The starting position of the j-th $(0 \leq j < m)$ frame is given by $j \cdot (L - z)$.

Lemma 8.1 *Assume that the error pattern generation method described in the previous subsection is used for generating l-bit burst error patterns in a linear l-bit burst error correcting and d-bit burst error detecting code. Then the frame length L satisfies the following inequality:*

$$l \leq L \leq l + d.$$

Proof The rank of submatrix $\mathbf{H}_i$ should be at least l, and therefore $l \leq L$. Clearly, any linear combination of consecutive $l + d$ or lesser columns of the parity-check matrix of the code above is nonzero. Hence L can have a maximum value of $l + d$, meaning $L \leq l + d$. This proves that $l \leq L \leq l + d$, as required. Q.E.D.

Lemma 8.2 *If $l \leq L$ and $z \geq l - 1$, any l-bit burst error is completely included in at least one of the frames.*

This lemma can be easily proved, and therefore the proof is omitted.

When frame length L is given by $l \leq L \leq l + d$ and length of overlap z is greater than or equal to $l - 1$, we have the following two theorems obtained by rewriting Theorem 8.1.

Theorem 8.2 *When a burst error E of length l or smaller is completely included in the j-th frame, the following holds:*

$$S \cdot \boldsymbol{H}^{\dagger\, T}_{j\cdot(L-z)} = E \quad \textit{and} \quad S \cdot \boldsymbol{B}^{\dagger\, T}_{j\cdot(L-z)} = \boldsymbol{0}.$$

Theorem 8.3 *When a burst error E'_d of length greater than l and smaller than or equal to d is completely included in the j-th frame, the following holds:*

$$S \cdot \boldsymbol{H}^{\dagger\,T}_{j\cdot(L-z)} = E'_d \quad \textit{and} \quad S \cdot \boldsymbol{B}^{\dagger\,T}_{j\cdot(L-z)} = \boldsymbol{0}.$$

On the other hand, when E is not included or partially included in the j-th frame, the following theorem holds.

Theorem 8.4 *When a burst error of length d bits or less is not included or partially included in the j-th frame, then the following equation holds:*

$$S \cdot \boldsymbol{H}^{\dagger\,T}_{j\cdot(L-z)} = E_d, \ \textit{or}\ S \cdot \boldsymbol{B}^{\dagger\,T}_{j\cdot(L-z)} \neq \boldsymbol{0}.$$

Here E_d is a burst error of length greater than l.

Proof The theorem is proved by contradiction. First, consider the case where the d-bit burst error pattern is not at all included in the j-th frame. Assume that $S \cdot \mathbf{H}^{\dagger\,T}_{j\cdot(L-z)} = E$ and $S \cdot \mathbf{B}^{\dagger\,T}_{j\cdot(L-z)} = \mathbf{0}$ hold. Here E is a burst error pattern of length l or shorter. From $\mathbf{H}^{\dagger}_{j\cdot(L-z)} \cdot \mathbf{H}_{j\cdot(L-z)} = \mathbf{I}$ and $\mathbf{H}^{\dagger}_{j\cdot(L-z)} \cdot \mathbf{B}_{j\cdot(L-z)} = \mathbf{O}$ we have

$$\begin{aligned} S \cdot \begin{bmatrix} \mathbf{H}^{\dagger}_{j\cdot(L-z)} \\ \mathbf{B}^{\dagger}_{j\cdot(L-z)} \end{bmatrix}^T &= S \cdot \begin{bmatrix} \mathbf{H}^{\dagger\,T}_{j\cdot(L-z)} & \mathbf{B}^{\dagger\,T}_{j\cdot(L-z)} \end{bmatrix} \\ &= \begin{bmatrix} E & \mathbf{0} \end{bmatrix} \\ &= \begin{bmatrix} E \cdot \left(\mathbf{H}^{\dagger}_{j\cdot(L-z)} \cdot \mathbf{H}_{j\cdot(L-z)} \right) & E \cdot \left(\mathbf{H}^{\dagger}_{j\cdot(L-z)} \cdot \mathbf{B}_{j\cdot(L-z)} \right) \end{bmatrix} \\ &= E \cdot \mathbf{H}^{\dagger}_{j\cdot(L-z)} \cdot \begin{bmatrix} \mathbf{H}_{j\cdot(L-z)} & \mathbf{B}_{j\cdot(L-z)} \end{bmatrix}. \end{aligned}$$

Multiplying both sides from the right by nonsingular matrix

$$\mathbf{A}^{-1}_{j\cdot(L-z)} = \begin{bmatrix} \mathbf{H}^{\dagger}_{j\cdot(L-z)} \\ \mathbf{B}^{\dagger}_{j\cdot(L-z)} \end{bmatrix},$$

we obtain the following equation:

$$S = E \cdot \mathbf{H}^{\dagger\,T}_{j\cdot(L-z)}.$$

This shows that the syndrome S caused by d-bit burst error occurred outside the j-th frame matches with the syndrome of l-bit burst error E completely included inside the j-th frame, which is absurd. Therefore, if a d-bit burst error occurs and j-th frame does not include any error, $S \cdot \mathbf{H}^{\dagger\,T}_{j\cdot(L-z)}$ is equal to either error pattern E_d of length greater than l or $S \cdot \mathbf{B}^{\dagger\,T}_{j\cdot(L-z)} \neq \mathbf{0}$ holds.

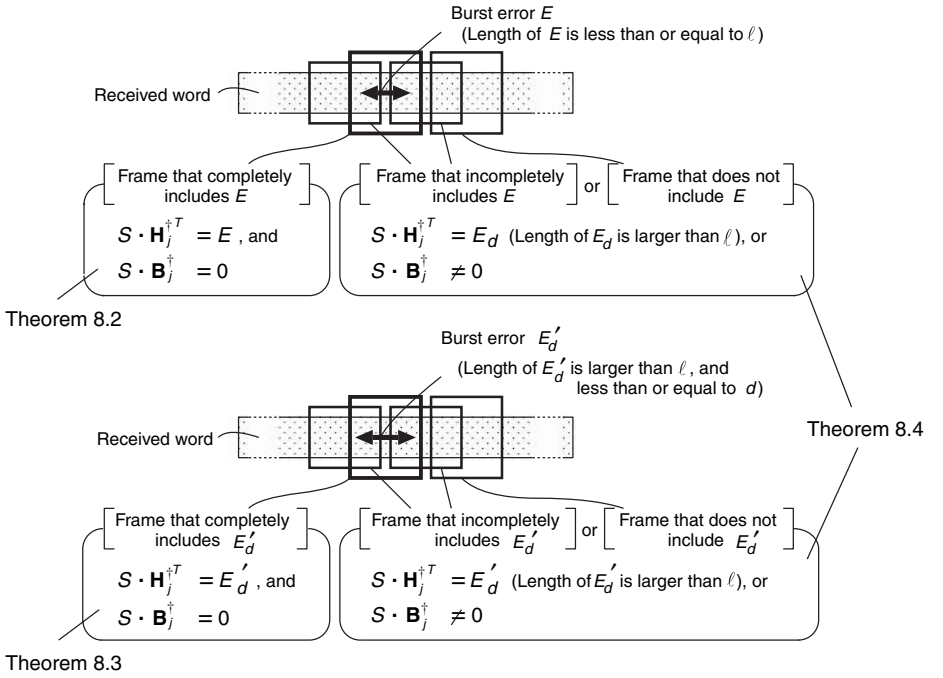


Figure 8.3 *Relations between burst error locations and $S \cdot \boldsymbol{H}_j^{\dagger T}$, $S \cdot \boldsymbol{B}_j^{\dagger T}$ for the ℓ-bit burst error correcting and d-bit burst error detecting codes.* Source: *[FUJI02]. © 2002 IEICE Japan.*

Next we consider the case where a part of the d-bit burst error pattern is included in the j-th frame but the error pattern is not completely included in that frame. We assume that $S \cdot \mathbf{H}_{j\cdot(L-z)}^{\dagger T} = E$ and $S \cdot \mathbf{B}_{j\cdot(L-z)}^{\dagger T} = \mathbf{0}$ hold. In this case the following equation is derived in the same way as the case where the j-th frame does not contain any error:

$$S = E \cdot \mathbf{H}_{j\cdot(L-z)}^{T}.$$

This indicates that the syndrome S for the case where a part of d-bit burst error pattern is included in the j-th frame, but not completely included in that frame, is identical to that for the case where the l-bit burst error pattern E is completely included in the j-th frame, and this contradicts the code function. Therefore, if a part of the d-bit burst error pattern is included in the j-th frame, then either $S \cdot \mathbf{H}_{j\cdot(L-z)}^{\dagger T}$ gives error pattern E_d having length greater than l, or $S \cdot \mathbf{B}_{j\cdot(L-z)}^{\dagger T} \neq \mathbf{0}$ holds. Q.E.D.

Figure 8.3 illustrates the relations between the burst error locations and $S \cdot \mathbf{H}_j^{\dagger T}$ and $S \cdot \mathbf{B}_j^{\dagger T}$ in accordance to Theorems 8.2, 8.3, and 8.4.

The following provides a decoding algorithm for the l-bit burst error correcting and d-bit burst error detecting codes.

Algorithm 8.1

***Step* 1.** *Calculate syndrome S. If $S = \boldsymbol{0}$, there is no error; otherwise, move to step* 2.

***Step* 2.** *Calculate* $S \cdot \boldsymbol{H}^{\dagger T}_{j\cdot(L-z)}$ *and* $S \cdot \boldsymbol{B}^{\dagger T}_{j\cdot(L-z)}$ *for each frame. If* $S \cdot \boldsymbol{H}^{\dagger T}_{j\cdot(L-z)}$ *is an l-bit burst error pattern E and* $S \cdot \boldsymbol{B}^{\dagger T}_{j\cdot(L-z)} = \boldsymbol{0}$*, then assume that an error* $E = S \cdot \boldsymbol{H}^{\dagger T}_{j\cdot(L-z)}$ *has occurred in the j-th frame. Next correct the error E, and finally the algorithm ends. If, however, for all frames* $S \cdot \boldsymbol{H}^{\dagger T}_{j\cdot(L-z)}$ *the burst error pattern is greater than l bits or* $S \cdot \boldsymbol{B}^{\dagger T}_{j\cdot(L-z)} \neq \mathbf{0}$ $(j = 0, \cdots, m-1)$*, we move to step* 3.

***Step* 3.** *Assume that an error of length greater than l bits, and less than or equal to d bits has occurred. Detect this error.*

Theorem 8.5 *For* $l \leq L \leq l + d$ *and* $z \geq l-1$*, the algorithm above can correct l-bit burst errors and detect d-bit burst errors where* $d > l$*.*

Theorem 8.5 can be easily proved by using Theorems 8.2, 8.3, and 8.4. Therefore the proof is omitted here.

Example 8.2 [FUJI02]

In this example we are interested in decoding the (22, 13) 3-bit burst error correcting and 4-bit burst error detecting Fire codes. From $l = 3$ and $d = 4$ we have $L = l + d = 7$ bits. Let the size of frame overlap $z = 2$ bits. The parity-check matrix $\mathbf{H}$ and submatrix $\mathbf{H}_{5\times j}$, where $j = 0,\ 1,\ 2,\ 3$, are shown below:

$\mathbf{H}_0$ $\mathbf{H}_5$ $\mathbf{H}_{10}$ $\mathbf{H}_{15}$

$$\mathbf{H} = \left[\begin{array}{cccccccccccccccccccc}
1&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0&0\\
0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1&0\\
0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&1\\
0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&0&0&1&0&0\\
1&0&0&1&0&1&1&1&0&0&1&0&1&1&1&0&0&1&0&1\\
0&1&0&1&1&1&0&1&0&1&1&1&1&0&0&1&0&1&1&1\\
0&0&1&0&1&1&1&0&0&1&0&1&1&1&0&0&1&0&1&1
\end{array}\right].$$

Assume that a 3-bit burst error (0000101) has occurred in the first frame (frame 1), starting at the 4-th bit. The syndrome is then given by $S = (000101010)$. Next $S \cdot \mathbf{H}^{\dagger T}_{5j}$ and $S \cdot \mathbf{B}^{\dagger T}_{5j}$ are calculated for each frame as follows:

$$\begin{aligned}
\text{Frame 0:}\quad & S \cdot \mathbf{H}^{\dagger T}_0 = (1001011), \quad S \cdot \mathbf{B}^{\dagger T}_0 = (01),\\
\text{Frame 1:}\quad & S \cdot \mathbf{H}^{\dagger T}_5 = (0000101), \quad S \cdot \mathbf{B}^{\dagger T}_5 = (00),\\
\text{Frame 2:}\quad & S \cdot \mathbf{H}^{\dagger T}_{10} = (0100011), \quad S \cdot \mathbf{B}^{\dagger T}_{10} = (01),\\
\text{Frame 3:}\quad & S \cdot \mathbf{H}^{\dagger T}_{15} = (0010001), \quad S \cdot \mathbf{B}^{\dagger T}_{15} = (00).
\end{aligned}$$

Now, $S \cdot \mathbf{B}^{\dagger T}_0 = \mathbf{0}$ holds for frames 1 and 3, but $S \cdot \mathbf{H}^{\dagger T}_{15} = (0010001)$ in frame 3 is not a 3-bit burst error pattern. Since $S \cdot \mathbf{H}^{\dagger T}_5 = (0000101)$ in frame 1 is a 3-bit burst error pattern, we assume that frame 1 is corrupted by error pattern (0000101), which can finally be corrected.

Next assume that a 4-bit burst error (0001101) has occurred in frame 1. Then the syndrome is given by $S = (001101000)$. In this case, $S \cdot \mathbf{H}_{5j}^{\dagger\,T}$ and $S \cdot \mathbf{B}_{5j}^{\dagger\,T}$ are calculated for each frame as follows:

$$\begin{aligned}
\text{Frame 0:} \quad & S \cdot \mathbf{H}_0^{\dagger\,T} = (0011010), \quad S \cdot \mathbf{B}_0^{\dagger\,T} = (00), \\
\text{Frame 1:} \quad & S \cdot \mathbf{H}_5^{\dagger\,T} = (0001101), \quad S \cdot \mathbf{B}_5^{\dagger\,T} = (00), \\
\text{Frame 2:} \quad & S \cdot \mathbf{H}_{10}^{\dagger\,T} = (1100111), \quad S \cdot \mathbf{B}_{10}^{\dagger\,T} = (10), \\
\text{Frame 3:} \quad & S \cdot \mathbf{H}_{15}^{\dagger\,T} = (1010010), \quad S \cdot \mathbf{B}_{15}^{\dagger\,T} = (10).
\end{aligned}$$

We observe that although $S \cdot \mathbf{B}_0^{\dagger\,T} = \mathbf{0}$ and $S \cdot \mathbf{B}_5^{\dagger\,T} = \mathbf{0}$, the error patterns (0011010) and (0001101) do not represent 3-bit burst error patterns. They are 4-bit burst error patterns, so the error is detected.

8.1.3 Parallel Decoding Circuit

Figure 8.4 shows the block diagram of a parallel decoding circuit for burst error control codes. The decoding circuit consists of a syndrome generator, m number of error pattern generators, an error pattern calculator, and an inverting circuit. All these circuits are implemented by combinational logic.

1. *Syndrome generator.* The parity-check matrix $\mathbf{H}$ and the received vector v are used to obtain the syndrome S. The parity checks of the received information bits in v correspond to '1' in each row vector of $\mathbf{H}$. For an $(N, N-R)$ code, we have R number of parity-check circuits, and therefore we have R syndrome bits.
2. *Error pattern generator.* For each frame there exists an error pattern generator that receives the R-bit syndrome vector as an input. Figure 8.5 shows an error

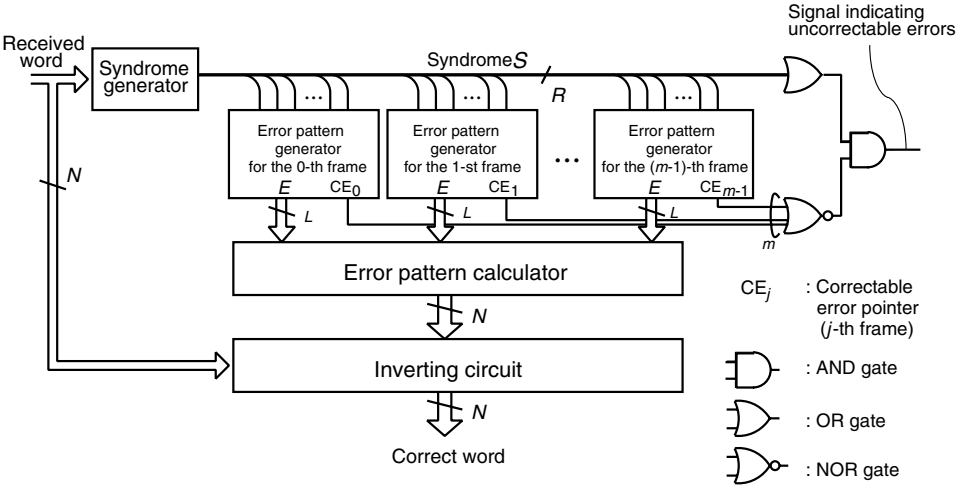


Figure 8.4 *Parallel decoding circuit for burst error control codes.* Source: *[FUJI02]. © 2002 IEICE Japan.*

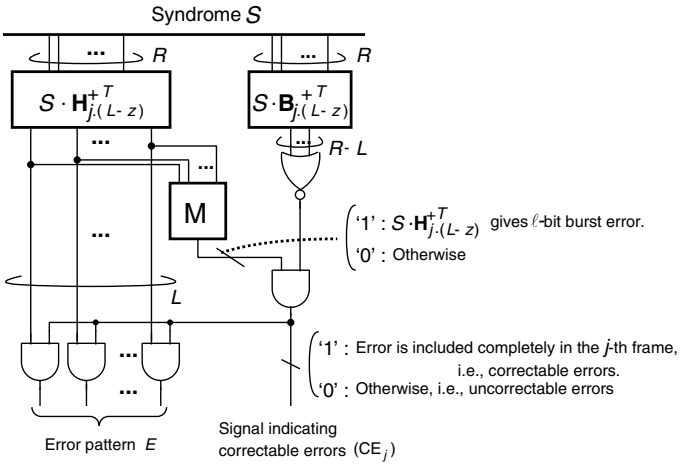


Figure 8.5 *Error pattern generator for j-th frame.* Source: *[FUJI02]. © 2002 IEICE Japan.*

pattern generator for the j-th frame. The output from the error pattern generator is an L-bit pattern E, which is either an l-bit burst error pattern or an all-zero vector. The j-th error pattern generator calculates $S \cdot \mathbf{H}^{\dagger\,T}_{j\cdot(L-z)}$ and $S \cdot \mathbf{B}^{\dagger\,T}_{j\cdot(L-z)}$. If $S \cdot \mathbf{H}^{\dagger\,T}_{j\cdot(L-z)}$ represents an l-bit burst error pattern, and $S \cdot \mathbf{B}^{\dagger\,T}_{j\cdot(L-z)}$ is an all-zero vector, then the output is error pattern $E = S \cdot \mathbf{H}^{\dagger\,T}_{j\cdot(L-z)}$; otherwise, the output is an all-zero vector. In Figure 8.5 the circuit M indicates whether or not the error is greater than l-bit burst error, that is, it gives output '0' if the error is greater than l bits, which finally gives all-zero error pattern. For example, an L-bit binary pattern $E = (e_0, e_1, \cdots, e_{L-1})$ represents an $(l+1)$-bit or greater error pattern if

$$\bigvee_{j=0}^{L-l-1} \left(e_j \wedge \bigvee_{i=j+l}^{L-1} e_i \right) = 1.$$

Figure 8.6 shows the logic circuitry M for this equation.

3. *Error pattern calculator and inverting circuit*. Using the m L-bit error patterns, the error pattern calculator outputs the overall N-bit error pattern. Since each frame overlaps with adjacent frames by z bits, the m number of L-bit vectors are adjusted to obtain the N-bit error pattern. Figure 8.7 shows the circuit that performs this adjustment by logically OR'ing the bits in the overlap, and outputs the N-bit error pattern. The inverting circuit performs bit by bit exclusive-OR addition of the N-bit error pattern and the received word.

Optimal $H_i^\dagger$ and $B_i^\dagger$ The smaller the number of 1's included in the $\mathbf{H}_i^\dagger$ and $\mathbf{B}_i^\dagger$ matrices, the smaller is the number of exclusive-OR gates needed to implement multiplication with the syndrome during error pattern generation. Therefore, for a given submatrix $\mathbf{H}_i$, the optimal $\mathbf{H}_i^\dagger$ and $\mathbf{B}_i^\dagger$ matrices are the ones with the least number of 1's.

The algorithm to obtain an optimal $\mathbf{H}_i^\dagger$ is given as below. Here $\mathbf{H}_i = [h_{i,0}\ h_{i,1}\ h_{i,2}\ \cdots\ h_{i,L-1}]$, where $h_{i,j}$ $(0 \le j \le L-1)$ is a binary column vector of length R.

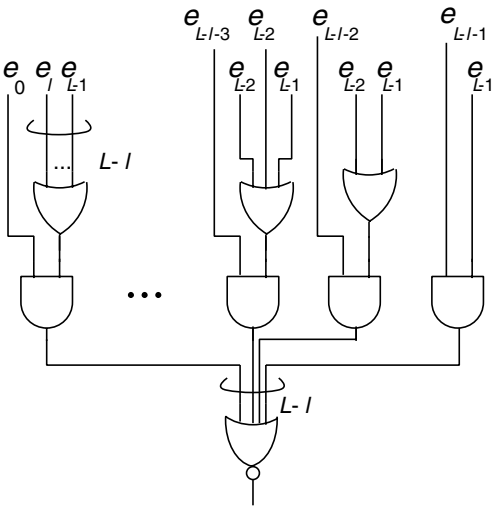


Figure 8.6 *An l-bit burst error detecting circuit (circuit M in Figure 8.5).* Source: *[FUJI02]. © 2002 IEICE Japan.*

Algorithm 8.2 for Optimal $H_i^{\dagger}$

Step 1. $t := 0$.

Step 2. *If* $t = L$, *end; else* $w := 1$.

Step 3. *Let* V *be a set of binary row vectors of length* R *and weight* w.

Step 4. *If* $V = \{\}$, *then* $w := w + 1$ *and go to step 3.*

Step 5. *Choose an arbitrary vector* x *from* V *and* $V := V - \{x\}$.

Step 6. *If* $xh_{i,j} = 0$ *and* $xh_{i,t} = 1$ *for all* $j \neq t$, *replace the* t*-th row of* $\boldsymbol{H}_i^{\dagger}$ *by* x, *set* $t := t + 1$, *and go to step 2.*

Step 7. *Go to step 4.*

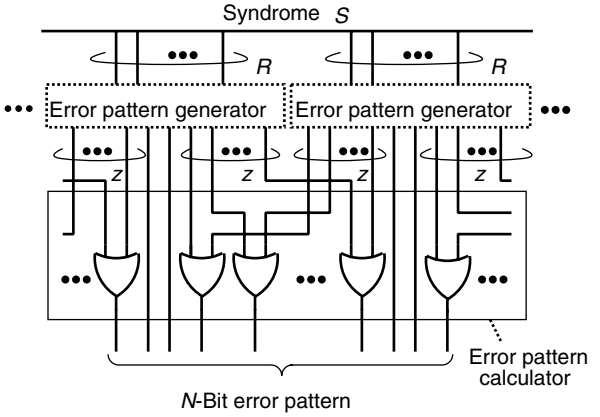


Figure 8.7 *Error pattern calculator for adjacent two frames.* Source: *[FUJI02]. © 2002 IEICE Japan.*

Since Algorithm 8.2 considers all the possible cases, the $\mathbf{H}_i^{\dagger}$ obtained is optimal. Next we present an algorithm to find out optimal $\mathbf{B}_i^{\dagger}$. In this case

$$\mathbf{B}_i^{\dagger} = \begin{bmatrix} u_{i,0} \\ u_{i,1} \\ u_{i,2} \\ \vdots \\ u_{i,R-L-1} \end{bmatrix},$$

where $u_{i,j}$ $(0 \le j < R - L)$ is a binary row vector.

Algorithm 8.3 for Optimal $B_i^{\dagger}$

Step 1. *Set $t := 0$, $w := 1$. Let V be a set of binary row vectors of length R and weight 1.*

Step 2. *If $t = R - L$, end.*

Step 3. *If $V = \{\}$, then $w := w + 1$. V is a set of binary row vectors of length R and weight w, which cannot be represented as a linear combinations of $u_{i,0}, \cdots, u_{i,t-1}$, but V is a set of all binary row vectors of length R and weight w for $t = 0$.*

Step 4. *Let x be an arbitrary vector in V, and $V := V - \{x\}$.*

Step 5. *If $x \cdot \boldsymbol{H}_i \neq 0$, go to step 3.*

Step 6. *#Exclude from V all the vectors that can be represented as a linear combination of x and $u_{i,0}, \cdots, u_{i,t-1}$. Set $u_{i,t} := x$, $t := t + 1$, and go to step 2.*

Since Algorithm 8.3 considers all the possible cases, the $\mathbf{B}_i^{\dagger}$ obtained is optimal.

8.1.4 Evaluation and Discussion

Circuit Gate Amount and Check-Bit Length Figure 8.8 shows the check-bit length and the parallel decoding circuit complexity of the Fire codes correcting burst

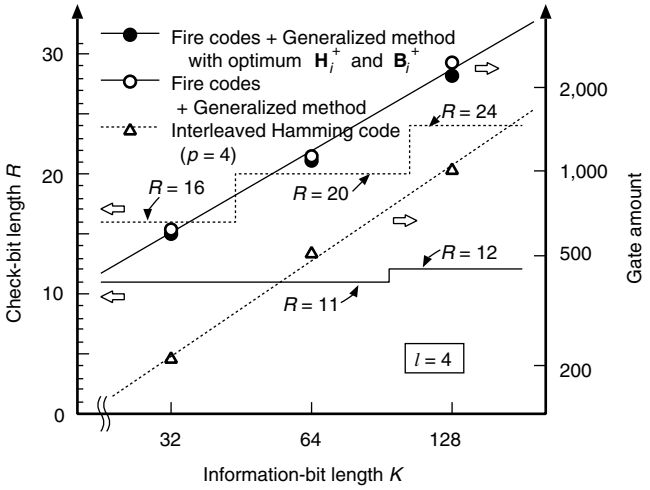


Figure 8.8 *Check-bit lengths and gate amounts of parallel decoding circuits for 4-bit burst error correcting codes.* Source: *[FUJI02]. © IEICE Japan.*

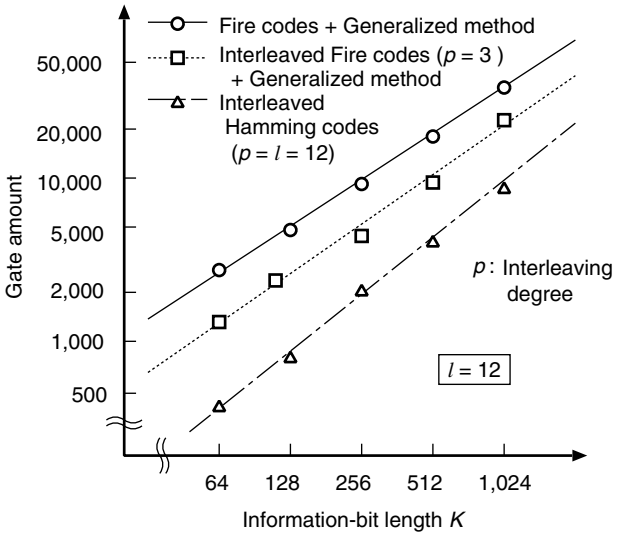


Figure 8.9 *Gate amounts of parallel decoding circuits for 12-bit burst error correcting codes.* Source: *[FUJI02]. © 2002 IEICE Japan.*

errors of length $l = 4$ bits. It also illustrates the hardware gate amount when $\mathbf{H}_i^{\dagger}$ and $\mathbf{B}_i^{\dagger}$ are optimal. Figures 8.9 and 8.10 illustrate the hardware complexity and the check-bit length of the Fire codes correcting burst errors of length $l = 12$ bits. Figures 8.8, 8.9, and 8.10 also show the check-bit lengths and the hardware complexities of degree l interleaved single-bit error correcting Hamming codes, which are capable of correcting l-bit burst errors. In this case the decoding circuit consists of l number of decoding circuits for single-bit error correcting code placed in parallel. Here the 4-input AND, OR, and NOR gates are counted as 1 gate, and the 2-input exclusive-OR gates are counted as 1.5 gates. The hardware complexity of the proposed decoding method is worse than that of decoding interleaved codes. However, the interleaved single-bit error correcting Hamming codes require many more check bits than the Fire codes with same burst error correction capability. In general,

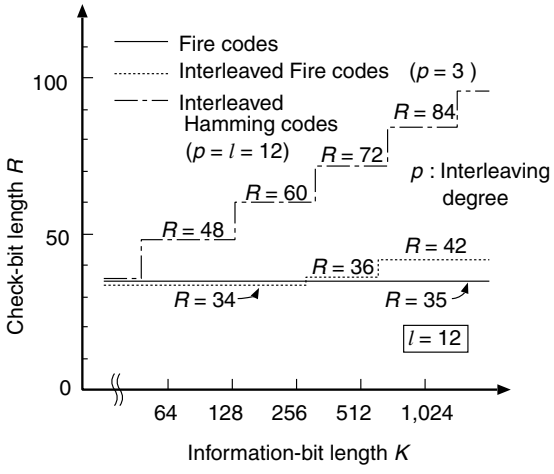


Figure 8.10 *Check-bit lengths for 12-bit burst error correcting codes.* Source: *[FUJI02]. © 2002 IEICE Japan.*

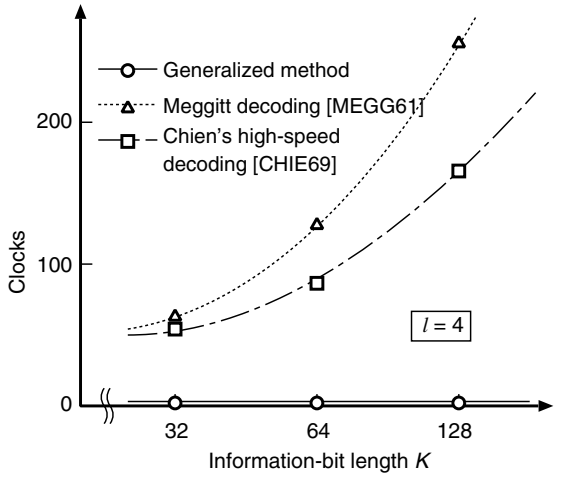


Figure 8.11 *Clocks for decoding 4-bit burst error correcting Fire code.* Source: *[FUJI02]. © 2002 IEICE Japan.*

for any burst error length, a smaller hardware complexity of the interleaved codes with interleaving degree p is achieved at the expense of introducing more check bits than necessary. We will discuss the interleaved Fire codes in Figures 8.9 and 8.10 last in this subsection.

For the 4-bit burst error correcting Fire codes, we will compare the indicated generalized decoding method with the existing methods, including Meggit decoding [MEGG61] and Chien's high-speed decoding [CHIE69] performed sequentially. As shown in Algorithm 8.1, the decoding speed is defined as the time required from syndrome generation to error correction. Figure 8.11 shows the number of clock cycles required for decoding. Compared to the sequential decoding methods that require a decoding speed proportional to the code length, the indicated method requires only two or fewer clock cycles for $K \leq 6,000$ information bits. The clock cycle is assumed to be 100 MHz and gate delay to be 1 ns. Again, the decoder gate amount is compared in Figure 8.12. In this figure, one shift register is assumed to have six gates.

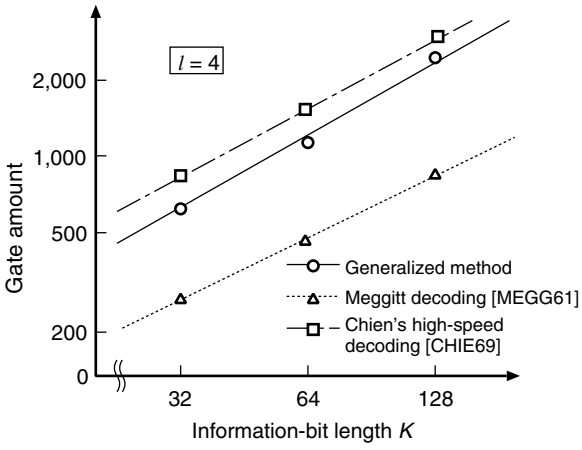


Figure 8.12 *Gate amounts of decoding circuits for 4-bit burst error correcting Fire Codes.* Source: *[FUJI02]. © 2002 IEICE Japan.*

Parallel Decoding Byte Error Control Codes Since byte errors are a special case of burst errors, the decoding method in this chapter can also be applied to the byte error control codes. In the case of byte errors, each byte corresponds to a frame, and the overlap between frames is zero. Therefore setting $L = b$, $z = 0$ in this decoding method yields a parallel decoder of the b-bit byte error correcting code. In such a case the error pattern calculator in Figure 8.4 becomes unnecessary because the frames are nonoverlapped.

We will now consider applying the decoding method mentioned in this chapter, for example, to the single-byte error correcting and double-byte error detecting Reed-Solomon code with code length m bytes. This class of codes has been mentioned precisely in Section 5.2. The parity-check matrix of the code has three rows and m columns with elements of $GF(2^b)$ as shown below:

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \cdots & \mathbf{I} & \cdots & \mathbf{I} \\ \mathbf{I} & \mathbf{T} & \cdots & \mathbf{T}^j & \cdots & \mathbf{T}^{m-1} \\ \mathbf{I} & \mathbf{T}^2 & \cdots & \mathbf{T}^{2j} & \cdots & \mathbf{T}^{2(m-1)} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_0 & \mathbf{H}_b & \cdots & \mathbf{H}_{jb} & \cdots & \mathbf{H}_{(m-1)b} \end{bmatrix}.$$

Note that $m \leq 2^b - 1$, $\mathbf{H}_{jb}$ $(0 \leq j < m)$ is a $3b \times b$ binary submatrix, $\mathbf{T}$ is a $b \times b$ companion matrix defined by a b-th degree primitive polynomial, and $\mathbf{I}$ is a $b \times b$ identity matrix, where $\{\mathbf{I},\ \mathbf{T},\ \mathbf{T}^2,\ \ldots,\ \mathbf{T}^{2(m-1)}\} \in GF(2^b)$.

Assuming that there is an error E in the j-th byte of the received word. The syndrome $S = [S_1\ S_2\ S_3]$ will be given by

$$S_1 = E, \quad S_2 = S_1 \cdot \mathbf{T}^j, \quad S_3 = S_1 \cdot \mathbf{T}^{2j}. \tag{8.4}$$

Each binary vector S_1, S_2, and S_3 has length b. The existing decoding method performs this calculation in parallel byte by byte in order to obtain the error pattern and the error location. Once all the relations given by Eq. (8.4) are satisfied in the j-th byte, we determine that the j-th byte is in error, and the error pattern is given by S_1.

This existing method completely matches the method described in this section where $\mathbf{B}_{jb}$ is appended to $\mathbf{H}_{jb}$. That is, $\mathbf{H}_{jb} = \begin{bmatrix} \mathbf{I} \\ \mathbf{T}^j \\ \mathbf{T}^{2j} \end{bmatrix}$ and $\mathbf{B}_{jb} = \begin{bmatrix} \mathbf{O} & \mathbf{O} \\ \mathbf{I} & \mathbf{O} \\ \mathbf{O} & \mathbf{I} \end{bmatrix}$, so we get

$$\mathbf{H}_{jb}^{\dagger} = [\mathbf{I} \quad \mathbf{O} \quad \mathbf{O}], \quad \mathbf{B}_{jb}^{\dagger} = \begin{bmatrix} \mathbf{T}^j & \mathbf{I} & \mathbf{O} \\ \mathbf{T}^{2j} & \mathbf{O} & \mathbf{I} \end{bmatrix}.$$

These matrices combined with the syndrome S in Theorem 8.1 yield Eq. (8.4).

Once $\mathbf{B}_{jb}$ is given, the method described in this section matches the existing ones, even the ones using single-byte error correcting codes such as the Hong-Patel codes [HONG72]. This is because the decoding method of this section is very general, so it completely includes the existing parallel decoding methods. In addition Algorithms 8.2 and 8.3, which give the optimal $\mathbf{H}_{jb}^{\dagger}$ and $\mathbf{B}_{jb}^{\dagger}$, can be used with less decoding hardware.

The hardware gate amount of this generalized decoding method is compared to the conventional method for the single 4-bit byte error correcting codes. The result obtained

by optimization of the algorithms above shows that it is possible to decrease the decoding circuit gate amount by 2% to 10%. Similar results are found for different byte lengths and for other byte error correcting RS codes.

The conventional parallel decoding method can be applied to the burst / byte error control codes designed by using only elements over $GF(2^b)$. The conventional method cannot be applied, for example, to the single-byte error correcting and single-byte plus single-bit error detecting (SbEC-(Sb + S)ED) codes shown in Section 6.4. However, the generalized parallel decoding method is applicable to these codes and in general, to any linear error control codes.

Decoding of Interleaved Codes Here we consider the parallel decoding of the interleaved burst error correcting codes. We can apply degree-p interleaving to an (l/p)-bit burst error correcting codes, and obtain an l-bit burst error correcting code. For this interleaved burst error correcting code, the entire decoding circuit usually consists of p number of parallel decoding circuits of the (l/p)-bit burst error correcting code. The hardware gate amount of this decoder is much less than that of the generalized decoding method. For example, the hardware gate amount is reduced to half of that of the generalized decoder for the case where the burst length $l = 12$ bits and the interleaving degree $p = 3$, as shown in Figure 8.9. Furthermore, as shown in Figure 8.10, interleaved b-bit burst error correcting Fire codes with interleaving degree p require much less check-bit length compared to that of the interleaved single-bit error correcting Hamming codes with degree $l = pb$.

8.1.5 Multiple Burst / Byte Error Correction

We have discussed generalized parallel decoding single-burst / byte error correcting codes in the previous subsections. As we will demonstrate below, these types of code can be extended to parallel decoding multiple burst / byte error control codes. In other words, when t number of l-bit burst errors are given by $E_1,\ E_2,\ E_3,\ \cdots,\ E_t$, completely included in t frames each with length L starting at bit positions $i_1,\ i_2,\ i_3,\ \cdots,\ i_t$, respectively, the syndrome S is given by

$$S = E_1 \cdot \mathbf{H}_{i_1}^T + E_2 \cdot \mathbf{H}_{i_2}^T + E_3 \cdot \mathbf{H}_{i_3}^T + \cdots + E_t \cdot \mathbf{H}_{i_t}^T.$$

Then we annex an $R \times (R - tL)$ matrix $\mathbf{B}_{(i_1,\, i_2,\, i_3,\, \cdots,\, i_t)}$ to an $R \times tL$ matrix $\mathbf{H}_{(i_1,\, i_2,\, i_3,\, \cdots,\, i_t)} = [\mathbf{H}_{i_1}\mathbf{H}_{i_2}\mathbf{H}_{i_3}\cdots\mathbf{H}_{i_t}]$ to obtain an $R \times R$ nonsingular matrix $\mathbf{A}_{(i_1,\, i_2,\, i_3,\, \cdots,\, i_t)} = [\mathbf{H}_{(i_1,\, i_2,\, i_3,\, \cdots,\, i_t)}\mathbf{B}_{(i_1,\, i_2,\, i_3,\, \cdots,\, i_t)}]$. Its inverse matrix $\mathbf{A}^{-1}_{(i_1,\, i_2,\, i_3,\, \cdots,\, i_t)}$ is given by

$$\mathbf{A}^{-1}_{(i_1,\, i_2,\, i_3,\, \cdots,\, i_t)} = \begin{bmatrix} \mathbf{H}^{\dagger}_{(i_1,\, i_2,\, i_3,\, \cdots,\, i_t)} \\ \mathbf{B}^{\dagger}_{(i_1,\, i_2,\, i_3,\, \cdots,\, i_t)} \end{bmatrix}.$$

In this case $\mathbf{H}^{\dagger}_{(i_1,\, i_2,\, i_3,\, \cdots,\, i_t)}$ and $\mathbf{B}^{\dagger}_{(i_1,\, i_2,\, i_3,\, \cdots,\, i_t)}$ are $tL \times R$ and $(R - tL) \times R$ matrices, respectively. So the following conditions hold:

$$S \cdot \mathbf{H}^{\dagger\, T}_{(i_1,i_2,i_3,\cdots,i_t)} = [E_1E_2E_3\cdots E_t],$$
$$S \cdot \mathbf{B}^{\dagger\, T}_{(i_1,i_2,i_3,\cdots,i_t)} = \mathbf{0}.$$

From these equations, we obtain the error patterns E_j $(1 \leq j \leq t)$ and the error locations i_j, $1 \leq j \leq t$. Hence we can correct these multiple burst errors.

The discussion above also holds for multiple-byte errors. So the decoding method presented here can further be applied to multiple-byte error correcting codes.

8.2 PARALLEL DECODING CYCLIC BURST ERROR CORRECTING CODES

This section presents a simplified method for parallel decoding burst error correcting cyclic codes [UMAN03,05]. With this method we define the entire decoding process in terms of a binary companion matrix $\mathbf{T}$ that generates a multiplicative group under the usual matrix multiplication. This method does not involve any matrix inversions.

8.2.1 Preliminaries

Let $\mathbf{C}$ be a binary $(N,\ K)$ cyclic or shortened quasi-cyclic code with l-bit burst error-correcting capability. Assume that $\mathbf{C}$ is defined by a generator polynomial $\mathbf{g}(x)$ over $GF(2)$ with degree R, where $R = N - K$. That is,

$$\mathbf{g}(x) = \sum_{i=0}^{R} g_i x^i, \qquad g_i \in GF(2),$$

where $g_0 = g_R = 1$. Furthermore $N \leq \lambda$, where λ denotes the exponent of $\mathbf{g}(x)$. Without loss of generality, we can assume that the j-th column of the parity-check matrix $\mathbf{H}$ of the code $\mathbf{C}$ is given by the vector of binary coefficients in the remainder obtained by dividing x^j by $\mathbf{g}(x)$. Therefore the parity-check matrix $\mathbf{H}$ can be written as

$$\mathbf{H} = \begin{bmatrix} | & | & | & & | & & | & | \\ \beta^0 & \beta^1 & \beta^2 & \cdots & \beta^i & \cdots & \beta^{N-2} & \beta^{N-1} \\ | & | & | & & | & & | & | \end{bmatrix},$$

where, for $0 \leq i \leq N-1$, $\begin{vmatrix} \\ \beta^i \\ \\ | \end{vmatrix}$ denotes the R-bit binary coefficient vector representing $x^i \bmod \mathbf{g}(x)$.

The elements $x^i \bmod \mathbf{g}(x)$, for $i = 0,\ 1,\ 2,\ \cdots,\ \lambda - 1$, form a multiplicative group where $x^\lambda \bmod \mathbf{g}(x) = x^0 \bmod \mathbf{g}(x) = 1$. Therefore we can represent these elements in companion matrices as well. Define an $R \times R$ companion matrix corresponding to $x^i \bmod \mathbf{g}(x)$ as follows:

$$\mathbf{T}^i = \begin{bmatrix} | & | & | & & | \\ \beta^i & \beta^{i+1} & \beta^{i+2} & \cdots & \beta^{i+R-1} \\ | & | & | & & | \end{bmatrix}.$$

Then the set $\{\mathbf{T}^0,\ \mathbf{T}^1,\ \mathbf{T}^2,\ \mathbf{T}^3,\ \ldots,\ \mathbf{T}^{\lambda-1}\}$ is also a multiplicative group with the usual matrix multiplication over $GF(2)$. The matrix $\mathbf{T}$ that generates the multiplicative group is

given in terms of the binary coefficients of the generator polynomial of the code. This matrix has been presented earlier, in Definition 2.9 of Subsection 2.1.3 and in Definition 5.1 of Subsection 5.1.1.

8.2.2 Parallel Decoding

For parallel decoding, it is preferable to perform error pattern and error location calculations as matrix or vector multiplications over $GF(2)$. The matrix or vector multiplication over $GF(2)$ corresponds to simple exclusive-OR additions and is therefore suitable for combinational logic realizations. The companion matrix $\mathbf{T}$ corresponding to the generator polynomial $\mathbf{g}(x)$ is a handy tool because the error pattern calculation becomes treated as matrix and vector multiplications, and not polynomial calculations. Let $E \in GF(2^R) - \{0\}$ represent an R-bit error pattern (R-bit row vector) starting at the j-th bit of the received word. The syndrome generated by this error is given by

$$S = E \cdot \mathbf{T}^j.$$

Then, since $\mathbf{T}$ is a nonsingular matrix, the error pattern E is simply given by

$$E = S \cdot \mathbf{T}^{-j} = S \cdot \mathbf{T}^{\lambda - j}.$$

For burst error correction we need information about the burst error pattern as well as the location where the burst error occurs. As illustrated in Figure 8.2, we divide the received word into a number of overlapping L-bit frames where each frame overlaps with its adjacent frames by z bits. In this case L is equal to R. So the last frame in the received word will have less than or equal to R bits.

The N-bit received word is divided by m overlapping R-bit frames where each frame overlaps its adjacent frames by exactly z bits. If $z = l - 1$, every l-bit burst error pattern is completely included in a unique frame, as was shown in Subsection 8.1.2.

The binary column vectors of the $\mathbf{H}$ matrix corresponding to the j-th frame are shown below:

$$\begin{matrix} | & | & | & & | \\ \beta^{j(R-l+1)} & \beta^{j(R-l+1)+1} & \beta^{j(R-l+1)+2} & \cdots & \beta^{j(R-l+1)+(R-1)} \\ | & | & | & & | \end{matrix}.$$

These column vectors are exactly the same as that of the companion matrix $\mathbf{T}^{j(R-l+1)}$. Therefore the j-th frame is associated with the companion matrix $\mathbf{T}^{j(R-l+1)}$ for syndrome calculations and $\mathbf{T}^{\lambda - j(R-l+1)}$ for error pattern calculations. In order to perform l-bit burst error correction on the received word, we need to locate the frame that is corrupted by an R-bit error pattern representing an l-bit burst error. Theorem 8.6 illustrates how the location of the corrupted frame and the corresponding error pattern can be determined uniquely.

Theorem 8.6 *Let $\boldsymbol{C}$ be an (N, K) binary cyclic or shortened quasi-cyclic code with l-bit burst error correcting capability. Let S be the syndrome generated by a received word v of $\boldsymbol{C}$. Then $S \cdot \boldsymbol{T}^{\lambda - j(R-l+1)} = E$, where $0 \leq j \leq m-1$, such that $E \in GF(2^R)$ represents an l-bit burst error pattern if and only if v is corrupted by the error pattern E at the j-th frame.*

TABLE 8.1 Decoder Gate Amount for 4-Bit Burst Error-Correcting Codes

Decoder components	$K = 32$	$K = 64$	$K = 128$
Syndrome generator	249	492	927
Syndrome decoder	535	1,000	1,740
Error corrector	90	157	270
Error detector	7	8	10
Total	881	1,657	2,947

Source: [UMAN05]. © 2005 IEEE.

Here $m = \lceil (N - z)/(R - z) \rceil$, *and* $\mathbf{T}$ *denotes the* $R \times R$ *companion matrix corresponding to the generator polynomial of* $\mathbf{C}$.

Proof Suppose that the received word v is corrupted by the error pattern $E \in GF(2^R) - \{0\}$ at the j-th frame of the received word. Then $S = E \cdot \mathbf{T}^{j(R-l+1)}$. Therefore $E = S \cdot \mathbf{T}^{\lambda - j(R-l+1)}$ holds, as required. Now, assume that $S \cdot \mathbf{T}^{\lambda - w(R-l+1)} = E^{\dagger}$, where $E^{\dagger} \in GF(2^R) - \{0\}$ represents a correctable l-bit burst error pattern for some $0 \leq w \leq m - 1$. Then $S = E^{\dagger} \cdot \mathbf{T}^{w(R-l+1)}$, that is

$$E \cdot \mathbf{T}^{j(R-l+1)} = E^{\dagger} \cdot \mathbf{T}^{w(R-l+1)}. \tag{8.5}$$

No l-bit burst error patterns are included in two frames when the received word is divided by $m = \lceil (N - z)/(R - z) \rceil$ R-bit frames where adjacent frames overlap by exactly $l - 1$ bits. Therefore, since the code is l-bit burst error correcting, Eq. (8.5) implies $j = w$. However, then $E = E^{\dagger}$ because $\mathbf{T}$ is a nonsingular matrix. This completes the proof. Q.E.D.

Table 8.1 shows the hardware complexity of the parallel decoding circuit for the 4-bit burst error correcting code that is generated by $\mathbf{g}(x) = (x^{11} + 1)(x^4 + x + 1)$. The codes considered in this table are shortened quasi-cyclic codes of the original (165, 150) code with information lengths K equal to 32, 64, and 128 bits. In this table, a 4-input AND / OR gate is counted as 1 gate and a 2-input XOR (exclusive-OR) gate as 1.5 gates.

8.3 TRANSIENT BEHAVIOR OF PARALLEL ENCODING / DECODING CIRCUITS OF ERROR CONTROL CODES

The relation between the transient behavior (i.e., *glitches*) of the encoding / decoding circuit and the $\mathbf{H}$ matrix construction of an error control code (ECC) has not been addressed before. In the parallel encoding and decoding circuits of error correcting codes, glitches are known to consume extra power and induce simultaneous switching noise [LO05]. It is shown in this section that the probability of a given number of glitches that may accumulate in the encoding / decoding circuit exhibits a Gaussian-like distribution. An estimation methodology was developed so that the transient behavior of an ECC for very large word length can be predicted. As a result the principle of *minimum-weight & equal-weight-row* construction of $\mathbf{H}$ matrix (defined in Subsection 3.1.1) is demonstrated to be the best design strategy.

8.3.1 Introduction

For high-speed transfer of data, parallel encoding / decoding of the error control codes is essential. However, parallel decoding circuits can be very bulky and they constitute primarily from exclusive-OR (XOR) circuits. For example, for a 128-bit information length, a 12-bit burst error correcting code, such as a Fire code, will need about 2,000 to 5,000 logic gates in its parallel decoding circuit, as shown in Subsection 8.1.4. Most of these logic gates, around 80% of them, are XOR gates. As the advent of system-on-chip (SoC), these parallel decoding circuits are an integrated part of a system fabricated along side other circuits and subsystems. Hence the transient behavior (i.e., *glitches*) of these parallel decoding circuits need to be carefully analyzed.

We use the term *transient behavior* to describe the circuit activities between the insertion of inputs to the final stabilization of the circuit outputs. Exclusive-OR (i.e., XOR) is a hazardous Boolean function such that its tendency to produce glitches is inherent to the function itself. In other words, there is no way to avoid glitches in the XOR circuits. A glitch is a temporary and unwanted logic state occurs at the circuit output. Often a glitch will not complete a full logic swing and thus may not have impact if the circuit at the next stage is not fast enough to respond. However, as the circuit speed increases, even a half-swing glitch may induce some response at the subsequent circuit. This motivates us to perform in-depth analysis of the transient behavior of the encoding and decoding circuits of the error control codes.

Besides creating a difficult situation for the timing of logic designs [LAVA93, BENI00], glitches in general will consume extra energy [MEHT95, ROY99, BENI00, GHOS04] and elevate the *simultaneous switching noise* (*SSN*) [CHEN97, PARR01, TANG02, ROSS04]. As demonstrated in [TANG02], a *power bus noise* is tied to the total number of switching activities accumulated at any given time. Because glitches cause extra switching activities, they intensify the power bus noise problem. So the total number of glitches is directly proportional to the degree of impact at any given time. The increase of power bus noise will in fact add circuit delays [JIAN00, BAI01].

Until recently the transient behavior of the encoding / decoding circuits of error control codes had rarely been studied. In one recent study [ROSS04], the SSN was analyzed for the Hamming encoded bus. This work concentrates on how the added check bits, to be carried by the additional wires of the bus, can enlarge the SSN problem. In another recent work [GHOS04], the memory traces of benchmark programs are used to determine the probability of transient behavior in each memory bit or pair. Such information is then used to select the **H** matrix in Hsiao's odd-weight-column SEC-DED code such that the power consumption can be minimized. This is essentially the extension of the idea of [ZHOU00], where the switching activities can be reduced in an XOR circuit if the probability of transient in individual input bit is known.

The XOR circuits in a parallel decoding circuit usually have large number of inputs and are typically formed as an XOR tree. The glitches will not only be generated by any XOR gate but will also propagate along the sensitized path to the primary output. The most devastating effect is that the glitches will accumulate in succeeding stages of propagation. The impact to the power bus, in the form of energy consumption or the causes for simultaneous switching noise, is all the glitches accumulated at every stage of the XOR tree.

Modern-day SoC's are extremely dense with tens and even hundreds of millions of transistors. The simultaneous switching of so many transistors contributes to the problem

of power bus instability. Because the power bus noise is so closely allied to the total number of switching activities at any given time, and the glitches cause extra switching activities, it follows that glitches create noise problem.

8.3.2 Generation, Propagation, and Accumulation of Glitches in Exclusive-OR Tree Circuits

Recent work on glitches in exclusive-OR (XOR) functions is in the area of power estimation. In this context the interest in glitches is usually in a more accurate estimate of total power consumption. Hence these studies are mostly concerned with the probability that glitches will occur under various delay models and / or input pattern possibilities [MEHT95, ROY99]. Here we are interested in the worst case of glitch phenomena, namely (1) all input patterns are equally likely to occur, (2) signals cannot be controlled, so they arrive exactly at the same time, and (3) individual logic gates have different delay times.

There are many XOR gate implementation possibilities. Without lost of generality, we will concentrate on the structure of the large XOR function without specifying the actual implementations of the XOR gates involved.

Glitch Generation The generation and propagation of glitches in logic circuit are well known. For an n-variable exclusive-OR function, a glitch is generated whenever more than one input variable is changed simultaneously. Because the changes can never occur exactly at the same time, a multiple-variable change is accomplished by changing one variable at a time. Note that we are dealing with the worst-case analysis, so any nonzero difference is considered. We may describe a multiple-variable change as a traverse on the binary cube.

Figure 8.13 illustrates the well-known 2-input Boolean functions. Figure 8.13(a) shows an OR / NOR function. A glitch can occur only if the input is going from (01) to (10), or from (10) to (01), and only if (01) changes to (00) before arriving at (10). Similarly only one such possibility exists for the AND / NAND function, as shown in Figure 8.13(b). However, for the XOR / XNOR function shown in Figure 8.13(c), any simultaneous changes in both input variables will induce a glitch. The assumption here is, of course, that no two signals can change exactly at the same time. This assumption may be relaxed for slower operating frequency but not for modern-day high-speed logic circuits.

For an n-input XOR function, there are a total of 2^n possible input patterns. For transient analysis, we are concerned about the input sequence pairs, the pairing of the

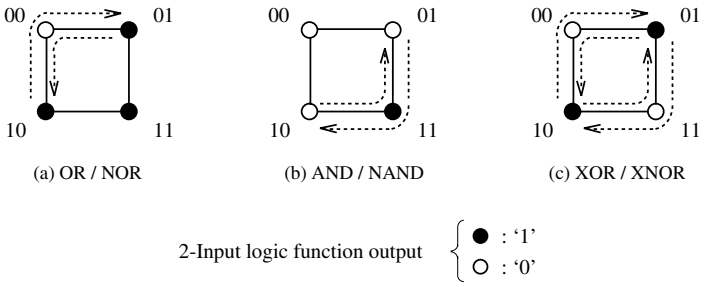


Figure 8.13 *Generation of glitches in 2-input Boolean functions.*

previous input and the present input. There are 2^{2n} possible input pairs. To simplify the discussion, we use N to denote a $0 \rightarrow 0$ or a $1 \rightarrow 1$ transient, and T to represent a $0 \rightarrow 1$ or a $1 \rightarrow 0$ transient. For example, when the inputs to a 4-input XOR function changes from (0, 0, 1, 1) to (0, 1, 0, 1), we denote the transition as (N, T, T, N). Since each transient pattern represents 2^n input pairs, there are only 2^n possible transient patterns.

A glitch is generated at the output of a 2-input XOR function if both inputs receive T. Thus

$$T \oplus T \Rightarrow G,$$

where G denotes a temporary change to the opposite logic value such that a logic 1 appears while the logic value is supposed to maintain at 0, and vice versa. This can be easily verified by looking at Figure 8.13(c).

Glitch Propagation Typically a large exclusive-OR function is synthesized as an XOR tree constructed from 2-input XOR gates. In this construction, a glitch can propagate through the full height, $\log_2 n$, for an n-variable function. The exact condition that makes XOR gates easily testable also easily guarantees the propagation of glitches through the XOR gates. Whenever a glitch arrives at one of the inputs, a 2-input XOR gate will always propagate this glitch to its output regardless of the condition of the other input. Hence

$$N \oplus G \Rightarrow G$$

and

$$T \oplus G \Rightarrow T + G.$$

Here we use the "$+$" sign to represent the combination of a logic value change T and a glitch. The logic value may change before the glitch occurs, and it may change after the glitch. However, we use the same notation because our interest is in the propagation of the glitch.

What happens when both inputs receive glitch? Since the glitch can be propagated regardless, glitches on both inputs will be propagated to the output. There is, of course, an odd chance that both glitches will arrive at the same time. In that case both glitches may get cancel out if they are of different polarities or only one glitch is generated at the output if the same polarity. In the analysis we will ignore such an unlikely event and assume that the two glitches are both propagated to the output. In other words, the glitches are accumulated as they propagate through the circuit.

Glitch Accumulation There are two types of glitch accumulations: spatial and temporal. *Spatial glitch accumulation* is the accumulation of glitches, which appear one by one, at the output of a large XOR circuit. This is a result of glitch propagation as described above. Since we use "$+$" to represent the accumulated effect of transient behavior, "Σ" will be used to represent the accumulation of the glitches. We find that

$$T \oplus (T + G) \Rightarrow G + G \Rightarrow \sum^{2} G,$$

TABLE 8.2 Transient Behavior of XOR Functions

$\oplus$	N	T	$\Sigma^d G$	$T + \Sigma^d G$
N	N	T	$\Sigma^d G$	$T + \Sigma^d G$
T	T	G	$T + \Sigma^d G$	$\Sigma^{d+1} G$
$\Sigma^{d'} G$	$\Sigma^{d'} G$	$T + \Sigma^{d'} G$	$\Sigma^{d+d'} G$	$T + \Sigma^{d+d'} G$
$T + \Sigma^{d'} G$	$T + \Sigma^{d'} G$	$\Sigma^{d'+1} G$	$T + \Sigma^{d+d'} G$	$\Sigma^{d+d'+1} G$

Source: *[LO05]. © 2005 IEEE.*

or in general,

$$T \oplus (T + \sum^{d} G) \Rightarrow \sum^{d+1} G.$$

Table 8.2 summarizes these transient behaviors.

Temporal glitch accumulation refers to the fact that all glitches, which occur at different times but still within the same circuit transition period, have the collected effect on the power bus. For a large XOR circuit built from 2-input XOR gates, the temporal accumulation counts all the glitches on all 2-input XOR gates from the insertion of the input to the stabilization of the output.

Figure 8.14 shows an 8-input exclusive-OR function realized with seven 2-input XOR gates in the conventional XOR tree construction. Here, for simplicity, N and T are

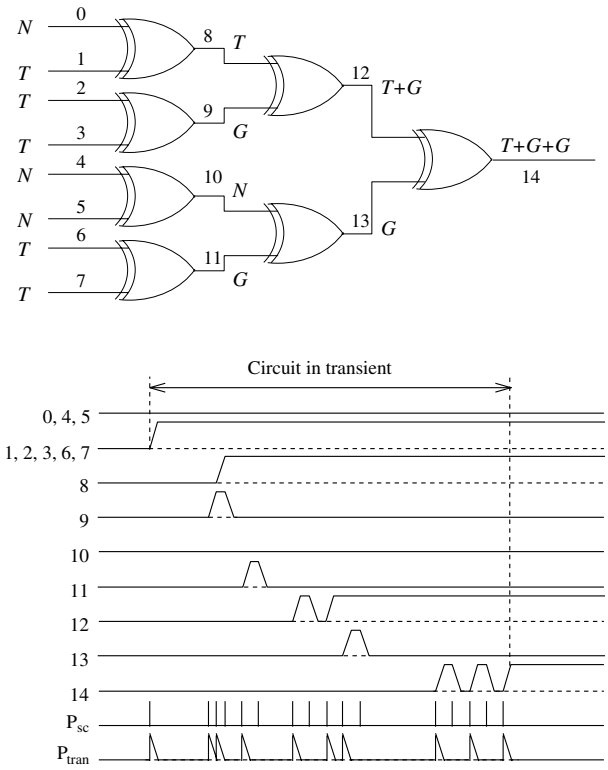


Figure 8.14 *Glitch generation, propagation, and accumulation in an 8-input XOR function.* Source: *[LO05]. © 2005 IEEE.*

represented by a $0 \rightarrow 0$ and a $0 \rightarrow 1$ transients, respectively. The glitch, G, is thus represented by a momentary change to logic 1 state, as illustrated at nets 9, 11, 12, 13, and 14. These glitches affect both P_{sc} and P_{tran} because they behave just like logic state changes. Here P_{sc} is the power consumption from the short circuit current every time a logic gate is changing its output state. P_{tran} is the power consumption due to the changing of the output capacitance of a logic gate. The worst-case scenario is, of course, when all four inputs change simultaneously. As the glitches are propagated forward, they accumulate. In the given example we see how glitches may accumulate and show their effects at the primary output, and thus the spatial accumulation.

The temporal accumulation, on the other hand, is the total number of glitches shown in Figure 8.14 including all the internal nodes. Recall that T represents a change of input, either $0 \rightarrow 1$ or $1 \rightarrow 0$. The changes in the input logic values induce a change in the output logic value within one clock cycle. Therefore glitches that occur along the circuit paths to the output will have a cumulative impact. For the worst-case analysis, we simply add up the number of all the glitches, G's, in the circuit. The number of *temporal accumulated glitches* (*TAG*) in Figure 8.14 is six. Figure 8.14 shows P_{SC} and P_{Tran} separately so their impacts can be clearly distinguished. The true impact to the power supply should be the sum of these two factors.

8.3.3 Glitches in Encoding / Decoding Circuits of Error Control Codes

The encoding / decoding circuits of ECCs have multiple XOR functions working together. The relations among these XOR functions, or parity functions, are defined by the **H** matrix of the code. Essentially the total weight of **H** matrix, meaning the total number of 1's in **H**, dictates how complex the encoding / decoding circuits will be.

Exhaustive Examinations of Transient Behavior The **H** matrix of single error correcting and double error detecting (SEC-DED) Hsiao's odd-weight-column codes is shown below. We will concentrate on the encoding circuit in the following analysis. The decoding circuit will have the similar transient behavior.

$$\mathbf{H} = \begin{array}{c} \begin{array}{cccccccc|ccccc} d_0 & d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & c_0 & c_1 & c_2 & c_3 & c_4 \end{array} \\ \left[\begin{array}{cccccccc|ccccc} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array}. \quad (8.6)$$

Figure 8.15 shows the encoding circuit of the above (13, 8) odd-weight-column SEC-DED code. The transient behavior analysis methods described above are applied exhaustively. Table 8.3 shows the summary of the results. WT denotes the number of T's in the $k = 8$ inputs to the encoding circuit. The example shown in Figure 8.15 has a transient input pattern with WT = 4. However, for the five XOR tree circuits in this encoding circuit, they receive transient input patterns with WT = 2, 4, 2, 2, and 2 for

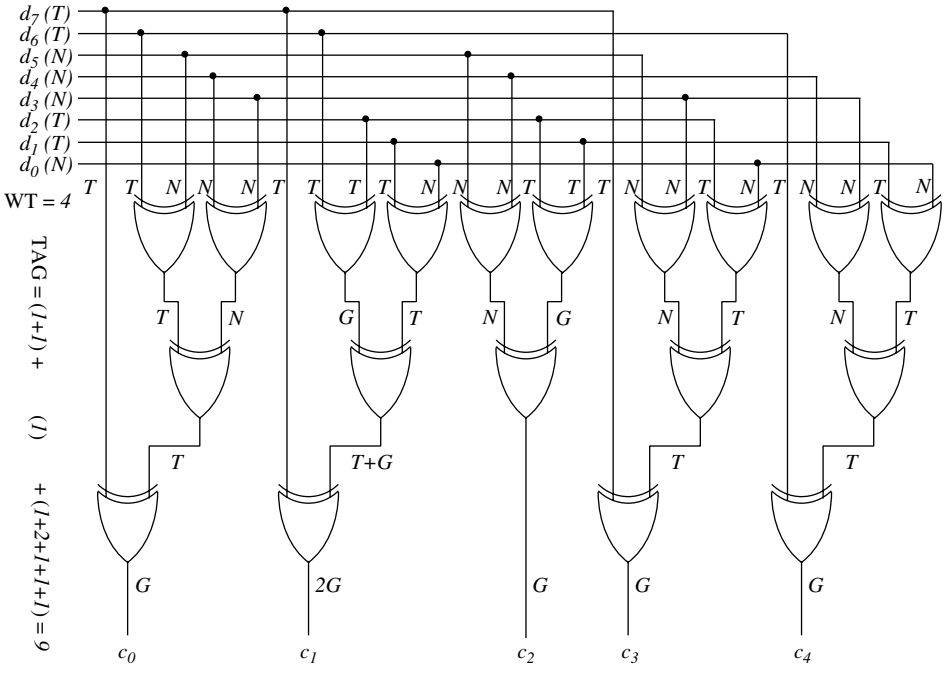


Figure 8.15 *Encoding circuit of the (13, 8) odd-weight column SEC-DED code.* Source: *[LO05]. © IEEE.*

circuits c_0, c_1, c_2, c_3, and c_4, respectively. Obviously a different construction of the **H** matrix will lead to a very different encoding circuit design and thus different transient behavior.

Since even the transient patterns with same weight may result in different TAG numbers, we count the frequency in which the same number of TAG is produced when the transient patterns have the same weight. For instance, the one transient pattern of WT = 4 shown in Figure 8.15 results in TAG = 9. There are total $\binom{8}{4} = 70$ transient patterns of WT = 4 and actually only 19 of them (shown as 'freq. = 19' in Table 8.3) will result in TAG = 9, as shown in Table 8.3. In fact there are five transient patterns of WT = 4 that

TABLE 8.3 TAG Frequencies for all Transient Weights (WT) of the SEC-DED Encoding Circuit in Figure 8.15

WT < 2		WT = 2		WT = 3		WT = 4		WT = 5		WT = 6		WT = 7	
TAG	freq.	TAG	freq.	TAG	freq.	TAG	freq.	TAG	freq.	TAG	freq.	TAG	freq.
0	9	1	4	4	6	7	5	11	2	16	5	22	6
		2	9	5	12	8	9	12	16	17	10	24	1
		3	8	6	13	9	19	13	11	18	4	28	1
		4	3	7	11	10	6	14	9	19	2		
		5	2	8	9	11	18	15	7	20	4		
		6	2	9	3	12	9	16	2	21	2		
				10	0	13	1	17	9	22	1	WT = 8	
				11	2	14	2					TAG	freq.
						15	1					28	1

Source: *[LO05]. © 2005 IEEE.*

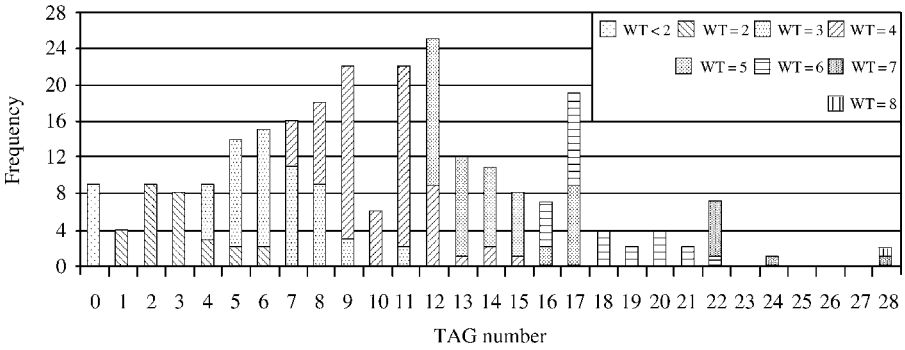


Figure 8.16 *Histogram of the frequency distribution and accumulation given in Table 8.3.* Source: *[LO05]. © IEEE.*

produce only TAG = 7, and one transient pattern of WT = 4 that produces TAG = 15. Figure 8.16 shows the histogram of the results shown in Table 8.3. In Figure 8.16, the total number of transient patterns that may produce the same number of TAG are added. For instance, TAG = 9 may be produced by 3 transient patterns of WT = 3 and 19 patterns of WT = 4. Thus the total number of transient patterns that may result in TAG = 9 is 22.

Since we assume that all inputs have equal arrival rates, we derive the probability for a particular TAG number as follows:

$$Pr(\text{TAG} = \alpha) = \frac{\text{Number of transient pairs that produce TAG} = \alpha}{\text{Total Number of possible transient pairs}}.$$

For example, from Table 8.3 we find that two patterns of WT = 3, 18 patterns of WT = 4, and two pairs of WT = 5 can all produce TAG = 11. Therefore there are 22 out of the total 256 transient pairs that will result in TAG = 11. Accordingly, we say $Pr(\text{TAG} = 11) = 22/256 = 0.086$. Figure 8.17 shows the distribution of the TAG probabilities for the (13, 8) SEC-DED code. The x-axis represents the TAG number and the y-axis denotes the probability.

Next we perform an exhaustive examination of the following (22, 16) odd-weight-column SEC-DED code. In this case the number of 1's per row is exactly 8 for all rows. This means that all six XOR circuits have 8 inputs, a power of two. The previous (13, 8)

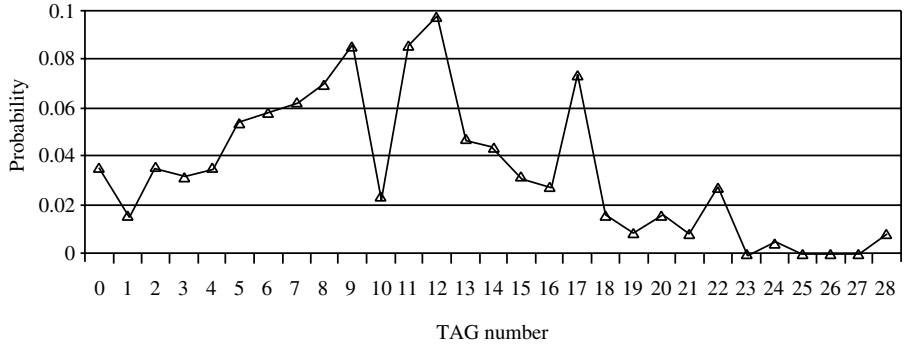


Figure 8.17 *Distribution of TAG probabilities of the (13, 8) SEC-DED code.* Source: *[LO05]. © 2005 IEEE.*

encoding circuits did not have such perfectly even distribution of **H** row weights and the XOR circuits did not have power of two inputs.

$$\mathbf{H} = \begin{array}{c} \begin{matrix} d_0 & d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & d_8 & d_9 & d_{10} & d_{11} & d_{12} & d_{13} & d_{14} & d_{15} & c_0 & c_1 & c_2 & c_3 & c_4 & c_5 \end{matrix} \\ \left[\begin{array}{cccccccccccccccc|cccccc} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array} \tag{8.7}$$

Table 8.4 shows the TAG frequencies for the (22, 16) SEC-DED code with all possible 2^{16} input transient pairs. Since the **H** row weights are evenly distributed, the TAG

TABLE 8.4 TAG Frequencies for All Transient Weights (WT) of the SEC-DED Encoding Circuit of the (22, 16) Odd-Weight-Column SEC-DED Code

WT < 2		WT = 3		WT = 4		WT = 5		WT = 6		WT = 7		WT = 8	
TAG	freq.	TAG	freq.	TAG	freq.	TAG	freq.	TAG	freq.	TAG	freq.	TAG	freq.
0	17	3	67	6	152	9	87	12	8	17	90	21	10
		4	143	7	310	10	404	13	138	18	743	22	418
		5	142	8	460	11	914	14	734	19	1924	23	1556
WT = 2		6	97	9	385	12	1033	15	1552	20	2779	24	2800
TAG	freq.	7	55	10	261	13	867	16	1937	21	2532	25	3194
0	7	8	36	11	143	14	563	17	1650	22	1697	26	2368
1	38	9	14	12	74	15	292	18	1080	23	1015	27	1386
2	33	10	3	13	23	16	135	19	518	24	421	28	712
3	24	11	2	14	9	17	51	20	253	25	164	29	272
4	8	12	1	15	3	18	15	21	96	26	53	30	106
5	6					19	4	22	30	27	19	31	42
6	3					20	2	23	10	28	2	32	2
8	1					21	1	24	2	30	1	33	4

WT = 9		WT = 10		WT = 11		WT = 12		WT = 13		WT = 14		WT = 15	
TAG	freq.	TAG	freq.	TAG	freq.	TAG	freq.	TAG	freq.	TAG	freq.	TAG	freq.
26	90	30	8	36	87	42	152	48	67	54	7	63	16
27	743	31	138	37	404	43	310	49	143	55	38		
28	1924	32	734	38	914	44	460	50	142	56	33		
29	2779	33	1552	39	1033	45	385	51	97	57	24		
30	2532	34	1937	40	867	46	261	52	55	58	8		
31	1697	35	1650	41	563	47	143	53	36	59	6		
32	1015	36	1080	42	292	48	74	54	14	60	3	WT = 16	
33	421	37	518	43	135	49	23	55	3	62	1	TAG	freq.
34	164	38	253	44	51	50	9	56	2			72	1
35	53	39	96	45	15	51	3	57	1				
36	19	40	30	46	4								
37	2	41	10	47	2								
39	1	42	2	48	1								

Source: *[LO05]. © 2005 IEEE.*

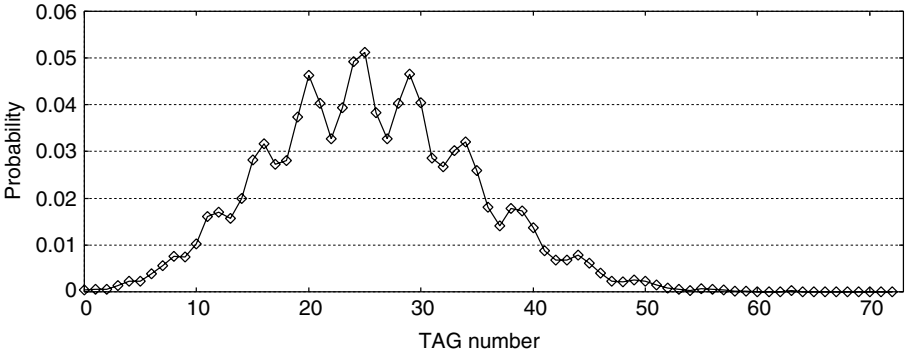


Figure 8.18 *Distribution of TAG probabilities of the (22,16) SEC-DED code.* Source: *[LO05]. © IEEE.*

frequency distribution is also symmetrical. For example, the frequency column for $\mathrm{WT} = 2$ is identical to that of $\mathrm{WT} = 14$. Similarly the frequency columns for $\mathrm{WT} = 3$ is identical to that of $\mathrm{WT} = 13$, and so on. Figure 8.18 shows the result of exhaustive examination of the (22, 16) SEC-DED code. As expected, the probability distribution looks almost like normal distribution.

Estimating the Transient Behavior The previous example demonstrates how an ECC's encoding circuit can be analyzed. However, an exhaustive examination is only possible for short word lengths. To analyze the transient behavior of an ECC with long word lengths, the following estimation technique is required.

From the examination of encoding circuits above recall that the probability distribution tends to be Gaussian-like regardless of the **H** matrix's construction. Not surprisingly, normal, or Gaussian, distributions are commonly found in situations like this. Recall also that the TAG count for a transient pair is the total of the TAG in the individual XOR circuit. For each transient pair, different XOR circuits will receive different transient patterns according to the **H** matrix. Therefore a different transient pair with the same WT will produce different TAG numbers. However, in Table 8.3 we saw that for transient pairs with the same weight, the TAG numbers also are Gaussian-like in their distribution.

Let $\mathrm{TAG}_{avg}(\mathrm{WT} = i)$ denote the average TAG number for $\mathrm{WT} = i$, we estimate that the mean and standard deviation of the Gaussian distribution as

$$\mu = \mathrm{TAG}_{avg}\left(\mathrm{WT} = \frac{k}{2}\right) \tag{8.8}$$

and

$$\sigma = \mathrm{TAG}_{avg}\left(\mathrm{WT} = \frac{k}{2}\right) - \mathrm{TAG}_{avg}\left(\mathrm{WT} = \frac{3k}{8}\right). \tag{8.9}$$

Here we follow the convention that uses k for the information-bit length, r for the check-bit length, and n to denote the codeword length such that $n = k + r$. In practice, $\mathrm{TAG}_{avg}(\mathrm{WT} = i)$ may be difficult to obtain as all $\binom{k}{i}$ transient pairs have to be exhaustively examined. On the other hand, as we will see in Subsection 8.3.5, the

maximum number can be easily calculated. Let $\text{TAG}_{\max}(\text{WT} = i)$ denote the worst-case TAG number when $\text{WT} = i$. Then the alternative estimation can be made as follows:

$$\mu_{\max} = \text{TAG}_{\max}\left(\text{WT} = \frac{k}{2}\right) \tag{8.10}$$

and

$$\sigma_{\max} = \text{TAG}_{\max}\left(\text{WT} = \frac{k}{2}\right) - \text{TAG}_{\max}\left(\text{WT} = \frac{3k}{8}\right). \tag{8.11}$$

Example 8.3

Consider the (13, 8) SEC-DED code encoding circuit shown in Figure 8.15. From Table 8.3 find $\text{TAG}_{avg}(\text{WT} = 4) = 10$ and $\text{TAG}_{avg}(\text{WT} = 3) = 6.43$. Therefore the estimated $\mu = 10$ and $\sigma = 3.57$. We will use only the worst-case numbers $\text{TAG}_{\max}(\text{WT} = 4) = 15$ and $\text{TAG}_{\max}(\text{WT} = 3) = 11$. Then $\mu_{\max} = 15$ and $\sigma_{\max} = 4$. Figure 8.19 shows the plots based on the actual data and the previous estimation schemes. Obviously the first estimation scheme where the average numbers are used fits better with the real data. The worst-case estimation tends to be pessimistic because the estimation always gives lower probability for the lower TAG numbers and higher probability for higher TAG numbers.

Example 8.4

From Table 8.4, for the (22, 16) SEC-DED code encoding circuit, finds $\text{TAG}_{avg}(\text{WT} = 8) = 25.15$ and $\text{TAG}_{avg}(\text{WT} = 6) = 16.46$. Therefore the estimated $\mu = 25.15$ and $\sigma = 8.69$. The worst-case numbers are $\text{TAG}_{\max}(\text{WT} = 8) = 33$ and $\text{TAG}_{\max}(\text{WT} = 6) = 24$. Thus $\mu_{\max} = 33$ and $\sigma_{\max} = 9$. Figure 8.20 shows the plots based on the actual data and from the estimations. Again, observe that the estimation based on avenge TAG number matches very well with the real data. The estimation using the maximum TAG

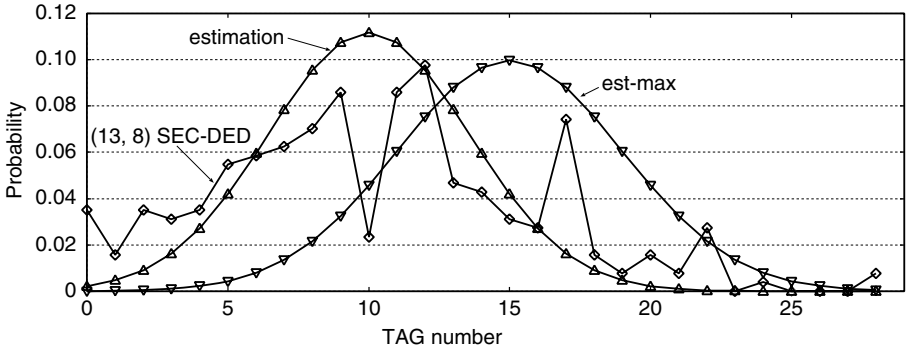


Figure 8.19 *Plots of TAG probabilities in the encoding circuit of (13, 8) SEC-DED codes: From actual data, shown as "(13, 8)SEC-DED," from the estimation using average TAG numbers, shown as "estimation," and from the estimation using the maximum TAG numbers, shown as "est-max."* Source: *[LO05]. © 2005 IEEE.*

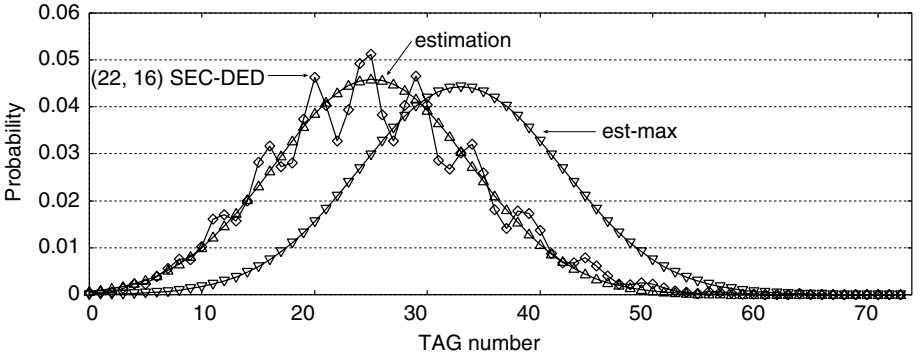


Figure 8.20 *Plots of TAG probabilities in the encoding circuit of (22, 16) SEC-DED codes: From actual data, shown as "(22, 16)SEC-DED," from the estimation using average TAG numbers, shown as "estimation," and from the estimation using the maximum TAG numbers, shown as "est-max."* Source: *[LO05]. © 2005 IEEE.*

numbers is naturally pessimistic. In this example the second method overestimates the mean by $\mu_{\max} - \mu = 7.85$ and standard deviation by $\sigma_{\max} - \sigma = 0.31$. However, as the maximum TAG numbers can be easily obtained, this second method is more practical.

An important observation can also be made about the difference between Figures 8.19 and 8.20. In the case of the (22, 16) SEC-DED code, the estimations fit better the real data because of its perfectly even distribution of **H** row weights. However, for most practical code lengths such perfect arrangement is rarely found. Therefore the estimation, in most cases, matches only loosely the real data as shown in Figure 8.19.

Importance of Weight in H Matrix Design As defined in Section 3.1, a *minimum-weight & equal-weight-row code* is a code that has the minimum-weight in the **H** matrix and in which the number of 1's in each row of **H** is equal, or as close as possible, to the average number. We will use an odd-weight-column code design to examine this practical design principle.

Example 8.5 [LO05]

Consider the Hsiao's (72, 64) SED-DED code, where $n = 72,\ k = 64$, and $r = 8$. Since there are eight check bits, there are eight rows in the **H** matrix. We can only pick from the odd-weight columns. So the minimum weight **H** matrix is to be constructed by all $\binom{8}{1}$ weight-1 columns, for the check bits, all $\binom{8}{3}$ weight-3 columns, and eight weight-5 columns. The total number of 1's, or the weight, in the **H** matrix is thus 208. When choosing the eight weight-5 column correctly, we can make each row having exactly 26 1's for encoding. This is thus a minimum-weight & equal-weight-row code. An example of such a (72, 64) code can be found in Figure 4.2 in Chapter 4. For this code we can find $\mu_{\max} = 192$ and $\sigma_{\max} = 48$.

Next, construct a ''medium-weight'' version of (72, 64) code. This time we will use 32 weight-3 columns and 32 weight-5 columns, in addition to the eight weight-1 columns. The weight of the **H** matrix is then 256. By carefully selecting the **H** columns, we can achieve 32 one's for each row. The estimation parameters for this code are $\mu_{\max} = 256$ and $\sigma_{\max} = 80$.

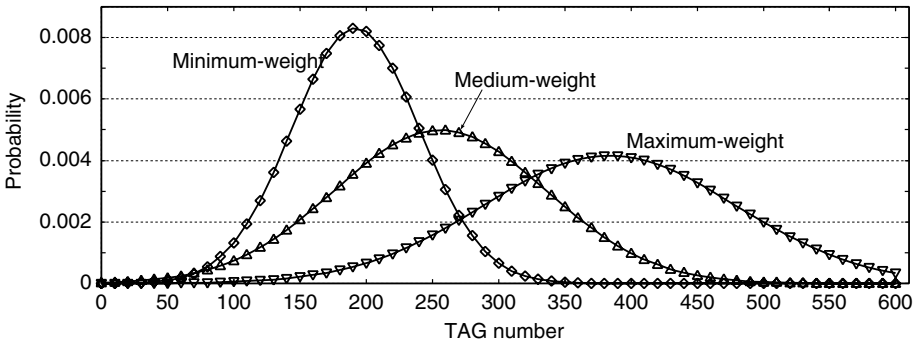


Figure 8.21 *Estimated TAG probability plots of the (72, 64) odd-weight-column SEC-DED codes.* Source: *[LO05]. © 2005 IEEE.*

Finally, construct a ''maximum-weight'' version. This is accomplished by using all $\binom{8}{5}$ weight-5 columns and all $\binom{8}{7}$ weight-7 columns, in addition to the eight weight-1 columns. The weight of the **H** matrix is 336, or 42 one's per row. Note that all three versions are of equal-weight-row type but with different **H** matrix weights. This code has $\mu_{\max} = 384$ and $\sigma_{\max} = 96$.

Figure 8.21 shows the estimated TAG probability plots of all three versions above. Obviously the minimum-weight version has the best transient behavior. Further the gap between the minimum-weight and the medium-weight seems to be smaller than that of the medium-weight and the maximum-weight. One possible explanation is that the increase in the average number of one's in a row is 6 in the former case and 10 in the latter case. The number of one's in a row is also the number of inputs to parity functions.

8.3.4 Two Potential Solutions to Reduce Glitch Accumulation

Here we will examine two potential solutions to reduce the glitch accumulation: gate sharing and pipelining. Typically the gate-sharing possibilities are exploited by the logic synthesis process to reduce the number of logic gates needed in a multi-output circuit.

Example 8.6 [LO05]

The encoding circuit of the (13, 8) SEC-DED code can be further optimized as shown in Figure 8.22. In the original form, as shown in Figure 8.15, when d_6 and d_7 both have T's, glitches are generated and propagated through circuits for c_0 and c_1, respectively. With the gate-sharing version such a scenario will still create the same glitch accumulation at c_0 and c_1, but the glitch on the shared XOR gate will only be counted once for the TAG. Figure 8.23 shows the TAG probabilities of the encoding circuits shown in Figures 8.15 and 8.22, respectively.

From Figure 8.23 we find that gate sharing can reduce the maximum TAG from 28 to 22. Also the distribution is shifted significantly to the left, or lower TAG numbers then have higher probability. However, gate sharing is very difficult to formulate as it is an NP-hard problem. Therefore there is no easy way to predict the reduction of glitch accumulation based only on the **H** matrix design.

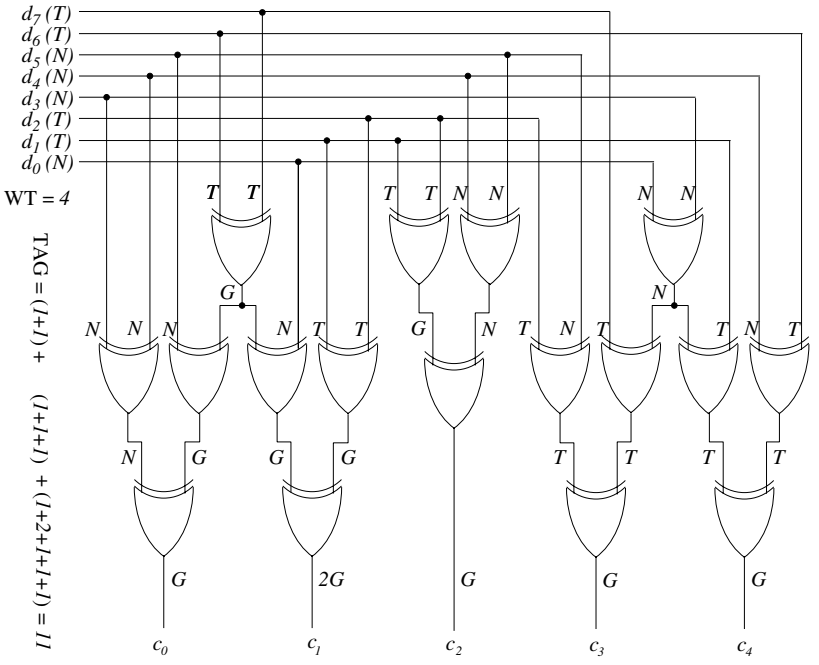


Figure 8.22 *Encoding circuit of the (13, 8) SEC-DED code with shared XOR gates.* Source: *[LO05]. © 2005 IEEE.*

Another potential solution is to insert pipeline registers. Figure 8.24 shows an example of how pipeline registers (flip-flops) are inserted in the c_0 circuit of the (13, 8) SEC-DED encoding circuit. The 5-input XOR function is constructed in three stages. The flip-flops are inserted at each stage. In the schematic of Figure 8.24, note that d_7 is connected directly to the XOR gate at the third stage. However, for the pipeline operations each stage holds the different intermediate results from different data inputs. Therefore two flip-flops are inserted in the d_7 line such that it can hold different data and be synchronized with the rest of the circuit.

Recall from the previous examples that the accumulation of glitches is mainly due to the easy propagation of glitches by an XOR gate. The pipeline register can effectively filter out the glitches and thus guarantee no glitch propagation. Note that while *G* can

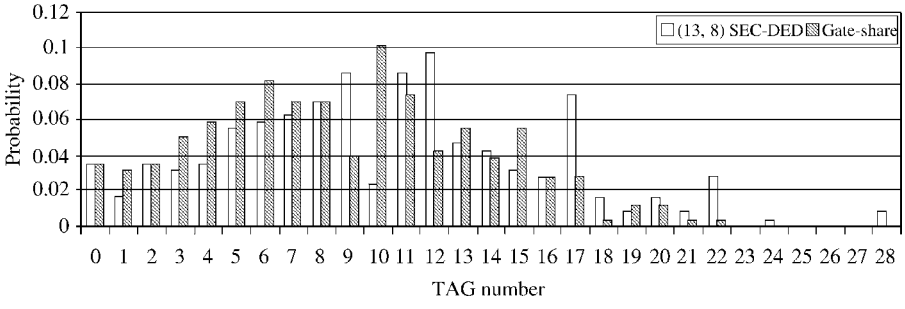


Figure 8.23 *TAG probabilities plots of the (13, 8) SEC-DED encoding circuit and of the gate-shared version.* Source: *[LO05]. © 2005 IEEE.*

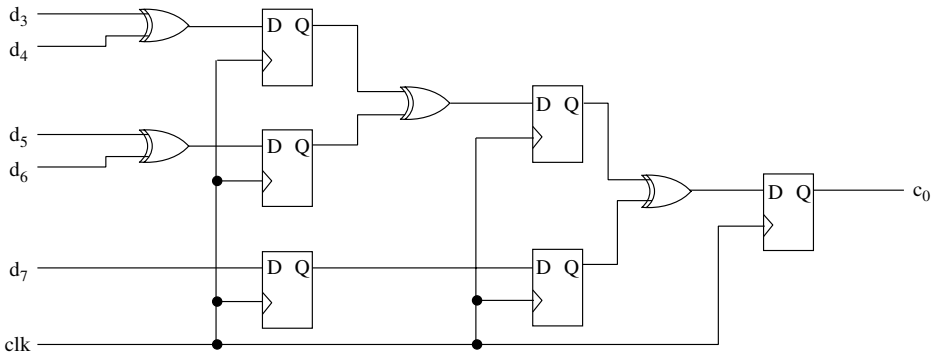


Figure 8.24 *Encoding circuit for c_0 of the (13, 8) SEC-DED code with pipeline register (flip-flops) inserted.* Source: *[LO05]. © 2005 IEEE.*

be blocked by the flip-flop, T will not be filtered out. In this extreme example the worst-case scenario, with all five inputs having T, will generate only two glitches, or TAG = 2. Recall that the worst case is TAG = 6 for an 5-input XOR function without pipeline.

We believe that the pipelining of the encoding / decoding circuits is inevitable for the ever-increasing word length. The insertion of pipeline registers will add to the hardware overhead as well as extra delay time. The rise in delay time due to the flip-flops is not a major concern because the performance is measured in throughput rather than the actual latency time of encoding data. Nevertheless, the scenario shown in Figure 8.24 is simply too costly for practical applications. The hardware overhead can be reduced by inserting pipeline registers every few XOR levels. However, this has to also be related to the targeted data rate of the overall design.

8.3.5 Maximum Temporal Accumulated Glitches (TAGs) and Matrix Code Design

In the worst-case estimation technique presented previously, the mean and standard deviation are calculated directly from the maximum TAG numbers. The maximum TAG number can be further estimated with just the knowledge of how the 1's in the **H** matrix is distributed. In other words, only a few simple steps will be needed to estimate the transient behavior of any error control code defined by an **H** matrix, regardless of the information length.

First, let's revisit the topic discussed previously about obtaining the TAG number in an XOR circuit. The upper bound of the number of TAG of an m-input XOR function can be calculated as follows:

$$TAG_m = TAG_m(WT = m) = \frac{m \log_2 m}{2}, \tag{8.12}$$

when m is a power of two and $TAG_1 = 0$. If m is not a power of two, then we first find its binary form such that

$$m = m_{l-1} \times 2^{l-1} + m_{l-2} \times 2^{l-2} + \cdots + m_1 \times 2^1 + m_0 \times 2^0,$$

where $l = \lceil \log_2 m \rceil$. We assume that the m-input XOR function will be composed systematically with smaller XOR functions whose input numbers are all power of two. The worst-case TAG can be computed as

$$TAG_m = \sum_{i=1}^{l-1} m_i TAG_{2^i} + \sum_{i=1}^{l-2} m_i \left(\left\lfloor \frac{m}{2} \right\rfloor - \sum_{j=i+1}^{l-1} m_j 2^{j-1} \right) + m_{l-1} \left\lfloor \frac{m}{2} \right\rfloor + (m_0 - 1)2^{w-1}, \tag{8.13}$$

where $w \geq 1$ such that m_w is the smallest nonzero digit in the binary form of m besides m_0.

Example 8.7 [LO05]

The worst-case TAG number of a 28-input XOR circuit can be computed as follows: First, we find $m_4 = m_3 = m_2 = 1$, $m_1 = m_0 = 0$, and $w = 2$. Therefore

$$TAG_{28} = TAG_{2^4} + TAG_{2^3} + TAG_{2^2} + (14 - (2^2 + 2^3)) + (14 - 2^3) + 14 - 2^1 = 68.$$

We will now use this formula to calculate the maximum TAG number of the minimum-weight & equal-weight-row Hsiao's SEC-DED code. Table 8.5 shows Hsiao's code parameters and the maximum TAG number for several power of 2 information bit lengths up to 1,024. All codes listed in Table 8.5 are minimum-weight & equal-weight-row codes.

TABLE 8.5 Code Parameters and Maximum TAG Numbers of the Minimum-Weight & Equal-Weight-Row Hsiao SEC-DED Codes

n	k	r	**H** structure[a]	**H** Weight[b]	Average **H**[b]	Maximum TAG
13	8	5	$\binom{5}{1} + 8/\binom{5}{3}$	24	4.8	28
22	16	6	$\binom{6}{1} + 16/\binom{6}{3}$	48	8	72
39	32	7	$\binom{7}{1} + 32/\binom{7}{3}$	96	13.7	183
72	64	8	$\binom{8}{1} + \binom{8}{3} + 8/\binom{8}{5}$	208	26	504
137	128	9	$\binom{9}{1} + \binom{9}{3} + 44/\binom{9}{5}$	472	52.4	1,376
266	256	10	$\binom{10}{1} + \binom{10}{3} + 136/\binom{10}{5}$	1,040	104	3,560
523	512	11	$\binom{11}{1} + \binom{11}{3} + 347/\binom{11}{5}$	2,230	202.8	8,764
1,036	1,024	12	$\binom{12}{1} + \binom{12}{3} + \binom{12}{5} + 12/\binom{12}{7}$	4,704	392	20,976

Source: *[LO05]. © 2005 IEEE.*

[a] The notation $j/\binom{r}{i}$ means that j out of all possible $\binom{r}{i}$ combinations is used.

[b] Encoding **H** matrix where

H weight: Total number of 1's in encoding **H** matrix

Average **H**: **H** weight divided by number of rows r = average number of 1's in a row.

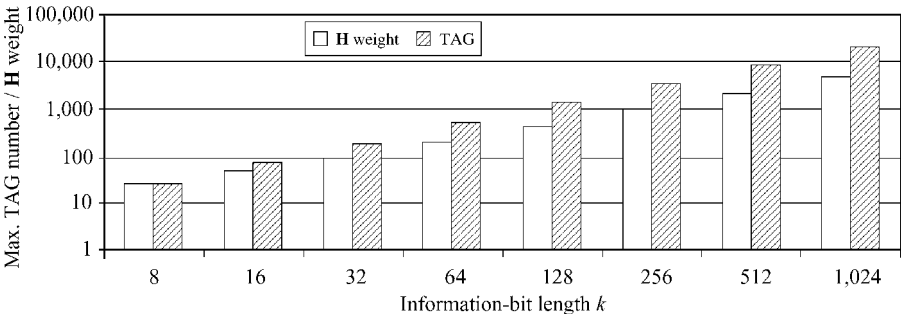


Figure 8.25 ***H*** *weights and maximum TAGs of odd-weight-column SEC-DED codes for various k.* Source: *[LO05]. © 2005 IEEE.*

From this table we observe that the maximum TAG number is intimately tied to the **H** weight, or the total number of 1's in **H**. Also the maximum TAG grows exponentially with respect to k. This can be clearly seen in Figure 8.25.

EXERCISES

8.1 **(a)** In Example 8.2, a 2-bit burst error (0110000) is assumed to have occurred in the second frame (frame 2). Using the (22, 13) 3-bit burst error correcting and 4-bit burst error detecting Fire code indicated in this example, calculate $\mathbf{H}_{10}^{\dagger} \cdot S$, and $\mathbf{B}_{10}^{\dagger} \cdot S$ for each frame, and then explain how this error can be corrected.

(b) For the 5-bit burst error (010011) occurred in the frame 2, explain how this error can be detected.

(c) Design the parallel decoding circuit of the (22, 13) 3-bit burst error correcting and 4-bit burst error detecting Fire code in Example 8.2, implemented by the combinational circuits.

8.2 For word length N and frame length L, prove that the number of frames m overlapped with adjacent ones by length z is expressed as

$$m = \left\lceil \frac{N - z}{L - z} \right\rceil,$$

where $\lceil x \rceil$ represents the smallest integer greater than or equal to x.

8.3 Prove Lemma 8.2.

8.4 Prove Theorem 8.3.

8.5 For the (42, 33) cyclic 3-bit burst error correcting and 4-bit burst error detecting Fire code generated by $\mathbf{g}(x) = (x^6 + 1)(x^3 + x + 1)$ over $GF(2)$, answer the following:

(a) Find the binary 9×9 companion matrix **T**.

(b) Express the parity-check matrix **H** of this (42, 33) code over $GF(2)$.

(c) Using the **H**, write the matrices of $\mathbf{T}^0$, $\mathbf{T}^7$, $\mathbf{T}^{14}$, $\mathbf{T}^{21}$, $\mathbf{T}^{28}$, and $\mathbf{T}^{35}$ necessary for parallel decoding.

(d) Design the parallel decoding circuit of this (42, 33) code.

(e) Suppose that the syndrome generated by a received word is $S = (011011111)^T$. Show that frame 2 is corrupted by an error pattern (001110000).

(f) Suppose that the syndrome generated by a received word is $S = (011111010)^T$. Show that none of the frames (from frame 0 to frame 5) results in a correctable 3-bit burst error pattern.

8.6 The 2-input XOR function is expressed as $F = A \oplus B = (A + B) \cdot (\overline{A \cdot B})$ with OR, NAND, and AND gates. Explain how a glitch may be generated when the inputs (A, B) change from (0, 1) to (1, 0).

8.7 Construct an 8-input XOR circuit using seven 2-input XOR gates in a perfect binary tree, and obtain the TAG frequency table for all transient weight of the circuit. Use the transient behavior of the XOR functions shown in Table 8.2 to determine the worst-case TAG number of this 8-input XOR circuit when all inputs receive transient T's.

8.8 For the syndrome generation circuit of Hsiao's (13, 8) SEC-DED code shown in Eq. (8.6), answer the following questions:

(a) For the received 9 inputs of $d'_0(N)$, $d'_1(N)$, $d'_2(N)$, $d'_3(T)$, $d'_4(N)$, $d'_5(N)$, $d'_6(N)$, $d'_7(N)$, and $c'_0(T)$ (i.e., WT = 2), indicate the T and G marks at the inputs and the outputs of each XOR gate in the syndrome S_0 generation circuit.

(b) In the above S_0 circuit, count the temporal accumulated glitches (TAGs).

(c) There are $\binom{9}{2} = 36$ transient patterns of WT = 2. Count among them the frequency for TAG = 4.

(d) Using Eqs. (8.12) and (8.13), find the maximum TAGs of this S_0 circuit.

(e) Design the gate-shared syndrome generation circuit of the code, and discuss the difference of TAG probabilities of the gate-shared circuit and the nonshared circuit.

REFERENCES

[BAI01] G. Bai, S. Bobba, and I. N. Hajj, "Static Timing Analysis Including Power Supply Noise Effect on Propagation Delay in VLSI Circuits," *Proc. Design Automation Conf.* (June 2001): 295–300.

[BENI00] L. Benini, G. D. Micheli, A. Macii, M. Poncino, and R. Scarsi, "Glitch Power Minimization by Selective Gate Freezing," *IEEE Trans. VLSI Syst.*, 8 (June 2000): 287–298.

[CHEN97] H. H. Chen and D. D. Ling, "Power Supply Noise Analysis Methodology for Deep-submicron VLSI Chip Design," *Proc. IEEE Design Automation Conf.* (June 1997): 638–643.

[CHIE69] R. T. Chien, "Burst-Correcting Codes with High-Speed Decoding," *IEEE Trans. Info. Theory*, IT-15 (January 1969): 109–113.

[ELSP62] B. Elspas and R. A. Short, "A Note on Optimum Burst-Error-Correcting Codes," *IRE Trans. Info. Theory*, IT-9 (January 1962): 39–42.

[FUJI02] E. Fujiwara, K. Namba, and M. Kitakami, "Parallel Decoding for Burst Error Control Codes" (in Japanese), *Trans. IEICE Japan*, J85-A (November 2002): 1284–1295. (Translated into English in: *Electron. Commun. in Japan*, 87 (January 2004): 38–48).

[GHOS04] S. Ghosh, S. Basu, and N. A. Touba, "Reducing Power Consumption in Memory ECC Checkers," *Proc. IEEE Int. Test Conf.* (October 2004).

[HAMA97] M. Hamada and E. Fujiwara, ''A Class of Error Control Codes for Byte Organized Memory Systems—SbEC-(Sb + S)ED Codes—,'' *IEEE Trans. Comput.*, 46 (January 1997): 105–109.

[HONG72] S. J. Hong and A. M. Patel, ''A General Class of Maximal Codes for Computer Applications,'' *IEEE Trans. Comput.*, C-21 (December 1972): 1322–1331.

[JIAN00] Y.-M. Jiang, A. Krstic, and K.-T. Cheng, ''Dynamic Timing Analysis Considering Power Supply Noise Effects,'' *Proc. Int. Symp. on Quality of Electronic Design* (March 2000): 137–144.

[KASA62a] T. Kasami, ''Systematic Cyclic Codes for Burst Error Correction'' (in Japanese), *J. IEICE*, 45 (January 1962): 9–16.

[KASA62b] T. Kasami, ''Burst Error Correcting Codes Constructed by Computer'' (in Japanese), *J. IECE Japan*, 45 (November 1962): 1527–1532.

[LAVA93] L. Lavagno and A. Sangiovanni-Vincentelli, *Algorithms for Synthesis and Testing of Asynchronous Circuits*, Kluwer Academic (1993).

[LO05] J. C. Lo, Y. L. Wan, and E. Fujiwara, ''Transient Behavior of the Encoding / Decoding Circuits of Error Control Codes,'' *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems* (October 2005): 120–128.

[MEGG61] J. E. Meggitt, ''Error Correcting Codes and Their Implementation for Data Transmission Systems,'' *IRE Trans. Info. Theory*, IT-7 (1961): 232–244.

[MEHT95] H. Mehta, M. Borah, R. M. Owens, and M. J. Irwin, ''Accurate Estimation of Combinational Circuit Activity,'' *Proc. IEEE Design Automation Conf.* (June 1995): 618–622.

[NISH97] H. Nishihara and S. Ura, *Introduction to Optical Electronics* (in Japanese), Corona Publishing (1997).

[PARR01] P. Parra, A. J. Acosta, and M. Valencia, ''Reduction of Switching Noise in Digital CMOS Circuits by Pin Swapping of Library Cells,'' *Proc. Int. Workshop—Power and Timing Modeling, Optimization and Simulation* (September 2001): paper 9.3.

[PETE72] W. W. Peterson and E. J. Weldon Jr., *Error-Correcting Codes*, 2d ed., MIT Press (1972).

[ROSS04] D. Rossi, A Muccio, A. K. Nieuwland, A. Katoch, and C. Metra, ''Impact of ECCs on Simultaneous Switching Output Noise for On-Chip Buses of High Reliability Systems,'' *Proc. IEEE Int. Online Test Symp.* (July 2004): 135–140.

[ROY99] Y. Y. ans K. Roy and R. Drechsler, ''Power Consumption in XOR-Based Circuits,'' *Proc. Asian and South Pacific Design Automation Conf.* (January 1999): 299–302.

[TANG02] K. T. Tang and E. G. Friedman, ''Simultaneous Switching Noise in On-Chip CMOS Power Distribution Networks,'' *IEEE Trans. VLSI Syst.*, 10, 4 (August 2002): 487–493.

[UMAN03] G. Umanesan and E. Fujiwara, ''Parallel Decoding Burst Error Correcting Codes,'' *Proc. IEEE Int. Symp. on Information Theory*, (July 2003): 420.

[UMAN05] G. Umanesan and E. Fujiwara, ''Parallel Decoding Cyclic Burst Error Correcting Codes,'' *IEEE Trans. Comput.*, 54 (January 2005): 87–92.

[ZHOU00] H. Zhou and D. F. Wong, ''Optimal Low Power XOR Gate Decomposition,'' *Proc. IEEE Design Automation Conf.* (June 2000): 104–107.

CONTENTS

9

Codes for Error Location: Error Locating Codes

Almost all the error control code functions are error correction and error detection. Another important error control function that lies midway between the functions of error correction and error detection is error location.

The codeword of error locating codes can be regarded as consisting of mutually exclusive blocks. The codes indicate which blocks are in error, without providing a precise determination of the erroneous digit position within each block. This type of code was originally proposed for use in an efficient retransmission of the word in communication systems, without whole words being retransmitted [WOLF63]. In dependable computer systems, on the other hand, this has a powerful potential for fault isolation and reconfiguration. In byte-organized systems the code could effectively locate the erroneous block containing faulty bytes and replace it by a spare one [FUJI94, KITA95].

The first attempt at an error locating code was by J. K. Wolf and B. Elspas in 1963 [WOLF63], but since then there have been only a few papers dedicated to the idea presented in journals and conference proceedings [WOLF65a, WOLF65b, CHAN65, GOET67, VAID92]. Recently To and Sakaniwa [TO89] have proposed an error discriminating code that covers a broad range of code functions including error location as well as error correction and detection.

This chapter covers the error locating codes for high-speed computer systems. Among the topics discussed are single-byte error locating codes [FUJI94], single-bit error correcting and single faulty package / chip locating codes [KITA95], burst error locating codes [KITA05], and also the necessary and sufficient conditions of the error locating codes [KITA97].

9.1 ERROR LOCATION OF FAULTY PACKAGES AND FAULTY CHIPS

As was shown in Figure 1.11 in Subsection 1.4.1, error location falls midway between the functions of error correction and error detection. In the codes designed by Wolf and Elspas,

Code Design for Dependable Systems: Theory and Practical Applications, by Eiji Fujiwara

the codeword is divided into p distinct bytes, each having b-bit length. The code detects $e(< b)$ or fewer errors, all occurring within a single byte and identifies that byte. For this reason the code is referred to as the *Single e-bit (within a b-bit byte) Error Locating code*, or $S_{e/b}EL$ code. For instance, if we let $\mathbf{E}_i(\mathbf{E}_j)$ be the set of e or fewer errors occurring within the $i(j)$-th byte, the code must satisfy the relation

$$E_1 \cdot \mathbf{H}^T \neq E_2 \cdot \mathbf{H}^T \neq \mathbf{0} \quad \text{for all } E_1 \in \mathbf{E}_i, \quad \text{and} \quad \text{for all } E_2 \in \mathbf{E}_j, i \neq j.$$

The number of check bits r is bounded from below by

$$r \geq \log_2 \left\{ 1 + p \sum_{i=1}^{\lceil e/2 \rceil} \binom{b}{i} \right\}, \tag{9.1}$$

where $\lceil x \rceil$ is the smallest integer not less than x.

In general, the error locating code is derived from the tensor product of the parity-check matrices [WOLF65a].

Definition 9.1 Let the $\mathbf{X} = (x_{i,j})$ and $\mathbf{Y} = (y_{i,j})$ matrices be an $a \times b$ matrix and a $c \times d$ matrix, respectively. The matrix $\mathbf{Z}$, defined as the *tensor product* of $\mathbf{X}$ and $\mathbf{Y}$, is the $ac \times bd$ matrix given by

$$\mathbf{Z} = \mathbf{X} \otimes \mathbf{Y} = \begin{bmatrix} x_{1,1}\mathbf{Y} & \cdots & x_{1,b}\mathbf{Y} \\ \vdots & \ddots & \vdots \\ x_{a,1}\mathbf{Y} & \cdots & x_{a,b}\mathbf{Y} \end{bmatrix}.$$

□

Let $\mathbf{H}_D$ be the $\rho \times b$ parity-check matrix for a binary $(b, b - \rho)$ linear code $\boldsymbol{C}_D$ that detects the class of errors E_D. Let $\mathbf{H}_C$ be the $m \times p$ parity-check matrix for a nonbinary $(p, p - m)$ linear code $\boldsymbol{C}_C$, with symbols from $GF(2^\rho)$, that corrects the class of errors E_C. Here, a column vector with ρ-bit length in the parity-check matrix of $\boldsymbol{C}_D$ corresponds to a symbol in $\boldsymbol{C}_C$, meaning a symbol from $GF(2^\rho)$. Finally, let $\boldsymbol{C}$ be the binary $(pb, pb - \rho m)$ linear code with the $\rho m \times pb$ parity-check matrix $\mathbf{H}$ given by

$$\mathbf{H} = \mathbf{H}_C \otimes \mathbf{H}_D.$$

Theorem 9.1 *If all binary byte errors corresponding to the erroneous bytes are within class E_D, and if the erroneous bytes form a pattern of errors over $GF(2^\rho)$ that falls in class E_C, then code $\boldsymbol{C}$ detects the errors and identifies the erroneous bytes.*

If $\boldsymbol{C}_D$ is an e-bit (within a b-bit byte) error detecting code with ρ check bits and $\boldsymbol{C}_C$ is a single-symbol error correcting code on $GF(2^\rho)$, then $\boldsymbol{C}$ is an $S_{e/b}EL$ code.

The codes described above only apply to errors having fewer than b bits. If the maximum number of errors located by the codes is equal to b, then the $S_{b/b}EL$ code is an SbEC code. This is shown in the following theorem [VAID92].

Theorem 9.2 *An error locating code that can locate all single-byte errors is a single-byte error correcting code.*

From the result above the existing error locating codes are not always suitable for application to byte-organized semiconductor memory systems.

In general, a semiconductor memory module has a hierarchical organization consisting of memory cards or memory packages on which memory chips are mounted. The memory card on which b-bit byte-organized RAM chips are mounted provides data output having B-bit length, where B is a multiple of b, meaning $B = p \times b$. The output of the clustered data from the package or card is called here a *block;* its code length is in B bits. The output of the clustered data from the chip is called a *byte;* its length is in b bits.

So we now have a new class of error locating codes that pertains to byte-organized systems. We introduce the term *block* to denote a set of bytes. Each codeword is divided into disjoint blocks, and the block is subdivided into bytes. This new class of codes will locate an erroneous block that contains a single-byte error. We can call these codes *single b-bit byte (within a B-bit block) error locating codes* (i.e., $S_{b/p\times b}$EL *codes*) or as *block error locating codes*. We will also use the terms *code length in bits*, *code length in bytes*, and *code length in blocks* to denote the lengths of a codeword in bits, bytes, and blocks, respectively. Figure 9.1 illustrates these relations.

The predominant errors, even in the byte-organized semiconductor memory chips, are soft errors induced by α particles and external noises. These errors are still apt to be manifest as single-bit errors in byte-organized RAM chips. Therefore an error locating code capable of correcting single-bit errors is very useful. We call these codes *single-bit error correcting and single b-bit byte (within a B-bit block) error locating codes* (i.e., SEC-$S_{b/p\times b}$EL *codes* [FUJI94]) or *block error locating codes with single-bit error correction capability.* In this regard, codes such as the $S_{b/p\times b}$EL codes and the SEC-$S_{b/p\times b}$EL codes discussed above can also be called *codes for locating the package / card with faulty chips*. Once the faulty package / card is located by a code, and the faulty package / card is replaced by a correct one, then the system can be recovered and proceed with normal operation. As for the location of the faulty chips, we depend on such codes as the *single-bit error correcting and single e-bit (within a b-bit byte) error locating codes*

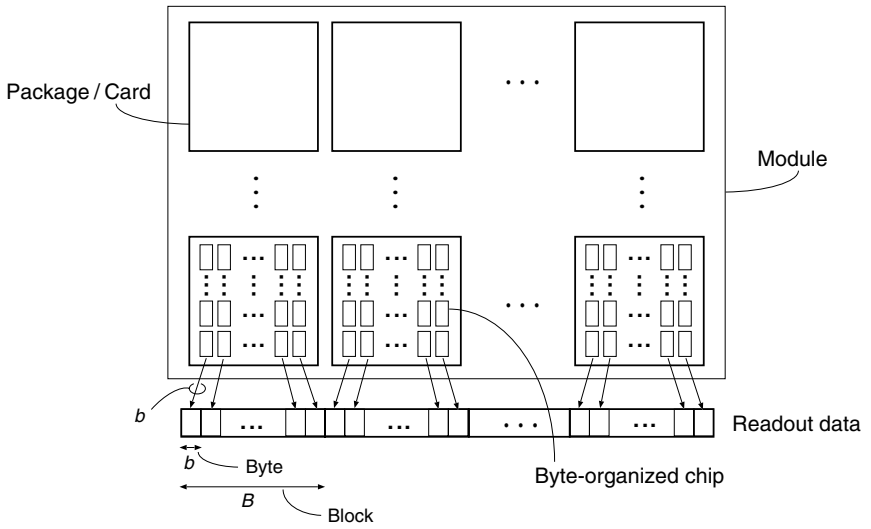


Figure 9.1 *Relationship between byte and block.*

(i.e., SEC-$S_{e/b}$EL *codes* [KITA95]) or *byte error locating codes with single-bit error correction capability*. This type of codes is called *codes for locating faulty chips*.

9.2 BLOCK ERROR LOCATING ($S_{b/p\times b}$EL) CODES

First, we study a class of block error locating codes that can locate an erroneous block containing a single-byte error. This is the $S_{b/p\times b}$EL *codes*, where $B = p \times b$, that locate a single b-bit byte within a B-bit block.

It is simple to construct the $S_{b/p\times b}$EL codes by using SbEC codes. This is shown in the following theorem.

Theorem 9.3 *Let the following matrix $\boldsymbol{H}'$ be a parity-check matrix of the SbEC code:*

$$\boldsymbol{H}' = \left[\begin{array}{c|c|c|c} \overleftarrow{\boldsymbol{H}_0} & \overleftarrow{\boldsymbol{H}_1} & \cdots & \overleftarrow{\boldsymbol{H}_{n'-1}} \end{array}\right],$$

where n' is the code length (in bytes) of the code and $\boldsymbol{H}_i$, $i = 0, 1, \cdots, n' - 1$, is the linearly independent column with rank b corresponding to the i-th byte. Then the code described by the following matrix $\boldsymbol{H}$ is an $S_{b/p\times b}$EL code:

$$\boldsymbol{H} = \left[\begin{array}{c|c|c|c} \overleftarrow{\boldsymbol{H}_0 \cdots \boldsymbol{H}_0} & \overleftarrow{\boldsymbol{H}_1 \cdots \boldsymbol{H}_1} & \cdots & \overleftarrow{\boldsymbol{H}_{n'-1} \cdots \boldsymbol{H}_{n'-1}} \end{array}\right],$$

where B is a multiple of b, meaning, $B = p \times b$.

This theorem can be easily proved because parity-check matrix of the $S_{b/p\times b}$EL codes is organized by p repetitions of $\mathbf{H}_i$ in the i-th block, $i = 0, 1, \cdots, n' - 1$, where $\mathbf{H}_i$ is the linearly independent column in $\mathbf{H}'$.

The maximal SbEC codes shown in Subsection 5.1.4 are used to express the code length (in bits) of the $S_{b/p\times b}$EL codes as follows:

$$N = B \cdot \frac{2^R - 1 - 2^b(2^c - 1)}{2^b - 1} + c \cdot \frac{B}{b},$$

where $R = br + c, 0 \leq c < b$, is the check-bit length of the $S_{b/p\times b}$EL codes and $B = p \times b$.

Theorem 9.3 leads to the following corollary.

Corollary 9.1 *Let the following matrix $\boldsymbol{H}'$ be a parity-check matrix of the $S_{b/p\times b}$EL code:*

$$\boldsymbol{H}' = \left[\begin{array}{c|c|c|c} \overleftarrow{\boldsymbol{H}_0} & \overleftarrow{\boldsymbol{H}_1} & \cdots & \overleftarrow{\boldsymbol{H}_{n'-1}} \end{array}\right],$$

where $B = p \times b$, n' is the code length (in blocks) of the code and $\boldsymbol{H}_i$, $i = 0,\ 1,\ \ldots,\ n' - 1$, is the submatrix corresponding to the i-th block. Then the code described by the following

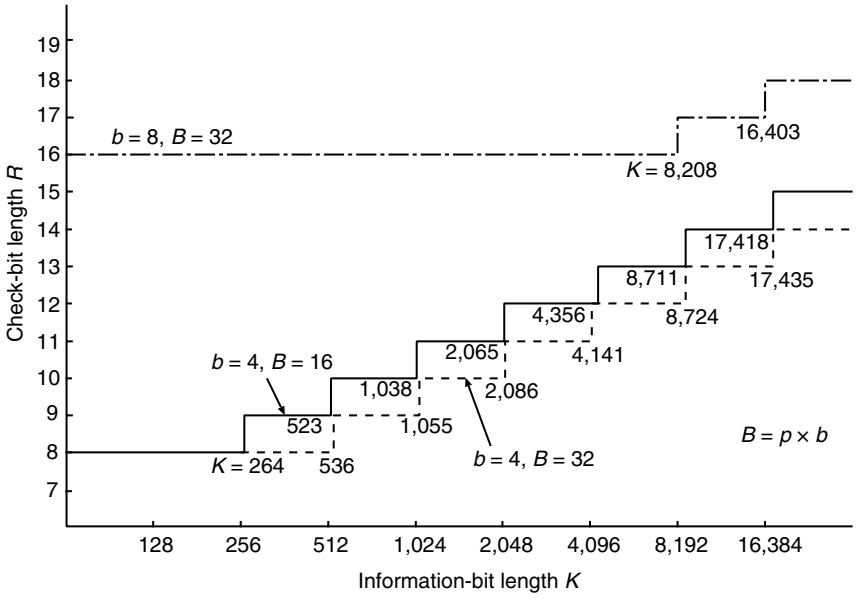


Figure 9.2 *Check-bit lengths compared with information-bit lengths of the $S_{b/p\times b}$EL codes.* Source: *[FUJI94].* © *1994 IEEE.*

matrix $\boldsymbol{H}$ *is an* $S_{b/B'}EL$ *code:*

$$\boldsymbol{H} = \Big[\underbrace{\boldsymbol{H}_0 \cdots \boldsymbol{H}_0}_{\longleftarrow B' \longrightarrow} \Big| \underbrace{\boldsymbol{H}_1 \cdots \boldsymbol{H}_1}_{\longleftarrow B' \longrightarrow} \Big| \cdots \Big| \underbrace{\boldsymbol{H}_{n'-1} \cdots \boldsymbol{H}_{n'-1}}_{\longleftarrow B' \longrightarrow}\Big],$$

where B' *is a multiple of B.*

This corollary can be easily proved in the same way as the theorem above.

Figure 9.2 shows the relation between the information-bit lengths and the check-bit lengths of the $S_{b/p\times b}$EL codes for the cases of $(b, B) = (4, 16),\ (4, 32)$, and $(8, 32)$.

9.3 SINGLE-BIT ERROR CORRECTING AND SINGLE-BLOCK ERROR LOCATING (SEC-$S_{b/p\times b}$EL) CODES

This section deals with the *SEC-$S_{b/p\times b}EL$ codes*, where $B = p \times b$ that correct single-bit errors and locate single b-bit byte errors within a B-bit block.

9.3.1 Code Conditions and Bounds

Necessary and Sufficient Conditions Let $\mathbf{E}_s$ be the error set consisting of all single-bit errors, and let $\mathbf{E}_i(\mathbf{E}_j)$ be the error set consisting of all single-byte errors in the $i(j)$-th block excluding single-bit errors. Thus $\mathbf{E}_i \cap \mathbf{E}_s = \mathbf{E}_j \cap \mathbf{E}_s = \phi$ for all $i \neq j$, where ϕ is the empty set. The following theorem describes necessary and sufficient conditions that characterize SEC-$S_{b/p\times b}$EL codes.

Theorem 9.4 *A linear code, described by the parity-check matrix* $\boldsymbol{H}$*, corrects all errors in* $\boldsymbol{E}_s$ *and locates all errors in* $\boldsymbol{E}_i$*, or* $\boldsymbol{E}_j$*, where* $i \neq j$*, if and only if*

1. $E \cdot \boldsymbol{H}^T \neq 0$ *for all* $E \in \{\boldsymbol{E}_s \cup \boldsymbol{E}_i \cup \boldsymbol{E}_j\}$,
2. $E_1 \cdot \boldsymbol{H}^T \neq E_2 \cdot \boldsymbol{H}^T$ *for all* $E_1, E_2 \in \boldsymbol{E}_s, E_1 \neq E_2$,
3. $E_3 \cdot \boldsymbol{H}^T \neq E_4 \cdot \boldsymbol{H}^T$ *for all* $E_3 \in \boldsymbol{E}_i$*, and for all* $E_4 \in \boldsymbol{E}_j$,
4. $E_1 \cdot \boldsymbol{H}^T \neq E_3 \cdot \boldsymbol{H}^T$ *for all* $E_1 \in \boldsymbol{E}_s$*, and for all* $E_3 \in \boldsymbol{E}_i$.

Proof It is apparent that conditions 1 and 2, and conditions 1 and 3 are necessary conditions for correcting all single-bit errors and for indicating the location of an erroneous block containing single-byte errors, respectively. Conditions 1 and 4 are also necessary conditions for distinguishing single-bit errors from a single-byte error excluding single-bit errors. So the SEC-$S_{b/p\times b}$EL codes must satisfy all the conditions 1 to 4.

Conversely, if a code satisfies conditions 1 to 4, then we can distinguish single-bit errors from single-byte errors. We can also correct all single-bit errors and locate all single-byte errors. Therefore this code is an SEC-$S_{b/p\times b}$EL code. Q.E.D.

Bounds

Theorem 9.5 *Linear* $(N, N-R)$ *SEC-*$S_{b/p\times b}$*EL codes satisfy*

$$R \geq 2b.$$

Proof For two bytes each belonging to different blocks, there exist $2b$ column vectors in the parity-check matrix on $GF(2)$. These vectors should be linearly independent by condition 3 of Theorem 9.4, and therefore the number of rows of the parity-check matrix should be $2b$ or more. Q.E.D.

Theorem 9.6 *Linear* $(N, N-R)$ *SEC* $-$ $S_{b/p\times b}$*EL codes satisfy*

$$N \leq \frac{B(2^R - 1)}{B + 2^b - b - 1}, \tag{9.2}$$

where $B = p \times b$.

Proof In the SEC-$S_{b/p\times b}$EL codes, in general, the syndromes caused by single-bit errors should be different from each other, and those caused by single-byte errors excluding single-bit errors should be different from the ones caused by single-bit errors. Therefore the errors in one block have at least $B + 2^b - b - 1$ different syndromes, and hence there are at least $\frac{N}{B}(B + 2^b - b - 1)$ different syndromes in the received word. Hence the following inequality is satisfied:

$$2^R \geq \frac{N}{B}(B + 2^b - b - 1) + 1.$$

From this, the inequality (9.2) can be deduced. Q.E.D.

9.3.2 Design for SEC-$S_{b/p\times b}$EL Codes

1. Codes Designed by Tensor Product — Codes I —

In general, we can design the error locating codes by means of the tensor product of two codes, one being an error correcting code and the other an error detecting code. The codes designed by this method are called here type I codes. This method can be applied to the design of SEC-$S_{b/p\times b}$EL codes by using the single-bit error correcting and single b-bit byte error detecting code, or the SEC-SbED code presented in Section 6.1, and a single b-bit byte error correcting code, or an SbEC code presented in Section 5.1.

Theorem 9.7 *The code described by the following matrix $\boldsymbol{H}$ is an SEC − $S_{b/p\times b}$EL code:*

$$\begin{aligned}\boldsymbol{H} &= \boldsymbol{H}'_{b'} \otimes \boldsymbol{H}''_b \\ &= \left[\boldsymbol{H}'_0 \;\middle|\; \boldsymbol{H}'_1 \;\middle|\; \cdots \;\middle|\; \boldsymbol{H}'_{(N/B)-1}\right] \otimes \boldsymbol{H}''_b \\ &= \left[\boldsymbol{H}'_0 \cdot \boldsymbol{H}''_b \;\middle|\; \boldsymbol{H}'_1 \cdot \boldsymbol{H}''_b \;\middle|\; \cdots \;\middle|\; \boldsymbol{H}'_{(N/B)-1} \cdot \boldsymbol{H}''_b\right] \\ &= \left[\boldsymbol{H}_0 \;\middle|\; \boldsymbol{H}_1 \;\middle|\; \cdots \;\middle|\; \boldsymbol{H}_{(N/B)-1}\right],\end{aligned}$$

where $\otimes$ represents tensor product, $B = p \times b$, N is the code length (in bits) of the SEC-$S_{b/p\times b}$EL code, $\boldsymbol{H}'_{b'}$ is the parity-check matrix of the Sb'EC code, $\boldsymbol{H}''_b$ is the parity-check matrix of the $(B, B - b')$ SEC-SbED codes, and $\boldsymbol{H}'_i$ is the submatrix of $\boldsymbol{H}'_{b'}$ corresponding to the i-th byte.

Proof It is apparent that the code satisfies condition 1 of Theorem 9.4 for any single-bit errors and any single-byte errors. Because the binary columns of $\mathbf{H}$ are distinct, condition 2 of Theorem 9.4 is satisfied. The syndrome resulting from any single-byte error in the i-th block is different from that in the j-th block for $i \neq j$ because each column in $\mathbf{H}_i$ is determined by the product of $\mathbf{H}'_i$ and $\mathbf{H}''_b$. Hence condition 3 is satisfied. In general, every $\mathbf{H}'_i$ includes $b' \times b'$ identity matrix, meaning $\mathbf{I}_{b'}$, and therefore every $\mathbf{H}_i$ has $\mathbf{H}''_b$ as a column element. This implies that the syndrome resulting from any single-bit error is different from that resulting from any single-byte error excluding single-bit errors. Based on this and on condition 3, condition 4 in Theorem 9.4 is satisfied. From Theorem 9.4 it follows that the code described by $\mathbf{H}$ is an SEC-$S_{b/p\times b}$EL code. Q.E.D.

Example 9.1 [FUJI94]

For $b = 4$ and $b' = 5$ the S5EC code with $r = 2$ described by the matrix $\mathbf{H}'_{b'}$ is

$$\begin{aligned}\mathbf{H}'_b &= \left[\begin{array}{c|c|c|c|c|c|c} \mathbf{H}'_0 & \mathbf{H}'_1 & \mathbf{H}'_2 & \mathbf{H}'_3 & \cdots & \mathbf{H}'_{32} \end{array}\right] \\ &= \left[\begin{array}{c|c|c|c|c|c} \mathbf{I}_5 & \mathbf{O}_5 & \mathbf{I}_5 & \mathbf{I}_5 & & \mathbf{I}_5 \\ \mathbf{O}_5 & \mathbf{I}_5 & \mathbf{I}_5 & \mathbf{T}_5 & \cdots & \mathbf{T}_5^{30} \end{array}\right],\end{aligned}$$

where $\mathbf{T}_5$ is a primitive element in GF(2^5), and $\mathbf{O}_5$ and $\mathbf{I}_5$ are the zero element and identity element in GF(2^5), respectively. Let $\mathbf{H}''_b$ be the parity-check matrix of the $(12, 7)$ SEC-S4ED code having $b' = 5$ check bits. With these two codes the (396, 386)

SEC-$S_{4/3\times4}$EL code obtained is shown in the following matrix $\mathbf{H}$:

$$\begin{aligned}\mathbf{H} &= \mathbf{H}'_{b'} \otimes \mathbf{H}''_b \\ &= \left[\begin{array}{c|c|c|c|c|c} \mathbf{I}_5 & \mathbf{O}_5 & \mathbf{I}_5 & \mathbf{I}_5 & \cdots & \mathbf{I}_5 \\ \mathbf{O}_5 & \mathbf{I}_5 & \mathbf{I}_5 & \mathbf{T}_5 & \cdots & \mathbf{T}_5^{30} \end{array}\right] \otimes \mathbf{H}''_b \\ &= \left[\begin{array}{c|c|c|c|c|c} \mathbf{H}''_b & \mathbf{O} & \mathbf{H}''_b & \mathbf{H}''_b & \cdots & \mathbf{H}''_b \\ \mathbf{O} & \mathbf{H}''_b & \mathbf{H}''_b & \mathbf{T}_5 \cdot \mathbf{H}''_b & \cdots & \mathbf{T}_5^{30} \cdot \mathbf{H}''_b \end{array}\right].\end{aligned}$$

The code length (in bits) of the SEC-$S_{b/p\times b}$EL codes, defined by Theorem 9.7, can be expressed as follows. In this case the maximal codes shown in Subsection 5.1.4 are used to determine the length of the Sb'EC codes.

$$N = b(2^{b'-b+1} - 1)\left(\frac{2^R - 1 - 2^{b'}(2^c - 1)}{2^{b'} - 1} - 1\right) + b(2^{b'+c-b+1} - 1) \qquad (9.3)$$

In this equation, $R = b'r + c$, $0 \le c < b'$, is the check-bit length of the SEC-$S_{b/p\times b}$EL codes.

Figure 9.3 shows the relations between the information-bit lengths and the check-bit lengths of the SEC-$S_{b/p\times b}$EL codes for $b = 4$ bits. In this case, B shows the maximum block length in bits determined by the value of $b'(> b)$.

2. Codes Designed by Odd / Even-Weight Column Square Matrices — Codes II —

Here the SEC-$S_{b/p\times b}$EL codes designed by another method are presented and called type II codes.

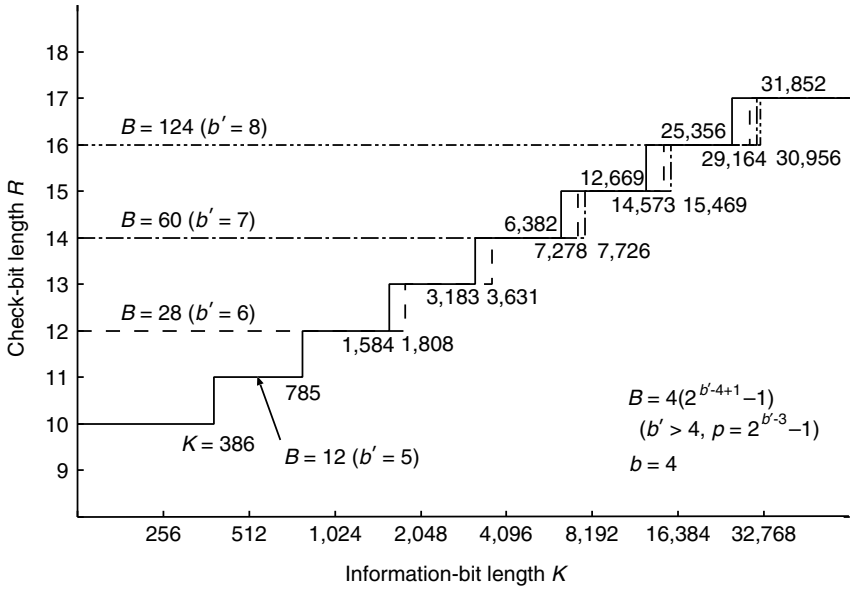


Figure 9.3 *Check-bit lengths compared with information-bit lengths of the SEC-$S_{4/p\times4}$EL type I codes.* Source: *[FUJI94]. © 1994 IEEE.*

Preliminaries

Definition 9.2 Let an *odd-weight column square matrix* be a nonsingular $b \times b$ matrix whose columns are odd weight. Let an *even-weight column square matrix* be a $b \times b$ matrix whose columns are b copies of an even-weight vector (including the zero vector). □

Because there are 2^{b-1} even-weight column vectors having dimension b, there exist 2^{b-1} even-weight column square matrices.

Here, we show an example design method of nonsingular odd-weight column $b \times b$ square matrices.

Definition 9.3 The matrix $\mathbf{M}_b$ shown below is defined as the matrix having b rows and 2^{b-1} odd-weight columns:

$$\mathbf{M}_b = \begin{bmatrix} \boldsymbol{a}_0 & \boldsymbol{a}_1 & \boldsymbol{a}_2 & \cdots & \boldsymbol{a}_{2^{b-1}-1} \end{bmatrix} = \left[\begin{array}{cccccc} p_0 & p_1 & p_2 & \cdots & p_{2^{b-1}-1} \\ \hline | & | & | & & | \\ 0 & \alpha^0 & \alpha & \cdots & \alpha^{2^{b-1}-2} \\ | & | & | & & | \end{array}\right]_{b\times 2^{b-1}}.$$

In $\mathbf{M}_b$, α is a root of the $(b-1)$-th degree binary primitive polynomial $\mathbf{g}(x)$, $\begin{vmatrix} \\ \alpha^i \\ \\ \end{vmatrix}$ is a coefficient vector of $x^i \bmod \mathbf{g}(x)$, and $p_i \in \{0,1\}$ is a bit determined to make the column vector $\boldsymbol{a}_i$, $i = 0, 1, \cdots, 2^{b-1}-1$, be odd weight. □

Lemma 9.1 *The following shows a nonsingular odd-weight column square matrix generated from any consecutive b column vectors in the matrix $\boldsymbol{M}_b$:*

$$\boldsymbol{A}_i = \begin{bmatrix} \boldsymbol{a}_{i_0} & \boldsymbol{a}_{i_1} & \cdots & \boldsymbol{a}_{i_{b-1}} \end{bmatrix}_{b\times b},$$

where $i_j \equiv i + j \bmod 2^{b-1}$, and $0 \le j \le b-1$.

From this lemma we obtain 2^{b-1} nonsingular odd-weight column square matrices.

Design of the Parity-Check Matrices Let $\mathbf{A}$ and $\mathbf{B}$ be the sets of the odd-weight column square matrices and the even-weight column square matrices, respectively. The following lemma provides the basic idea of the code design method.

Lemma 9.2 *Consider two different vectors each having degree r (i.e., $\boldsymbol{Q}$ and $\boldsymbol{Q}'$) and each being constructed of elements from the even-weight column square matrices and the odd-weight column square matrices (i.e., $\boldsymbol{Q}_j, \boldsymbol{Q}'_j \in \boldsymbol{A} \cup \boldsymbol{B}$ for $j = 0, 1, \cdots, r-1$). In each vector there exists at least one matrix included in $\mathbf{A}$.*

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_0 \\ \boldsymbol{Q}_1 \\ \vdots \\ \boldsymbol{Q}_{r-1} \end{bmatrix}, \quad \boldsymbol{Q}' = \begin{bmatrix} \boldsymbol{Q}'_0 \\ \boldsymbol{Q}'_1 \\ \vdots \\ \boldsymbol{Q}'_{r-1} \end{bmatrix}.$$

Assume that the i-th elements in $\boldsymbol{Q}$ *and* $\boldsymbol{Q}'$ *(i.e.,* $\boldsymbol{Q}_i$ *and* $\boldsymbol{Q}'_i$*), respectively, are different square matrices. In other words,* $\boldsymbol{Q}_i$ *is an odd-weight column square matrix and* $\boldsymbol{Q}'_i$ *is an even-weight column square matrix, and vice versa. Then any summation of binary column vectors in* $\boldsymbol{Q}$ *produces a different result than that given by any summation of binary column vectors in* $\boldsymbol{Q}'$.

Proof Let $\mathbf{V}$ and $\mathbf{V}'$, each having r b-tuples, be the results of the summation of column vectors in $\mathbf{Q}$ and $\mathbf{Q}'$, respectively:

$$\mathbf{V} = \begin{bmatrix} \boldsymbol{v}_0 \\ \boldsymbol{v}_1 \\ \vdots \\ \boldsymbol{v}_{r-1} \end{bmatrix}, \quad \mathbf{V}' = \begin{bmatrix} \boldsymbol{v}'_0 \\ \boldsymbol{v}'_1 \\ \vdots \\ \boldsymbol{v}'_{r-1} \end{bmatrix}.$$

Without loss of generality, $\mathbf{Q}_i$ and $\mathbf{Q}'_i$ can be regarded as the odd-weight column square matrix and the even-weight column square matrix, respectively. Assume $\boldsymbol{v}_i = \boldsymbol{v}'_i$. Then $\mathbf{V}$ is the vector resulting from the summation of an even number of columns in $\mathbf{Q}$, and $\mathbf{V}'$ is that resulting from the summation of odd number of columns in $\mathbf{Q}'$. There exists an odd-weight column matrix in $\mathbf{Q}'$, say $\mathbf{Q}'_l$, in the l-th row for $l \neq i$. So $\boldsymbol{v}'_l$ is an odd-weight b-tuple and $\boldsymbol{v}_l$ has even weight. Therefore, $\mathbf{V} \neq \mathbf{V}'$. Q.E.D.

Below we provide an example expressed as a submatrix $\mathbf{H}_i$ corresponding to the i-th block. In other words, if we use the matrix from $\mathbf{B}$ in the first row, then we write $\mathcal{B}$ in this place, and so on:

$$\mathbf{H}_i = \begin{bmatrix} \mathcal{B} \\ \mathcal{A} \\ \mathcal{A} \\ \vdots \end{bmatrix}.$$

In order to distinguish single-bit errors from single-byte errors, excluding single-bit errors, every submatrix has at least one $b \times b$ identity matrix, which is included in $\mathbf{A}$. In the submatrix $\mathbf{H}_i$, for example, this can be expressed as follows:

$$\mathbf{H}_i = \begin{bmatrix} \mathcal{B} \\ \mathcal{A} \\ \mathcal{A} \\ \vdots \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{B}_0 & \mathbf{B}_1 & \cdots & \mathbf{B}_{2^{b-1}-1} & \mathbf{B}_0 & \mathbf{B}_1 & \cdots & \mathbf{B}_{2^{b-1}-1} & \cdots & \mathbf{B}_0 & \mathbf{B}_1 & \cdots & \mathbf{B}_{2^{b-1}-1} & \cdots \\ \mathbf{I}_b & \mathbf{I}_b & \cdots & \mathbf{I}_b & \mathbf{I}_b & \mathbf{I}_b & \cdots & \mathbf{I}_b & \cdots & \mathbf{I}_b & \mathbf{I}_b & \cdots & \mathbf{I}_b & \cdots \\ \mathbf{A}_0 & \mathbf{A}_0 & \cdots & \mathbf{A}_0 & \mathbf{A}_1 & \mathbf{A}_1 & \cdots & \mathbf{A}_1 & \cdots & \mathbf{A}_{2^{b-1}-1} & \mathbf{A}_{2^{b-1}-1} & \cdots & \mathbf{A}_{2^{b-1}-1} & \cdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots \end{bmatrix},$$

where $\mathbf{A}_i \in \mathbf{A}$, $\mathbf{B}_i \in \mathbf{B}$, $0 \leq i \leq 2^{b-1} - 1$, and $\mathbf{I}_b = \mathbf{A}_0$ is a $b \times b$ identity matrix.

Theorem 9.8 *The code described by the following matrix* $\boldsymbol{H}$ *is an SEC-*$S_{b/p\times b}$*EL code with check-bit length* $R = br$ *and codeword length N bits:*

$$\boldsymbol{H} = \left[\begin{array}{c|c|c|c|c} \boldsymbol{H}_0 & \boldsymbol{H}_1 & \cdots & \boldsymbol{H}_{2^r-3} & \boldsymbol{H}_{2^r-2} \end{array}\right] = \left[\begin{array}{c|c|c|c|c} \mathcal{B} & \mathcal{B} & \cdots & \mathcal{A} & \mathcal{A} \\ \mathcal{B} & \mathcal{B} & \cdots & \mathcal{A} & \mathcal{A} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathcal{B} & \mathcal{A} & \cdots & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{B} & \cdots & \mathcal{B} & \mathcal{A} \end{array}\right]_{br\times N},$$

where $\mathcal{A}$ *and* $\mathcal{B}$ *are row vectors in which odd-weight square matrices and even-weight square matrices are used, respectively, and the top* $\mathcal{A}$ *in each column is the row vector including all* $b \times b$ *identity matrices. In this matrix each submatrix* $\boldsymbol{H}_i$ *includes different pattern of* $\mathcal{A}$*'s and* $\mathcal{B}$*'s and has at least one* $\mathcal{A}$.

The code length (in bits) of the code above is expressed by the following equation:

$$N = (2^r - 1)B = b(2^r - 1)2^{(b-1)(r-1)}, \tag{9.4}$$

where r is the number of check-bytes.

Proof It is apparent that the code satisfies condition 1 of Theorem 9.4 for any single-bit errors and any single-byte errors. The binary columns in $\mathbf{H}$ are distinct. Therefore condition 2 of Theorem 9.4 is satisfied as well. If $i \neq j$, then matrices $\mathbf{H}_i$ and $\mathbf{H}_j$ have different patterns of $\mathcal{A}$'s and $\mathcal{B}$'s. By Lemma 9.2, condition 3 of Theorem 9.4 is satisfied. Since every submatrix has at least one identity matrix, condition 4 of Theorem 9.4 is satisfied. Hence the code described by $\mathbf{H}$ is an SEC-$S_{b/p\times b}$EL code.

The number of blocks is equal to that of the nonzero integers expressed by r-digit binary numbers, meaning $2^r - 1$. The block length B is determined by the number of distinct column vectors using r matrices of the odd-weight column square matrices and the even-weight column square matrices. B is also determined by the condition that every $\mathbf{H}_i$ includes at least one row of identity matrices. Based on this, the maximum block length B bits can be expressed as $b(2^{b-1})^{r-1} = b2^{(b-1)(r-1)}$. Therefore Eq. (9.4) is valid. Q.E.D.

Example 9.2 [FUJI94]

For $b = 4$ and $R = 8$ the $(96, 88)$ SEC-$S_{4/8\times 4}$EL code is given by the following matrix $\mathbf{H}$:

$$\begin{aligned} \mathbf{H} &= \left[\begin{array}{c|c|c} \mathbf{H}_0 & \mathbf{H}_1 & \mathbf{H}_2 \end{array}\right] \\ &= \left[\begin{array}{c|c|c} \mathcal{B} & \mathcal{A} & \mathcal{A} \\ \mathcal{A} & \mathcal{B} & \mathcal{A} \end{array}\right] \\ &= \left[\begin{array}{cccc|cccc|cccc} \mathbf{B}_0 & \mathbf{B}_1 & \cdots & \mathbf{B}_7 & \mathbf{I}_4 & \mathbf{I}_4 & \cdots & \mathbf{I}_4 & \mathbf{I}_4 & \mathbf{I}_4 & \cdots & \mathbf{I}_4 \\ \mathbf{I}_4 & \mathbf{I}_4 & \cdots & \mathbf{I}_4 & \mathbf{B}_0 & \mathbf{B}_1 & \cdots & \mathbf{B}_7 & \mathbf{A}_0 & \mathbf{A}_1 & \cdots & \mathbf{A}_7 \end{array}\right]. \end{aligned}$$

Figure 9.4 shows this matrix expressed in binary form.

$$
\mathbf{H} = \left[\begin{array}{cccc}
0000\ 1111\ 0000\ 0000\ 1111\ 1111\ 0000\ 1111 & 1000\ 1000\ 1000\ 1000\ 1000\ 1000\ 1000\ 1000 & 1000\ 1000\ 1000\ 1000\ 1000\ 1000\ 1000\ 1000 \\
0000\ 1111\ 1111\ 0000\ 0000\ 0000\ 1111\ 1111 & 0100\ 0100\ 0100\ 0100\ 0100\ 0100\ 0100\ 0100 & 0100\ 0100\ 0100\ 0100\ 0100\ 0100\ 0100\ 0100 \\
0000\ 0000\ 1111\ 1111\ 0000\ 1111\ 0000\ 1111 & 0010\ 0010\ 0010\ 0010\ 0010\ 0010\ 0010\ 0010 & 0010\ 0010\ 0010\ 0010\ 0010\ 0010\ 0010\ 0010 \\
0000\ 0000\ 0000\ 1111\ 1111\ 0000\ 1111\ 1111 & 0001\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001 & 0001\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001 \\
\hline
1000\ 1000\ 1000\ 1000\ 1000\ 1000\ 1000\ 1000 & 0000\ 1111\ 0000\ 0000\ 1111\ 1111\ 0000\ 1111 & 1000\ 0001\ 0010\ 0101\ 1011\ 0111\ 1110\ 1100 \\
0100\ 0100\ 0100\ 0100\ 0100\ 0100\ 0100\ 0100 & 0000\ 1111\ 1111\ 0000\ 0000\ 0000\ 1111\ 1111 & 0100\ 1001\ 0011\ 0110\ 1101\ 1010\ 0101\ 1010 \\
0010\ 0010\ 0010\ 0010\ 0010\ 0010\ 0010\ 0010 & 0000\ 0000\ 1111\ 1111\ 0000\ 1111\ 0000\ 1111 & 0010\ 0101\ 1011\ 0111\ 1110\ 1100\ 1000\ 0001 \\
0001\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001 & 0000\ 0000\ 0000\ 1111\ 1111\ 0000\ 1111\ 1111 & 0001\ 0010\ 0101\ 1011\ 0111\ 1110\ 1100\ 1000
\end{array}\right]
$$

Figure 9.4 *Binary form of the (96, 88) SEC-$S_{4/8\times4}$EL codes.* Source : *[FUJI94]. © 1994 IEEE.*

Expanding the Code Length The code design method of Theorem 9.8 says that the check-bit length should be a multiple of the byte length b. This condition can be relaxed by taking any check-bit length $R > 2b$, as shown by the following theorem.

Theorem 9.9 *Let the following matrix $\boldsymbol{H}$ be a parity-check matrix of an SEC-$S_{b/p\times b}$EL code with a code length in bits $b(2^r - 1)2^{(b-1)(r-1)}$ and a check-bit length br:*

$$\boldsymbol{H} = \left[\boldsymbol{H}_0 \;\middle|\; \boldsymbol{H}_1 \;\middle|\; \cdots \;\middle|\; \boldsymbol{H}_{n-1}\right],$$

where $\boldsymbol{H}_i$, $i = 0, 1, \cdots, n-1$, is a submatrix of $\boldsymbol{H}$. The code described by the following matrix $\boldsymbol{H}'$ is an SEC-$S_{b/p\times b}$EL code with check-bit length $br + 1$:

$$\boldsymbol{H}' = \left[\begin{array}{c|c|c|c|c|c} \boldsymbol{H}_0 & \boldsymbol{H}_0 & \boldsymbol{H}_1 & \boldsymbol{H}_1 & \cdots & \boldsymbol{H}_{n-1} \;\; \boldsymbol{H}_{n-1} \\ 0\cdots 0 & 1\cdots 1 & 0\cdots 0 & 1\cdots 1 & & 0\cdots 0 \;\; 1\cdots 1 \end{array}\right].$$

If this procedure is performed c times on $\boldsymbol{H}$, then the code length in bits of the expanded code with check-bit length $R = br + c$, $0 \le c < b$ can be expressed as follows:

$$N = (2^r - 1)B = b(2^r - 1)2^{(b-1)(r-1)+c}. \tag{9.5}$$

Figure 9.5 illustrates the relation between the information-bit length and the check-bit length of the SEC-$S_{b/p\times b}$EL code determined by Theorems 9.8 and 9.9 for $b = 4$.

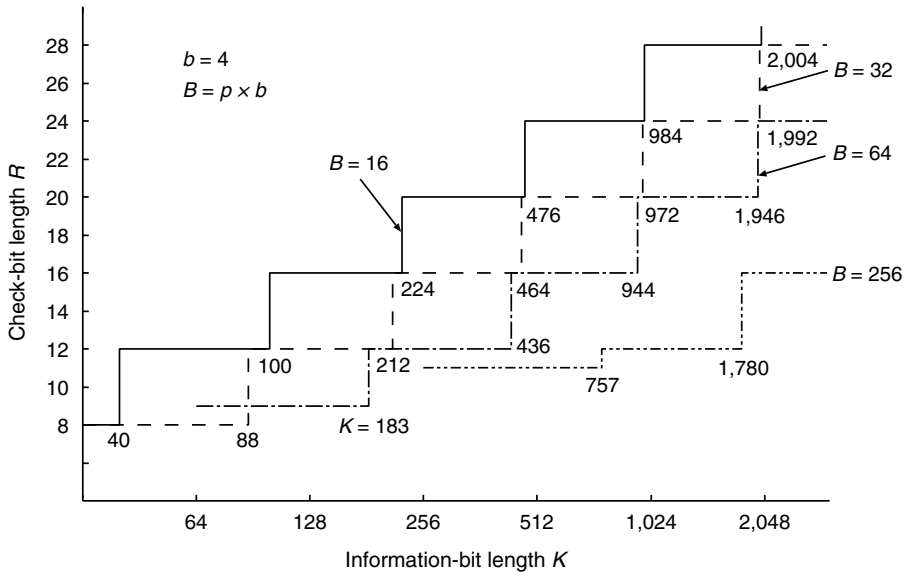


Figure 9.5 *Check-bit lengths compared with information-bit lengths of the SEC-$S_{b/p\times b}$EL type II codes for $b = 4$.* Source: *[FUJI94]. © 1994 IEEE.*

9.3.3 Decoding Procedure

It is apparent that single-bit error correction using an SEC-S$_{b/p\times b}$EL code can be performed in the same way as an SEC code.

The single-byte error location procedure of this code depends on the code design method. For type I codes (codes I), the decoding circuit of the Sb'EC codes includes the error locating circuit of the SEC-S$_{b/p\times b}$EL codes. This is because the location of any erroneous byte is determined in the decoding procedure of Sb'EC codes.

On the other hand, for type II codes (codes II), the error locating circuit is implemented by using the first br bits of the syndrome having length $R = br + c$ bits, where $0 \le c < b$. To see this, let the first br bits of the syndrome be S with r b-tuples, $S_0, S_1, \cdots, S_{r-1}$, shown below:

$$\begin{aligned} S &= \left[S_0 \; S_1 \; \cdots \; S_{r-1} \right] \\ &= \left[s_{0,0} \; s_{0,1} \; \cdots \; s_{0,b-1} \;\middle|\; s_{1,0} \; s_{1,1} \; \cdots \; s_{1,b-1} \;\middle|\; \cdots \;\middle|\; s_{r-1,0} \; s_{r-1,1} \; \cdots \; s_{r-1,b-1} \right] \\ & \quad s_{l,m} \in \{0,1\}, \quad 0 \le l \le r-1, \quad 0 \le m \le b-1. \end{aligned}$$

Let the syndrome S_l be obtained by the product of a byte error corresponding to the j-th byte in the i-th block and the transposed $b \times b$ square matrix located at the l-th position in the corresponding column in the parity-check matrix. The location of an erroneous block is determined by using the weight of S_l's. We define two variables, p_l and z_l, using $s_{l,m}$'s:

$$p_l = \sum_{m=0}^{b-1}{}^{\oplus} s_{l,m}$$

and

$$z_l = \bigvee_{m=0}^{b-1} s_{l,m},$$

where $\sum^{\oplus}$ represents modulo-2 sum. Next we define two additional binary variables, p' and q_l, using the variables p_l and z_l, where $0 \le l \le r-1$:

$$p' = \bigvee_{l=1}^{r-1} p_l$$

and

$$q_l = p' p_l \vee \overline{p'} z_l.$$

Here $\overline{p'}$ denotes the complement of p'. If the error vector has odd weight, then at least one of S_l's has odd weight, and hence $p' = 1$. In this case, S_l obtained by the product of odd-weight column square matrix has odd weight, and the one obtained by the product

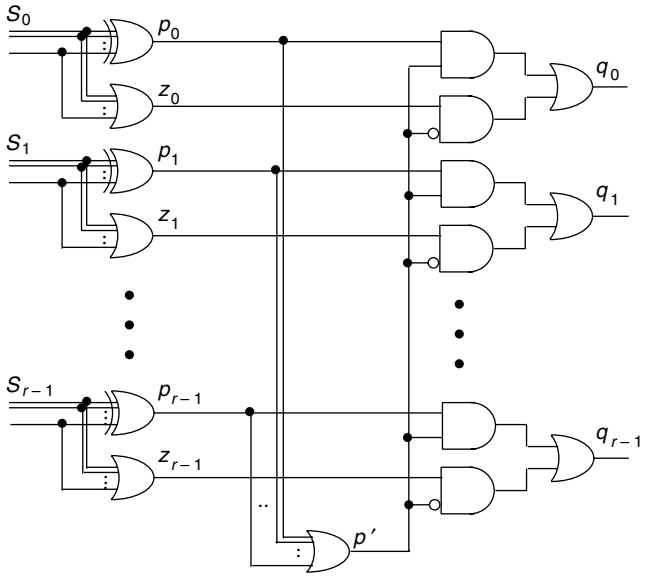


Figure 9.6 *Error locating circuit of SEC-$S_{b/p\times b}$EL type II codes.* Source: *[FUJI94]. © 1994 IEEE.*

of even-weight column square matrix has even weight. Therefore p_l indicates which matrices of the odd-weight column square matrix and the even-weight column square matrix are used at the l-th element of the corresponding column in the parity-check matrix. If the error vector has even weight, then $p' = 0$. In this case, z_l indicates which matrices of the odd-weight column square matrix and the even-weight column square matrix are used at the l-th element of the corresponding column in the parity-check matrix. The variable q_l combines the two cases above; that is, if $q_l = 1$, then the odd-weight column square matrix is used at the l-th element of the corresponding column in the parity-check matrix, and if $q_l = 0$, then the even-weight column square matrix is used. Based on the outcome above, the variable sequence $q_0 q_1 \cdots q_{r-1}$, where q_0 is the most significant bit, expresses the value equal to $i + 1$, where i is the location number of the erroneous block. This follows from the fact that if $\mathcal{A}$ and $\mathcal{B}$ are replaced by '1' and '0', respectively, in the column vector $\mathbf{H}_i$ shown in Theorem 9.8, which corresponds to the i-th block, then the binary vector takes the value of $i + 1$. Figure 9.6 illustrates the error locating circuit based on this concept.

9.3.4 Evaluation

Error Detection Capabilities The SEC-$S_{b/p\times b}$EL codes do not always detect random double-bit errors and also do not always detect double-byte errors. These errors sometimes induce the following erroneous decoding cases:

Case 1. Indicate location of the error-free block as an erroneous block, or mislocate.

Case 2. Invert the error-free bit, or miscorrect.

Case 3. Indicate as error free, or misdetect.

TABLE 9.1 Decoding Probabilities of the (72, 64) SEC-$S_{4/8\times4}$EL Code of Design Method II

Cases	Double-bit errors (%)	Double-byte errors (%)
Case 1. Mislocation	31.0	25.5
Case 2. Miscorrection	15.0	27.0
Case 3. Misdetection	0	1.1
Case 4. Detection	49.8	30.4
Case 5. Location	4.2	16.0

Source: [FUJI94]. © 1994 IEEE.

The following cases cover situations where the codes neither miscorrect, mislocate, or misdetect errors:

Case 4. Detect errors, but cannot correct or locate.

Case 5. Indicate correct location of the erroneous block in which all errors are included.

The (72, 64) SEC-$S_{4/8\times4}$EL code can be obtained by deleting the last 6 columns (i.e., 24 binary columns) from the matrix shown in Figure 9.4, and hence the last block size is 8 bits. The probabilities of the above-described five cases are calculated by computer simulation and are shown in Table 9.1.

Decoder Hardware Complexity Figure 9.7 shows the decoder hardware complexity of the SEC-$S_{b/p\times b}$EL codes for $b = 4$ bits and $B = 4 \times 4$ bits. In this figure we count a four-input AND / OR gate as one gate and an XOR gate as 2.5 gates. For the two code, design methods I and II, the difference in the gate count of the error correcting circuits in the decoder depends mainly on the number of check bits. On the other hand, the difference

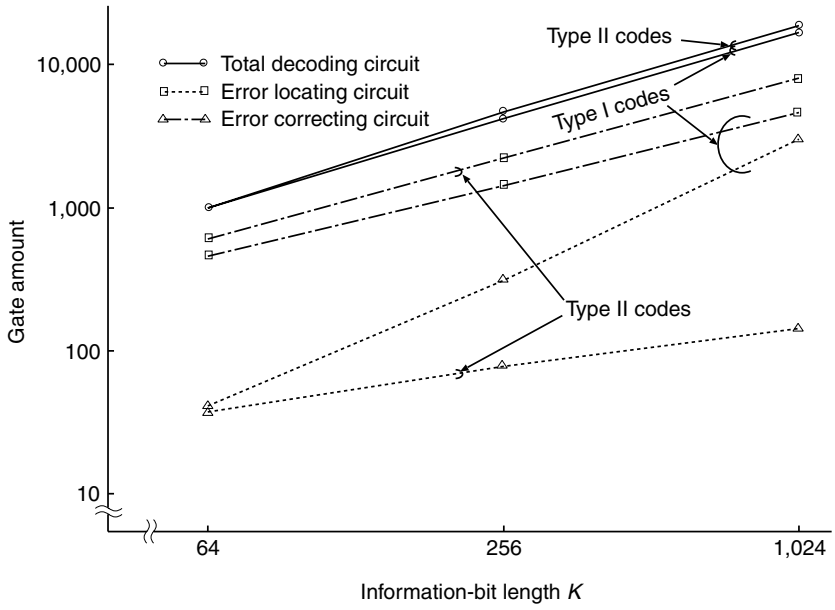


Figure 9.7 *Decoder gate counts of SEC-$S_{4/4\times4}$EL codes.* Source: *[FUJI94]. © 1994 IEEE.*

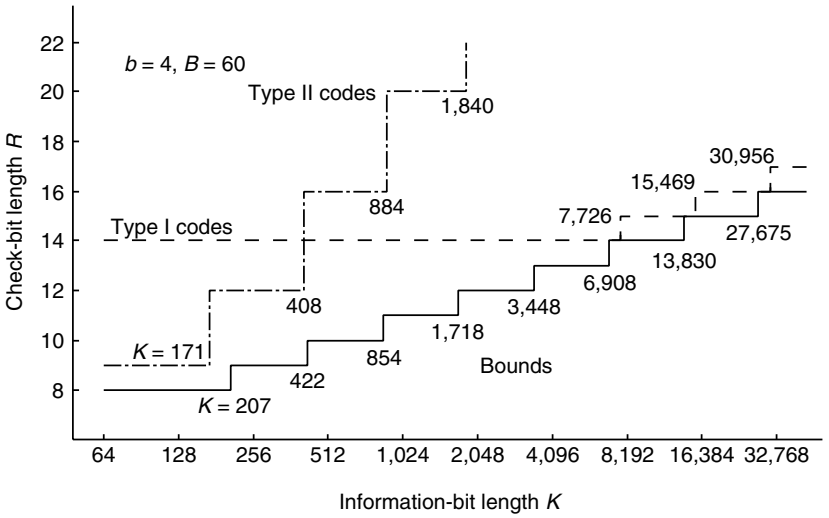


Figure 9.8 *Check-bit lengths compared with information-bit lengths of the SEC-$S_{4/15\times4}$EL codes.* Source: *[FUJI94]. © 1994 IEEE.*

in the gate count of the error locating circuits depends on the decoding procedure for error location, meaning the type II codes provide direct and therefore simple decoding from the syndrome, whereas the type I codes require the decoding procedures of both the Sb'EC codes and the SEC-SbED codes.

The total gate count of the decoding circuit for the SEC-$S_{4/4\times4}$EL codes is around 15% larger than that for the SEC-DED codes. This arises from the following facts. The redundancy of the former codes is greater than that of the latter codes, and therefore the syndrome generator and single-bit error correcting circuit of the former codes have almost a 10% larger gate count than those of the latter codes. Furthermore the single-byte error locating circuit is included in the decoder of the former codes.

Code Length Figure 9.8 shows the relation between the information-bit lengths and the check-bit lengths of the type I codes and the type II codes with code parameters of $b = 4$ bits and $B = 60$ bits. This also indicates the bounds on code length described in Theorems 9.5 and 9.6.

9.4 SINGLE-BIT ERROR CORRECTING AND SINGLE-BYTE ERROR LOCATING (SEC-$S_{e/b}$EL) CODES

We study here another class of error locating codes, namely *Single-bit Error Correcting and Single e-bit (within a b-bit byte) Error Locating codes*, or SEC-$S_{e/b}$EL *codes*.

9.4.1 Code Conditions and Bounds

Let $\mathbf{E}_s$ be the error set consisting of all single-bit errors, and let $\mathbf{E}_i(\mathbf{E}_j)$ be the set of e or fewer bits errors in the $i(j)$-th byte excluding single-bit errors, where $e < b$ and b is the byte length. Thus $\mathbf{E}_i \cap \mathbf{E}_s = \mathbf{E}_j \cap \mathbf{E}_s = \phi$ for all $i \neq j$, where ϕ is the empty set. The

following theorem describes the necessary and sufficient conditions that characterize SEC-$S_{e/b}$EL codes.

Theorem 9.10 *A linear code, described by the parity-check matrix* $\boldsymbol{H}$*, corrects all errors in* $\boldsymbol{E}_s$ *and locates all errors in* $\boldsymbol{E}_i$*, or* $\boldsymbol{E}_j$*, where* $i \neq j$*, if and only if:*

1. $E \cdot \boldsymbol{H}^T \neq 0$ *for all* $E \in \{\boldsymbol{E}_s \cup \boldsymbol{E}_i \cup \boldsymbol{E}_j\}$,
2. $E_1 \cdot \boldsymbol{H}^T \neq E_2 \cdot \boldsymbol{H}^T$ *for all* $E_1, E_2 \in \boldsymbol{E}_s$, $E_1 \neq E_2$,
3. $E_3 \cdot \boldsymbol{H}^T \neq E_4 \cdot \boldsymbol{H}^T$ *for all* $E_3 \in \boldsymbol{E}_i$, *and all* $E_4 \in \boldsymbol{E}_j$,
4. $E_1 \cdot \boldsymbol{H}^T \neq E_3 \cdot \boldsymbol{H}^T$ *for all* $E_1 \in \boldsymbol{E}_s$, *and all* $E_3 \in \boldsymbol{E}_i$,

where $\boldsymbol{H}^T$ *means the transpose of* $\boldsymbol{H}$.

Proof It is apparent that conditions 1 and 2 and conditions 1 and 3 are necessary conditions for correcting all single-bit errors and for indicating the location of an erroneous byte containing e or fewer errors, respectively. Conditions 1 and 4 are also necessary conditions for distinguishing single-bit errors from single-byte errors that exclude single-bit errors. So the SEC-$S_{e/b}$EL codes must satisfy conditions 1 to 4.

Inversely, a code that satisfies conditions 1 to 4 allows us to distinguish single-bit errors from single-byte errors. Then we can correct all single-bit errors and locate all single-byte errors. Therefore this code is an SEC-$S_{e/b}$EL code. Q.E.D.

Since the $S_{e/b}$EL codes, where $e = b$, are equivalent to the single b-bit byte error correcting codes (or SbEC codes), the SEC-$S_{e/b}$EL codes, where $e = b$, are also equivalent to the SbEC codes. This leads to the following theorem.

Theorem 9.11 *Linear SEC-*$S_{e/b}$*EL codes satisfy*

$$2 \leq e \leq b - 1.$$

The next two theorems give the lower bounds on check-bit length of the SEC-$S_{e/b}$EL codes.

Theorem 9.12 *Linear* $(N, N - R)$ *SEC-*$S_{e/b}$*EL codes satisfy*

$$2^R \geq \begin{cases} 1 + \dfrac{N}{b} \displaystyle\sum_{i=1}^{(e+1)/2} \binom{b}{i} & \text{for odd } e, \\ 1 + \dfrac{N}{b} \left\{ \displaystyle\sum_{i=1}^{e/2} \binom{b}{i} + \dfrac{\binom{b}{(e+2)/2}}{\lfloor 2b/(e+2) \rfloor} \right\} & \text{for even } e. \end{cases}$$

Proof The lower bounds on the check-bit length of the $S_{e/b}$EL codes are obtained by the fact that syndromes produced by $e/2$ or fewer bits errors should be different from each other [WOLF63].

1. ***For odd e.*** Syndromes produced by $(e - 1)/2$ or fewer bits errors in a given byte should differ from each other because, otherwise, there might exist $e - 1$ or fewer bits

errors in that byte that result in a zero syndrome. This violates condition 3 of Theorem 9.10. Conditions 3 and 4 say that syndromes produced by $(e-1)/2+1=(e+1)/2$-bit errors in a byte should be different from each other and also different from those produced by $(e-1)/2$ or fewer bits errors in that byte. In this case it is because, otherwise, there might exist e or fewer bits errors in that byte that result in a zero syndrome. This case violates the conditions of the code. Therefore syndromes produced by $(e+1)/2$ or fewer bits errors in a byte should differ from each other. It is apparent that syndromes produced by $(e+1)/2$ or fewer bits errors in a byte should also be different from those in another byte. Hence the following inequality holds:

$$2^R \geq 1+\frac{N}{b}\sum_{i=1}^{(e+1)/2}\binom{b}{i}.$$

2. *For even e.* By condition 3 of Theorem 9.10, the syndromes produced by $e/2$ or fewer bits errors in a byte should differ from each other. However, syndromes produced by $(e+2)/2$-bit errors in a byte are capable of being equal to those produced by other $(e+2)/2$-bit errors in the same byte only when there exist no erroneous bits at the same bit positions in these two $(e+2)/2$-bit errors. There exist at most

$$\left\lfloor b/\left(\frac{e+2}{2}\right)\right\rfloor=\left\lfloor\frac{2b}{e+2}\right\rfloor$$

of the $(e+2)/2$-bit errors in a byte that have the same syndromes, where $\lfloor x\rfloor$ means the largest integer less than or equal to x. From conditions 3 and 4 of Theorem 9.10, any syndromes produced by $(e+2)/2$-bit errors should be different from those produced by $e/2$-bit errors. Hence there exists at least the following number of distinct nonzero syndromes:

$$\frac{N}{b}\left\{\sum_{i=1}^{e/2}\binom{b}{i}+\frac{\binom{b}{(e+2)/2}}{\lfloor 2b/(e+2)\rfloor}\right\}.$$

Hence we have the following inequality:

$$2^R \geq 1+\frac{N}{b}\left\{\sum_{i=1}^{e/2}\binom{b}{i}+\frac{\binom{b}{(e+2)/2}}{\lfloor 2b/(e+2)\rfloor}\right\}.$$

Q.E.D.

Theorem 9.13 *Linear* $(N, N-R)$ *SEC-*$S_{e/b}$*EL codes satisfy*

$$R \geq 2e.$$

This theorem can be easily proved, and the proof is therefore omitted.

9.4.2 Design for SEC-S$_{e/b}$EL Codes

The tensor product of the error correcting codes and the error detecting codes is also applied in the design of SEC-S$_{e/b}$EL codes. That is, tensor product of an SbEC code and a *single-bit error correcting and e-bit error detecting code* (or SEC-eED *code*) produces the SEC-S$_{e/b}$EL code, as shown in the following theorem.

Theorem 9.14 *The code described by the following matrix* $\boldsymbol{H}$ *is an SEC-S*$_{e/b}$*EL code whose code length* $N = b \times n$ *bits:*

$$\begin{aligned}\boldsymbol{H} &= \boldsymbol{H}' \otimes \boldsymbol{H}'' \\ &= \left[\boldsymbol{H}'_0 \;\middle|\; \boldsymbol{H}'_1 \;\middle|\; \cdots \;\middle|\; \boldsymbol{H}'_{n-1}\right] \otimes \boldsymbol{H}'' \\ &= \left[\boldsymbol{H}'_0 \cdot \boldsymbol{H}'' \;\middle|\; \boldsymbol{H}'_1 \cdot \boldsymbol{H}'' \;\middle|\; \cdots \;\middle|\; \boldsymbol{H}'_{n-1} \cdot \boldsymbol{H}''\right] \\ &= \left[\boldsymbol{H}_0 \;\middle|\; \boldsymbol{H}_1 \;\middle|\; \cdots \;\middle|\; \boldsymbol{H}_{n-1}\right],\end{aligned}$$

where $\otimes$ *represents the tensor product,* $\boldsymbol{H}'$ *is the parity-check matrix of the S*b'*EC code whose code length is n bytes,* $\boldsymbol{H}'_i$ *is the submatrix of* $\boldsymbol{H}'$ *corresponding to the i-th byte,* $\boldsymbol{H}''$ *is the parity-check matrix of the* $(b, b - b')$ *SEC-eED code, and* $\boldsymbol{H}_i$ *is the submatrix of* $\boldsymbol{H}$ *corresponding to the i-th byte.*

Proof It is apparent that the code satisfies condition 1 of Theorem 9.10 for any single-bit errors and any e or fewer bits errors. Condition 2 of Theorem 9.10 is satisfied because the binary columns of $\mathbf{H}$ all differ. The syndromes resulting from any single-byte errors in the i-th byte is different from those in the j-th byte, where $i \neq j$, because each column in $\mathbf{H}_i$ is determined by the product of $\mathbf{H}'_i$ by $\mathbf{H}''$. In general, every $\mathbf{H}'_i$ includes a $b' \times b'$ nonsingular matrix, and every $\mathbf{H}_i$ includes a $b' \times b$ matrix obtained by the product of the nonsingular matrix and $\mathbf{H}''$. Consequently any syndrome resulting from e or fewer errors in a byte is nonzero and therefore satisfies condition 3 of Theorem 9.10. This condition also tells us that the syndromes caused by any single-bit errors are different from those caused by any single-byte errors excluding single-bit errors, and therefore condition 4 is also satisfied. From the above, the indicated matrix $\mathbf{H}$ satisfies all conditions of Theorem 9.10, and hence the code described by $\mathbf{H}$ is an SEC-S$_{e/b}$EL code. Q.E.D.

If we apply the maximal Sb'EC codes [HONG72] shown in Subsection 5.1.4, the maximum code length in bits of the SEC-S$_{e/b}$EL codes, defined by Theorem 9.14, can be expressed as follows:

$$N = b\frac{2^R - 1 - 2^{b'}(2^c - 1)}{2^{b'} - 1} + b'', \tag{9.6}$$

where R is the check-bit length of the SEC-S$_{e/b}$EL code and b'' is the code length in bits of the SEC-eED code having c check bits where $c = R \bmod b'$. If $e = 2$, for example, then $b = 2^{b'-1}$ and $b'' = 2^{c-1}$.

Example 9.3 [KITA95]: (36, 30) SEC-$S_{2/4}$EL Code

For $b = 4$, $e = 2$ and $b' = 3$, the following matrices $\mathbf{H}'$ and $\mathbf{H}''$ show the $(27, 21)$ S3EC code and the $(4, 1)$ SEC-DED code, respectively:

$$\mathbf{H}' = \left[\begin{array}{c|c|c|c|c|c|c|c|c} \mathbf{H}'_0 & \mathbf{H}'_1 & \mathbf{H}'_2 & \mathbf{H}'_3 & \mathbf{H}'_4 & \mathbf{H}'_5 & \mathbf{H}'_6 & \mathbf{H}'_7 & \mathbf{H}'_8 \end{array}\right]$$

$$= \left[\begin{array}{c|c|c|c|c|c|c|c|c} 100 & 000 & 100 & 100 & 100 & 100 & 100 & 100 & 100 \\ 010 & 000 & 010 & 010 & 010 & 010 & 010 & 010 & 010 \\ 001 & 000 & 001 & 001 & 001 & 001 & 001 & 001 & 001 \\ \hline 000 & 100 & 100 & 001 & 010 & 101 & 011 & 110 & 100 \\ 000 & 010 & 010 & 101 & 011 & 111 & 110 & 100 & 001 \\ 000 & 001 & 001 & 010 & 101 & 011 & 111 & 111 & 110 \end{array}\right],$$

$$\mathbf{H}'' = \left[\begin{array}{c} 1001 \\ 0101 \\ 0011 \end{array}\right].$$

The tensor product of these two codes produces the $(36, 30)$ SEC-$S_{2/4}$EL code shown in the following matrix $\mathbf{H}$:

$$\mathbf{H} = \mathbf{H}' \otimes \mathbf{H}''$$

$$= \left[\begin{array}{c|c|c|c|c|c|c|c|c} \mathbf{H}'_0 \cdot \mathbf{H}'' & \mathbf{H}'_1 \cdot \mathbf{H}'' & \mathbf{H}'_2 \cdot \mathbf{H}'' & \mathbf{H}'_3 \cdot \mathbf{H}'' & \mathbf{H}'_4 \cdot \mathbf{H}'' & \mathbf{H}'_5 \cdot \mathbf{H}'' & \mathbf{H}'_6 \cdot \mathbf{H}'' & \mathbf{H}'_7 \cdot \mathbf{H}'' & \mathbf{H}'_8 \cdot \mathbf{H}'' \end{array}\right]$$

$$= \left[\begin{array}{c|c|c|c|c|c|c|c|c} 1001 & 0000 & 1001 & 1001 & 1001 & 1001 & 1001 & 1001 & 1001 \\ 0101 & 0000 & 0101 & 0101 & 0101 & 0101 & 0101 & 0101 & 0101 \\ 0011 & 0000 & 0011 & 0011 & 0011 & 0011 & 0011 & 0011 & 0011 \\ \hline 0000 & 1001 & 1001 & 0011 & 0101 & 1010 & 0110 & 1100 & 1001 \\ 0000 & 0101 & 0101 & 1010 & 0110 & 1111 & 1100 & 1001 & 0011 \\ 0000 & 0011 & 0011 & 0101 & 1010 & 0110 & 1111 & 1111 & 1100 \end{array}\right].$$

9.4.3 Evaluation

Code Length Figure 9.9 illustrates the relation between the information-bit length and the check-bit length of the SEC-$S_{e/8}$EL codes for $e = 2$ and 6. In the figure the lower bounds are those obtained by Theorems 9.12 and 9.13. For comparison, the cases of SEC codes, SEC-DED-S8ED codes, and S8EC codes are also shown. Note that the codes are close to the bounds for smaller values of e and for larger information-bit lengths. Therefore they are very efficient under some code parameters. Note also that the SEC-$S_{2/8}$EL codes have larger check-bit lengths than the SEC codes by almost one bit. Although not indicated in the figure, they have almost the same check-bit lengths as the SEC-DED codes. The maximal SbEC codes [HONG72] and the efficient SEC-eED codes—meaning the BCH codes with Hamming distance $e + 2$ for $b < 1000$ and $(e + 2) < b/2$, the Hamming SEC-DED codes for $e = 2$, and the repetition codes [MCWL77] for $e = b - 2$ should then provide the efficient SEC-$S_{e/b}$EL code.

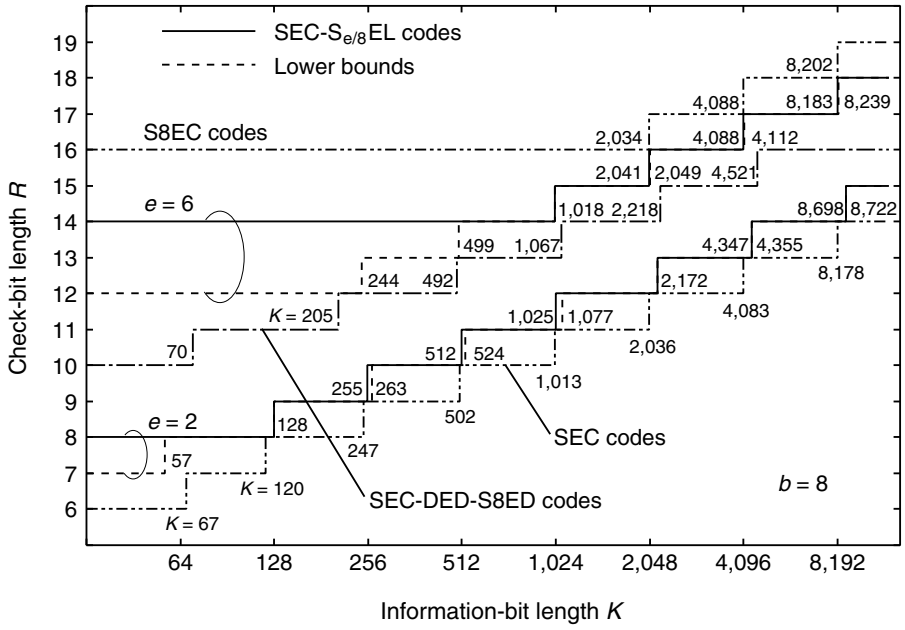


Figure 9.9 *Check-bit lengths compared with information-bit lengths of the SEC-S$_{e/8}$EL codes.* Source: *[KITA95]. © 1995 IEICE Japan.*

Figure 9.10 gives an example of the (72, 64) SEC-S$_{2/4}$EL codes that is a shortened version of the original (132, 124) SEC-S$_{2/4}$EL codes.

Next we find the restriction on e existing in the SEC-S$_{e/b}$EL codes, which we obtain by the tensor product of two codes.

Lemma 9.3 *An $(n, n-r)$ binary linear code with minimum Hamming distance d does not exist under the following condition of the parameters, d and n:*

$$\frac{2}{3}n < d < n.$$

Proof For $d = n$, the code is a repetition code and includes only two codewords. For $0 < d \leq n/2$, it is apparent that the code includes at least four codewords.

$$\mathbf{H} = \begin{bmatrix}
1001\ 0000\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001\ 1001 \\
0101\ 0000\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101\ 0101 \\
0011\ 0000\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011 \\
0000\ 1001\ 1001\ 0000\ 0000\ 0011\ 0110\ 1111\ 1100\ 1001\ 0011\ 0110\ 1100\ 1010\ 0110\ 1111\ 1111\ 1111 \\
0000\ 0101\ 0101\ 1001\ 0000\ 0011\ 0101\ 1001\ 0011\ 0101\ 1010\ 0101\ 1010\ 0110\ 1100\ 1001\ 0000\ 0000 \\
0000\ 0011\ 0011\ 0101\ 1001\ 0011\ 0101\ 1010\ 0101\ 1010\ 0110\ 1100\ 1001\ 0000\ 0000\ 0011\ 0110\ 1111 \\
0000\ 0000\ 0000\ 0011\ 0101\ 1001\ 0011\ 0101\ 1010\ 0101\ 1010\ 0110\ 1100\ 1001\ 0000\ 0000\ 0011\ 0110 \\
0000\ 0000\ 0000\ 0000\ 0011\ 0110\ 1111\ 1100\ 1001\ 0011\ 0110\ 1100\ 1010\ 0110\ 1111\ 1111\ 1111\ 1100
\end{bmatrix}$$

Figure 9.10 *Example of (72, 64) SEC-S$_{2/4}$EL codes.* Source: *[KITA95]. © 1995 IEICE Japan.*

Next we consider the case for $n/2 < d < n$. Assume that there exist more than two nonzero distinct codewords in the $(n, n-r)$ linear code. Let $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, where $\boldsymbol{c}_1 \neq \boldsymbol{c}_2$, be codewords of the linear code. In this case we can choose $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ that satisfy the following conditions:

1. $w_H(\boldsymbol{c}_1) = w_H(\boldsymbol{c}_2) = d$.
2. There exist no 0's in the same bit positions in $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$.

Here $w_H(\boldsymbol{c})$ means the Hamming weight of $\boldsymbol{c}$. Let f be the number of bit positions having the same value of 1 between $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$. Then the Hamming weight of $\boldsymbol{c}_1 + \boldsymbol{c}_2$ is expressed by $w_H(\boldsymbol{c}_1 + \boldsymbol{c}_2) = n - f$. Since $\boldsymbol{c}_1 + \boldsymbol{c}_2$ is another codeword of the code, the following relation holds:

$$n - f \geq d. \tag{9.7}$$

Under $d > n/2$, the code length n (bits) can be expressed as

$$n = 2d - f. \tag{9.8}$$

From Eq. (9.8) we have $f = 2d - n$. Substituting this relation into (9.7) leads to

$$\frac{1}{2}n < d \leq \frac{2}{3}n.$$

For the remaining region of d (i.e., $(2/3)n < d < n$) the number of codewords in the $(n, n-r)$ linear code is less than or equal to 2. The code length of the code having two codewords must be less than n because each codeword has larger than or equal to one '0'. This contradicts the code that we now consider whose code length has n bits.

We can conclude in this case that an $(n, n-r)$ binary linear code with minimum Hamming distance d does not exist for the condition that

$$\frac{2}{3}n < d < n.$$

Q.E.D.

Theorem 9.15 *The SEC-$S_{e/b}$EL codes based on the tensor product of the SbEC codes and the $(b, b-b')$ SEC-eED codes exist under the following condition:*

$$2 \leq e \leq \frac{2}{3}b - 2, \qquad \text{or} \qquad e = b - 2.$$

Proof For $e = b$, the SEC-$S_{e/b}$EL codes are equivalent to the SbEC codes, and therefore they do not exist. For $e = b - 1$, the SEC-eED codes do not exist because minimum distance of the code is $e + 2$. By Lemma 9.3, the SEC-eED codes whose code length is b bits also do not exist for $(2/3)b < d = e + 2 < b$, that is, for $(2/3)b - 2 < e < b - 2$.

TABLE 9.2 Rate of Decoding Cases in (72, 64) SEC-$S_{2/4}$EL Code

Decoding cases	Double-bit errors (%)	Double-byte errors (%)	3- or 4-Bit errors in a byte (%)
Case 1. Mislocation	34.0	27.0	0
Case 2. Miscorrection	9.0	29.5	80.0
Case 3. Misdetection	0	0.5	20.0
Case 4. Detection	52.8	43.0	0
Case 5. Location	4.2	0	0

Source: [KITA95]. © 2005 IEICE Japan.

From the above, the SEC-$S_{e/b}$EL codes exist under the following condition:

$$2 \le e \le \frac{2}{3}b - 2, \qquad \text{or} \qquad e = b - 2.$$

Q.E.D.

By Theorem 9.15, for example, the SEC-eED codes with $e = 4$ and 5, for $b = 8$, do not exist. This is why Figure 9.9 does not include codes with these values of e.

Error Detection Capabilities The SEC-$S_{e/b}$EL codes do not always detect random double-bit errors and random double-byte errors, and these codes also do not always indicate the correct location of an erroneous byte with larger than e bits errors. These errors sometimes engage the five decoding cases mentioned in Subsection 9.3.4.

Table 9.2 lists the rate of the decoding cases expressed by percentage of the shortened (72, 64) SEC-$S_{2/4}$EL code shown in Figure 9.10 for random double-bit errors, double-byte errors, and random 3- or 4-bit errors in a byte that are beyond the original error control capability of the code.

9.5 BURST ERROR LOCATING CODES

This section deals with a class of error locating codes for burst errors. These codes locate the erroneous frame containing the burst errors, *l-bit burst error locating codes*, called B_lEL codes [DASS82], and *single-bit error correcting and l-bit burst error locating codes*, called SEC-B_lEL codes [KITA98, 05].

9.5.1 Frame Set

Let a *frame* be defined as a set of clustered symbols in a codeword. A frame that begins and ends at the i-th and j-th positions, respectively, is described by $[i, j]$. Here we define the frame set F for a codeword having N symbols as

$$F = \{[di,\ di + L - 1] \,|\, 0 \le i \le (N - L + 1)/d\}.$$

Figure 9.11 illustrates the frame set F. Adjacent frames in F are partially overlapped by $L - d$ symbols. If $L - d \ge l - 1$ and $L \ge l$, there exists at least one frame in F that contains some l-burst errors. This is the same idea as the frame overlapping with adjacent frames by $l - 1$ mentioned in Subsection 8.1.2.

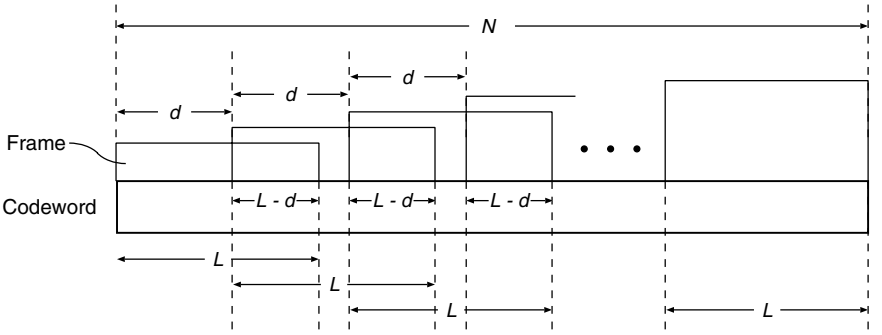


Figure 9.11 *Frame set.* Source: *[KITA05]. © 2005 IEEE.*

9.5.2 Burst Error Locating (B_lEL) Codes

By using the concept of the frame, we define a new class of burst error locating codes.

Definition 9.4 A code is called an *l-burst error locating code* if and only if there exists at least one frame that contains single l-burst error, and the code indicates the one such frame in F. □

Next we present the B_lEL codes [KITA05] that satisfy Definition 9.4.

Theorem 9.16 *Let $\boldsymbol{H}' = [h'_0\ h'_1\ \cdots\ h'_{n-1}]$ be a parity-check matrix of an $(n,\ n-R')$ l-bit burst error correcting code, where h'_i is an i-th binary column vector of $\boldsymbol{H}'$, $0 \le i \le n-1$, and R' is a check-bit length. Also let $\boldsymbol{H}'_i = [h'_i\ h'_{i+1}\ \cdots\ h'_{i+l-1}]$ be a sub-matrix including consecutive l columns from the i-th column of $\boldsymbol{H}'$, where $0 \le i \le n-l$. The code defined by the following parity-check matrix $\boldsymbol{H}$ is a $((pl+1)(n-l+1),\ (pl+1)(n-l+1)-R')$ B_lEL code with a frame set F:*

$$\begin{aligned}
\boldsymbol{H} &= \Big[\underbrace{\boldsymbol{H}'_0\,\boldsymbol{H}'_0\cdots\boldsymbol{H}'_0}_{p\times l}\,\underbrace{h'_0}_{1}\,\underbrace{\boldsymbol{H}'_1\,\boldsymbol{H}'_1\cdots\boldsymbol{H}'_1}_{p\times l}\,\underbrace{h'_1}_{1}\,\Big|\cdots \\
&\qquad \cdots\,\underbrace{\boldsymbol{H}'_{n-l}\,\boldsymbol{H}'_{n-l}\cdots\boldsymbol{H}'_{n-l}}_{p\times l}\,\underbrace{h'_{n-l}}_{1}\Big], \\
F &= \{[i(pl+1),\ (i+l)(pl+1)-1]\mid 0\le i\le n-2l+1\},
\end{aligned}$$

where p is an integer.

Proof Let $\mathbf{P}$ be the $n\times(pl+1)(n-l+1)$ matrix defined below:

$$\begin{aligned}
\mathbf{P} &= \Big[\underbrace{\mathbf{P}_0\,\mathbf{P}_0\cdots\mathbf{P}_0}_{p\times l}\,\underbrace{p_0}_{1}\,\underbrace{\mathbf{P}_1\,\mathbf{P}_1\cdots\mathbf{P}_1}_{p\times l}\,\underbrace{p_1}_{1}\,\Big|\cdots \\
&\qquad \underbrace{\mathbf{P}_{n-l}\,\mathbf{P}_{n-l}\cdots\mathbf{P}_{n-l}}_{p\times l}\,\underbrace{p_{n-l}}_{1}\Big].
\end{aligned} \tag{9.9}$$

In Eq. (9.9), $\mathbf{P}_i$ is an $n \times l$ binary matrix and p_i is a binary n-tuple, whose elements are defined below:

$$(\mathbf{P}_i)_{j,m} = \begin{cases} 1 & m = i + j, \\ 0 & \text{otherwise}, \end{cases} \qquad (p_i)_j = \begin{cases} 1 & j = i, \\ 0 & \text{otherwise}. \end{cases}$$

Let E be an error vector with length $(pl + 1)(n - l + 1)$ bits including an l-bit burst error. A vector E' with length n bits is defined by the product of E and the transposed $\mathbf{P}$, meaning $E' = E \cdot \mathbf{P}^T$. Let m be a bit position of the first nonzero element in E'. From the organization of the matrix $\mathbf{P}$, E' is an error vector with length n bits including the l-bit burst error. Then the burst error in E exists in the frame $[m(pl + 1), (m + l)(pl + 1) - 1]$.

It is apparent that the parity-check matrix $\mathbf{H}$ is expressed by the product of $\mathbf{H}'$ and $\mathbf{P}$, meaning $\mathbf{H} = \mathbf{H}' \cdot \mathbf{P}$. Let S be the syndrome caused by the error vector E. Then the following relation holds:

$$S = E \cdot \mathbf{H}^T = E \cdot (\mathbf{H}' \cdot \mathbf{P})^T = \left(E \cdot \mathbf{P}^T\right)\mathbf{H}'^T.$$

Since the code defined by $\mathbf{H}'$ is an l-bit burst error correcting code, $E \cdot \mathbf{P}^T$ is uniquely determined by the syndrome, and the frame that includes E is also determined. Therefore the code defined by $\mathbf{H}$ is a B$_l$EL code. Q.E.D.

9.5.3 Single-Bit Error Correcting and Burst Error Locating (SEC-B$_l$EL) Codes

Let $\mathbf{E}_s$ be an error set consisting of all single-bit errors, and $\mathbf{E}_l$ be an error set of all l-burst errors excluding single-bit errors, meaning $\mathbf{E}_s \cap \mathbf{E}_l = \phi$. The following theorem indicates the necessary and sufficient conditions of the SEC-B$_l$EL codes.

Theorem 9.17 *A linear code, described by a parity-check matrix $\boldsymbol{H}$, corrects all errors in $\boldsymbol{E}_s$ and indicates a frame in F_{all} that contains errors in $\boldsymbol{E}_l$ if and only if:*

1. $E \cdot \boldsymbol{H}^T \neq 0$ *for all* $E \in \{\boldsymbol{E}_s \cup \boldsymbol{E}_l\}$,
2. $E_1 \cdot \boldsymbol{H}^T \neq E_2 \cdot \boldsymbol{H}^T$ *for all* $E_1, E_2 \in \boldsymbol{E}_s, E_1 \neq E_2$,
3. $E_p \cdot \boldsymbol{H}^T \neq E_q \cdot \boldsymbol{H}^T$ *for all* $E_p, E_q \in \boldsymbol{E}_l$, $f_1, f_2 \in F_{all}$, $E_p \subset f_1, E_q \subset f_2, E_p, E_q \not\subset f_1 \cap f_2$,
4. $E_1 \cdot \boldsymbol{H}^T \neq E_p \cdot \boldsymbol{H}^T$ *for all* $E_1 \in \boldsymbol{E}_s$, *all* $E_p \in \boldsymbol{E}_l$.

Theorem 9.17 can be easily proved. Conditions 1 to 4 are for error detection, single-bit error correction, burst error location, and discrimination between single-bit errors and burst errors, respectively. The following theorem presents the SEC-B$_l$EL code [KITA05] that satisfies Theorem 9.17.

Theorem 9.18 *The code defined by the following parity-check matrix $\boldsymbol{H}_L$ is a $((pl + 1)(n - l + 1),\ (pl + 1)(n - l + 1) - R' - \lceil log_2(pl + 1) \rceil)$ SEC-B$_l$EL code:*

$$\boldsymbol{H}_L = \begin{bmatrix} \boldsymbol{H} \\ \boldsymbol{Q} \end{bmatrix},$$

where $\boldsymbol{H}$ *is the parity-check matrix of a* $((pl+1)(n-l+1),\ (pl+1)(n-l+1)-R')$ B_lEL *code defined in Theorem 9.16,* $\boldsymbol{Q}$ *is a* $\lceil log_2(pl+1)\rceil \times (pl+1)(n-l+1)$ *matrix added to make every binary column in* $\boldsymbol{H}_L$ *distinct, and* $\lceil x \rceil$ *is the smallest integer no less than x.*

Proof Since column h_j appears in $\mathbf{H}$ at most $pl+1$ times in the B_lEL code in Theorem 9.16, it is possible to make every column in $\mathbf{H}_L$ distinct by adding the matrix $\mathbf{Q}$ with $\lceil \log_2(pl+1)\rceil$ rows. From the organization of the matrix $\mathbf{H}_L$, conditions 1 and 2 of Theorem 9.17 are satisfied. Since $\mathbf{H}$ is a parity-check matrix of B_lEL code, condition 3 is satisfied. Condition 4 is also satisfied because $E_1 \cdot \mathbf{P} \neq E_p \cdot \mathbf{P}$ for all $E_1 \in \mathbf{E}_s$ and $E_p \in \mathbf{E}_l$, where $\mathbf{P}$ is the matrix defined by Eq. (9.9). Therefore the code defined by $\mathbf{H}_L$ is an SEC- B_lEL code. Q.E.D.

Example 9.4 [KITA05]

The following shows the parity-check matrix of a $(15,\ 9)$ 3-bit burst error correcting code with parameters of $l=3$, $n=15$ and $R'=6$:

$$\mathbf{H}' = \begin{bmatrix} 1&1&0&0&1&1&1&0&0&1&0&0&0&0&0 \\ 0&1&1&0&0&1&1&1&0&0&1&0&0&0&0 \\ 0&0&1&1&0&0&1&1&1&0&0&1&0&0&0 \\ 0&0&0&1&1&0&0&1&1&1&0&0&1&0&0 \\ 0&0&0&0&1&1&0&0&1&1&1&0&0&1&0 \\ 0&0&0&0&0&1&1&0&0&1&1&1&0&0&1 \end{bmatrix} = [h'_0\, h'_1\, h'_2, \cdots, h'_{14}].$$

Submatrices $\mathbf{H}'_0,\ \mathbf{H}'_1,\ \cdots,\ \mathbf{H}'_{12}$ can be obtained from $\mathbf{H}'$ as follows:

$$\mathbf{H}'_0 = \begin{bmatrix} 1&1&0 \\ 0&1&1 \\ 0&0&1 \\ 0&0&0 \\ 0&0&0 \\ 0&0&0 \end{bmatrix} \quad \mathbf{H}'_1 = \begin{bmatrix} 1&0&0 \\ 1&1&0 \\ 0&1&1 \\ 0&0&1 \\ 0&0&0 \\ 0&0&0 \end{bmatrix} \quad \cdots, \quad \mathbf{H}'_{12} = \begin{bmatrix} 0&0&0 \\ 0&0&0 \\ 0&0&0 \\ 1&0&0 \\ 0&1&0 \\ 0&0&1 \end{bmatrix}$$

$$= [h'_0\, h'_1\, h'_2], \qquad = [h'_1\, h'_2\, h'_3], \qquad = [h'_{12}\, h'_{13}\, h'_{14}].$$

The following $(91,\ 85)$ B_3EL code with $p=2$ can be designed:

$$\mathbf{H} = \left[\ \mathbf{H}'_0\ \mathbf{H}'_0\ \vdots\ h'_0\ \middle|\ \mathbf{H}'_1\ \mathbf{H}'_1\ \vdots\ h'_1\ \middle|\ \cdots\ \middle|\ \mathbf{H}'_{12}\ \mathbf{H}'_{12}\ \vdots\ h'_{12}\ \right].$$

This code has a frame set $\{[0,20],\ [7,27],\ [14,34],\ \cdots,\ [70,90]\}$. After appending the matrix $\mathbf{Q}$ having $\lceil \log_2(pl+1)\rceil = 3$ rows, the parity-check matrix of $(91,\ 82)$ SEC-B_3EL code becomes

$$\mathbf{H}_L = \left[\begin{array}{c|c|c|c|c} \mathbf{H}'_0\ \mathbf{H}'_0\ \vdots\ h'_0 & \mathbf{H}'_1\ \mathbf{H}'_1\ \vdots\ h'_1 & \cdots & \mathbf{H}'_{12}\ \mathbf{H}'_{12}\ \vdots\ h'_{12} \\ \hline \multicolumn{5}{c}{\mathbf{Q}} \end{array}\right].$$

Figures 9.12 and 9.13 show the parity-check matrices of the $(91,\ 85)$ B_3EL code and the $(91,\ 82)$ SEC-B_3EL code, respectively.

$$
\mathbf{H} = \left[\begin{array}{c:c|c:c|c:c|c:c|c:c|c:c|c:c|c:c|c:c|c:c|c:c|c:c|c:c}
110110 & 1 & 100100 & 1 & 001001 & 0 & 011011 & 0 & 111111 & 1 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 010010 & 0 & 100100 & 1 & 000000 & 0 & 000000 & 0 & 000000 & 0 \\
011011 & 0 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 011011 & 0 & 111111 & 1 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 010010 & 0 & 100100 & 1 & 000000 & 0 & 000000 & 0 \\
001001 & 0 & 011011 & 0 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 011011 & 0 & 111111 & 1 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 010010 & 0 & 100100 & 1 & 000000 & 0 \\
000000 & 0 & 001001 & 0 & 011011 & 0 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 011011 & 0 & 111111 & 1 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 010010 & 0 & 100100 & 1 \\
000000 & 0 & 000000 & 0 & 001001 & 0 & 011011 & 0 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 011011 & 0 & 111111 & 1 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 010010 & 0 \\
000000 & 0 & 000000 & 0 & 000000 & 0 & 001001 & 0 & 011011 & 0 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 011011 & 0 & 111111 & 1 & 110110 & 1 & 100100 & 1 & 001001 & 0
\end{array}\right]
$$

Figure 9.12 *Parity-check matrix **H** of the (91, 85) B_3EL code.* Source: *[KITA05].* *© 2005 IEEE.*

$$
\mathbf{H}_L = \left[\begin{array}{cccccccccccccccccccccccccccccccccccc}
110110 & 1 & 100100 & 1 & 001001 & 0 & 011011 & 0 & 111111 & 1 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 010010 & 0 & 100100 & 1 & 000000 & 0 & 000000 & 0 & 000000 & 0 \\
011011 & 0 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 011011 & 0 & 111111 & 1 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 010010 & 0 & 100100 & 1 & 000000 & 0 & 000000 & 0 \\
001001 & 0 & 011011 & 0 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 011011 & 0 & 111111 & 1 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 010010 & 0 & 100100 & 1 & 000000 & 0 \\
000000 & 0 & 001001 & 0 & 011011 & 0 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 011011 & 0 & 111111 & 1 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 010010 & 0 & 100100 & 1 \\
000000 & 0 & 000000 & 0 & 001001 & 0 & 011011 & 0 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 011011 & 0 & 111111 & 1 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 010010 & 0 \\
000000 & 0 & 000000 & 0 & 000000 & 0 & 001001 & 0 & 011011 & 0 & 110110 & 1 & 100100 & 1 & 001001 & 0 & 011011 & 0 & 111111 & 1 & 110110 & 1 & 100100 & 1 & 001001 & 0 \\
\hline
000000 & 0 & 000000 & 1 & 100100 & 1 & 100100 & 1 & 100100 & 1 & 100100 & 1 & 100100 & 1 & 100100 & 1 & 100100 & 1 & 100100 & 1 & 100100 & 1 & 100100 & 1 & 100100 & 1 \\
000000 & 1 & 110110 & 0 & 010010 & 1 & 010010 & 1 & 010010 & 1 & 010010 & 1 & 010010 & 1 & 010010 & 1 & 010010 & 1 & 010010 & 1 & 010010 & 1 & 010010 & 1 & 010010 & 1 \\
000111 & 0 & 000111 & 0 & 000111 & 0 & 000111 & 0 & 000111 & 0 & 000111 & 0 & 000111 & 0 & 000111 & 0 & 000111 & 0 & 000111 & 0 & 000111 & 0 & 000111 & 0 & 000111 & 0
\end{array}\right]
$$

Figure 9.13 *Parity-check matrix $\mathbf{H}_L$ of the (91, 82) SEC-B_3EL Code.* Source: *[KITA05]. © 2005 IEEE.*

9.5.4 Decoding Procedure

Single-bit error correction is easily executed. If the syndrome of the received word is equal to one column vector of the parity-check matrix $\mathbf{H}$, the corresponding bit in the received word is erroneous, and then the error is corrected by inverting the bit.

Burst error location is determined by decoding the burst error correcting codes defined by $\mathbf{H}'$. The upper R' bits of the syndrome are used for the burst error location by a parallel procedure based on the method shown in Section 8.1. That is, using the $R' \times l$ matrix $\mathbf{H}'_i$, appended by the $R' \times (R' - l)$ matrix $\mathbf{B}'_i$, we have an $R' \times R'$ nonsingular matrix $\mathbf{A}'_i = [\mathbf{H}'_i\ \mathbf{B}'_i]$ for $0 \le i \le n - l$. The inverse matrix $(\mathbf{A}'_i)^{-1}$ is as presented below, where $\mathbf{H}^{\dagger}_i$ and $\mathbf{B}^{\dagger}_i$ are $l \times R'$ and $(R' - l) \times R'$ matrices, respectively:

$$(\mathbf{A}'_i)^{-1} = \begin{bmatrix} \mathbf{H}^{\dagger}_i \\ \mathbf{B}^{\dagger}_i \end{bmatrix}.$$

Suppose that first R' bits of the syndrome are S. If $S \cdot \mathbf{B}_j^{\dagger T} = 0$, then the frame $[j(pl+1),\ (j+l)(pl+1) - 1]$ is determined to be erroneous for $0 \le j \le n - 1$. The last frame $[(n - 2l + 1)(pl + 1),\ (n - l + 1)(pl + 1) - 1]$ is erroneous if the following relation is satisfied:

$$\bigvee_{j=n-2l+1}^{n-l} \left(S \cdot \mathbf{B}_j^{\dagger T} = 0 \right).$$

For burst errors of lengths less than l bits, $S \cdot \mathbf{B}_j^{\dagger T} = 0$ will hold for the two adjacent frames. This means that the burst error has occurred in the overlapped area of two frames. In such a case it is enough to indicate one of the erroneous frames.

Figure 9.14 illustrates the parallel decoder of the SEC-B$_l$EL codes in a block diagram.

9.5.5 Evaluation

Figure 9.15 shows the check-bit lengths of the SEC-B$_3$EL codes. The parity-check matrix of the burst error correcting code is generated by computer search [KASA63]. For

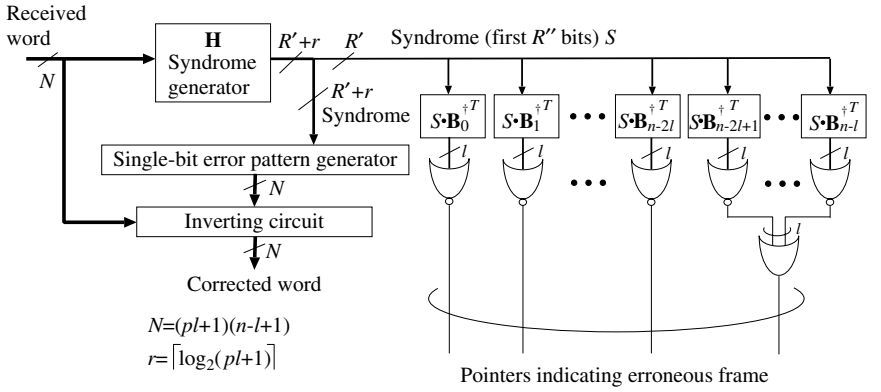


Figure 9.14 *Block diagram of the parallel decoder of SEC-B$_l$EL codes.* Source: *[KITA05]. © 2005 IEEE.*

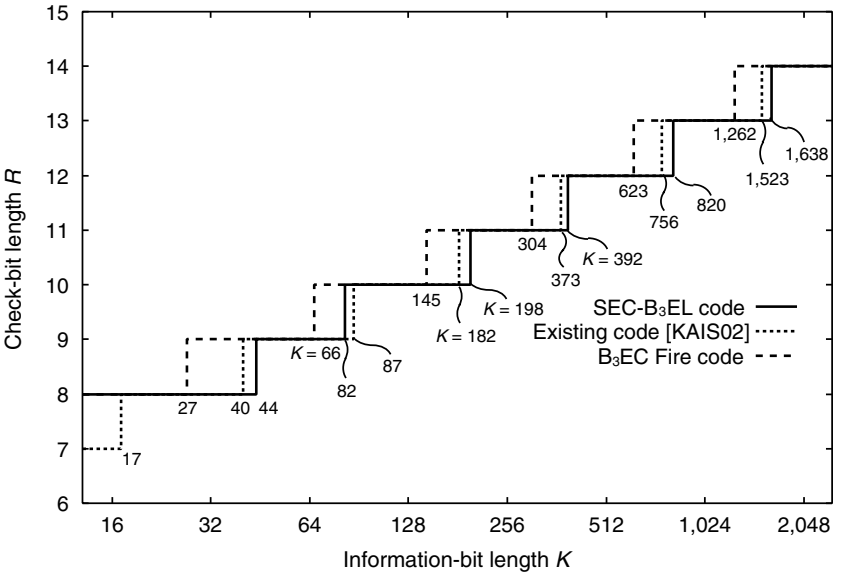


Figure 9.15 *Check-bit lengths compared with information-bit lengths of the SEC-B_3EL codes.* Source: *[KITA05]. © 2005 IEEE.*

comparison, the existing SEC-B_3EL codes [KAIS02] and the burst error correcting Fire codes [FIRE59] are also included in this figure.

The parallel decoder of the codes is designed by hardware description language. Figure 9.16 shows the decoder hardware amounts of the (132, 122), (231, 220), and (528, 516) SEC-B_3EL codes. In this figure the hardware amount is expressed by the

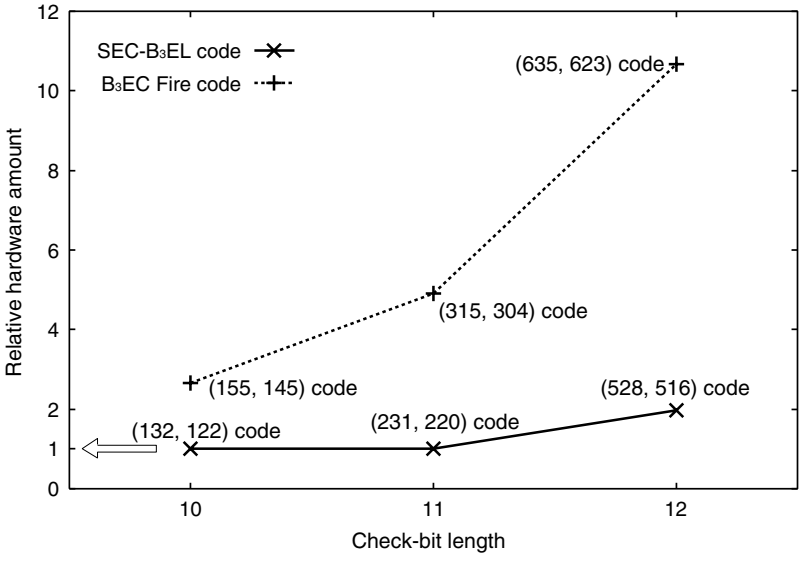


Figure 9.16 *Relative hardware amounts of the parallel decoder for the SEC-B_3EL codes.* Source: *[KITA05]. © 2005 IEEE.*

relative circuit area of the decoder, where the area for the $(132,\ 122)$ SEC-B_3EL code is considered to be 1. For comparison, the cases of the $(155,\ 145)$, $(315,\ 304)$, and $(635,\ 623)$ B_3EC Fire codes (i.e., three cases of 3-bit burst error correcting Fire codes) are also presented. The parallel decoder of the Fire code is designed by the method shown in Section 8.1. This figure says that the hardware amount of the parallel decoder of the SEC-B_3EL code is 20 to 40 percent of that of the burst error correcting Fire codes.

9.6 CODE CONDITIONS OF ERROR LOCATING CODES

This section mentions the necessary and sufficient conditions of the error locating codes in a generalized form, and clarifies the relation between the error locating codes and the error correcting / detecting codes [KITA97].

9.6.1 Preliminaries

Suppose that each of the codewords $\boldsymbol{X}$ and $\boldsymbol{Y}$ has N-bit length divided into n bytes, each having b-bit length (i.e., $N = b \times n$). Also suppose that the i-th bytes of $\boldsymbol{X}$ and $\boldsymbol{Y}$ are X_i and Y_i, $0 \leq i \leq n-1$, respectively. If e or fewer errors occur in the b-bit byte, this type of error is expressed as *e/b-error*, where $1 \leq e \leq b$. If $e = 1$, this shows a single-bit error in a byte, and if $e = b$, this shows an ordinary *byte error*.

Definition 9.5 The metric function for the vectors $\boldsymbol{X} = (X_0,\ X_1,\ \ldots,\ X_{n-1})$ and $\boldsymbol{Y} = (Y_0,\ Y_1,\ \ldots,\ Y_{n-1})$ is defined by

$$D_e(\boldsymbol{X}, \boldsymbol{Y}) = \sum_{i=0}^{n-1} \left\lceil \frac{d_H(X_i, Y_i)}{e} \right\rceil e,$$

where $d_H(X_i, Y_i)$ shows the Hamming distance between the i-th b-bit bytes X_i and Y_i, $1 \leq e \leq b$, and $\lceil A \rceil$ expresses the minimum integer greater than or equal to A. □

Theorem 9.19 *The function $D_e(\boldsymbol{X}, \boldsymbol{Y})$ satisfies the distance metrics, that is:*

1. $D_e(\boldsymbol{X}, \boldsymbol{Y}) > 0$ *for* $\boldsymbol{X} \neq \boldsymbol{Y}$ *and* $D_e(\boldsymbol{X}, \boldsymbol{Y}) = 0$ *for* $\boldsymbol{X} = \boldsymbol{Y}$,
2. $D_e(\boldsymbol{X}, \boldsymbol{Y}) = D_e(\boldsymbol{Y}, \boldsymbol{X})$,
3. $D_e(\boldsymbol{X}, \boldsymbol{Y}) \leq D_e(\boldsymbol{X}, \boldsymbol{Z}) + D_e(\boldsymbol{Z}, \boldsymbol{Y})$.

Theorem 9.19 can be easily proved, and so the proof is omitted.

Definition 9.6 The following shows the function for the vectors $\boldsymbol{X} = (X_0,\ X_1,\ \cdots,\ X_{n-1})$ and $\boldsymbol{Y} = (Y_0,\ Y_1,\ \cdots,\ Y_{n-1})$ defined by

$$\gamma_{f_1}^{f_2}(\boldsymbol{X}, \boldsymbol{Y}) = |\,\{i \,|\, f_1 \leq d_H(X_i, Y_i) \leq f_2, \quad 0 \leq f_1 < f_2 \leq b\}\,|,$$

where $|A|$ expresses the number of elements in set A. □

9.6.2 Code Conditions

By using the defined functions, $D_e(\boldsymbol{X}, \boldsymbol{Y})$ and $\gamma_{f_1}^{f_2}(\boldsymbol{X}, \boldsymbol{Y})$, we obtain the necessary and sufficient conditions of the error locating codes as well as the error correcting / detecting codes. We consider the error control codes of the generalized form, such as t e/b-errors correcting codes (i.e., $t_{e/b}$EC codes), m e/b-errors detecting codes (i.e., $m_{e/b}$ED codes), l e/b-errors locating codes (i.e., $l_{e/b}$EL codes), and the codes with combination of these functions, where t, m, and l are integers.

Conditions for $t_{e/b}$EC Codes

Lemma 9.4 *The following shows the necessary and sufficient condition of a code that corrects any errors included in the error set* $\boldsymbol{E}$*:*

$$\boldsymbol{X} + E_x \neq \boldsymbol{Y} + E_y \quad \textit{for all } \boldsymbol{X}, \boldsymbol{Y} \in C,\ \boldsymbol{X} \neq \boldsymbol{Y},\ \textit{and for all } E_x, E_y \in \boldsymbol{E}.$$

Theorem 9.20 *The necessary and sufficient conditions of the $t_{e/b}$EC codes that correct t e/b-errors in a word are shown as*

$$\gamma_{2e+1}^{b}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1 \quad \textit{or} \quad D_e(\boldsymbol{X}, \boldsymbol{Y}) \geq 2te + 1.$$

Proof Let E_{x_i} and E_{y_i} be the i-th bytes of E_x and E_y, respectively, where $0 \leq i \leq n-1$. If $\gamma_{2e+1}^{b}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1$, then there exists the byte (e.g., the i-th byte) that satisfies $d_H(X_i, Y_i) \geq 2e + 1$. In this case, even if there exist e/b-errors at the i-th byte of both $\boldsymbol{X}$ and $\boldsymbol{Y}$, the resulting i-th bytes of these vectors are not coincident. By Lemma 9.4, the code that satisfies $\gamma_{2e+1}^{b}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1$ is a $t_{e/b}$EC code.

Next let $\boldsymbol{E}_{t_{e/b}}$ be the error set of t or fewer e/b-errors. By Theorem 9.19,

$$D_e(\boldsymbol{X}, \boldsymbol{Y}) \leq D_e(\boldsymbol{X}, \boldsymbol{X} + E_x) + D_e(\boldsymbol{X} + E_x, \boldsymbol{Y} + E_y) + D_e(\boldsymbol{Y}, \boldsymbol{Y} + E_y).$$

For $D_e(\boldsymbol{X}, \boldsymbol{Y}) \geq 2te + 1$,

$$\begin{aligned} D_e(\boldsymbol{X} + E_x, \boldsymbol{Y} + E_y) &\geq D_e(\boldsymbol{X}, \boldsymbol{Y}) - D_e(\boldsymbol{X}, \boldsymbol{X} + E_x) - D_e(\boldsymbol{Y}, \boldsymbol{Y} + E_y) \\ &\geq 1. \end{aligned}$$

Then $\boldsymbol{X} + E_x \neq \boldsymbol{Y} + E_y$ is always satisfied. Therefore the code that satisfies $D_e(\boldsymbol{X}, \boldsymbol{Y}) \geq 2te + 1$ is a $t_{e/b}$EC code.

Conversely, if $\gamma_{2e+1}^{b}(\boldsymbol{X}, \boldsymbol{Y}) = 0$ and $D_e(\boldsymbol{X}, \boldsymbol{Y}) < 2te + 1$, then $0 \leq d_H(X_i, Y_i) \leq 2e$ for any bytes of $\boldsymbol{X}$ and $\boldsymbol{Y}$. Let w be the number of bytes that satisfy $e + 1 \leq d_H(X_i, Y_i) \leq 2e$. Then the number of bytes that satisfy $1 \leq d_H(X_i, Y_i) \leq e$ is at most $2t - 2w$. In this case we can select E_x and E_y to satisfy the following: (1) $E_{x_i} \neq 0, E_{y_i} \neq 0$ and $X_i + E_{x_i} = Y_i + E_{y_i}$ for the bytes having the relation $e + 1 \leq d_H(X_i, Y_i) \leq 2e$, (2) $E_{x_i} = X_i + Y_i,\ E_{y_i} = 0$ for the $(t - w)$ bytes having the relation $1 \leq d_H(X_i, Y_i) \leq e$ and $E_{x_i} = 0, E_{y_i} = X_i + Y_i$ for the remaining bytes. Then we have $\boldsymbol{X} + E_x = \boldsymbol{Y} + E_y$ for all $E_x, E_y \in \boldsymbol{E}_{t_{e/b}}$. Therefore, this does not satisfy the condition of the $t_{e/b}$EC code. Q.E.D.

Conditions for $m_{e/b}$ED Codes

Lemma 9.5 *The following shows the necessary and sufficient conditions of a code* $\boldsymbol{C}$ *that detects any errors included in the error set* $\boldsymbol{E}$*:*

$$\boldsymbol{X} + E_x \neq \boldsymbol{Y} \qquad \text{for all} \quad \boldsymbol{X}, \boldsymbol{Y} \in \boldsymbol{C},\ \boldsymbol{X} \neq \boldsymbol{Y},\ \text{and for all } E_x \in \boldsymbol{E}.$$

Theorem 9.21 *The necessary and sufficient conditions of the* $m_{e/b}$*ED codes that detect* m e/b*-errors in a word are*

$$\gamma^b_{e+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1 \quad or \quad D_e(\boldsymbol{X}, \boldsymbol{Y}) \geq me + 1.$$

Theorem 9.21 can be proved in the same manner as the previous theorem.

Conditions for $l_{e/b}$EL Codes

Lemma 9.6 *The following shows the necessary and sufficient conditions of a code* $\boldsymbol{C}$ *that locates any errors included in the error set* $\boldsymbol{E}$*:*

1. $\boldsymbol{X} + E_x = \boldsymbol{Y} + E_y \Rightarrow (E_{x_i} = 0 \wedge E_{y_i} = 0)$ *or* $(E_{x_i} \neq 0 \wedge E_{y_i} \neq 0)$ *for* $0 \leq i \leq n - 1$.
2. $\boldsymbol{X} + E_x \neq \boldsymbol{Y}$,

where $\forall \boldsymbol{X}, \forall \boldsymbol{Y} \in \boldsymbol{C},\ \boldsymbol{X} \neq \boldsymbol{Y},\ \forall E_x, \forall E_y \in \boldsymbol{E}$, *and* E_{x_i}, E_{y_i} *express the* i*-th byte errors in* E_x, E_y, *respectively.*

Lemma 9.6 can be roughly proved as follows: Condition 1 tells us that the relation $\boldsymbol{X} + E_x = \boldsymbol{Y} + E_y$ is satisfied only if the location of the byte errors in E_x is the same as that of the errors in E_y. In this case the errors cannot be corrected, but their locations can be correctly indicated. Condition 2 tells us that all errors in $\boldsymbol{E}$ are detected. Therefore both conditions 1 and 2 are the necessary and sufficient conditions of the error locating code.

Theorem 9.22 *The necessary and sufficient conditions of the* $l_{e/b}$*EL codes that locate* l e/b*-errors in the word are*

$$\begin{aligned} &\gamma^b_{2e+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1, \\ &\gamma^e_1(\boldsymbol{X}, \boldsymbol{Y}) = 0, \quad or \\ &D_e(\boldsymbol{X}, \boldsymbol{Y}) \geq 2le + 1. \end{aligned}$$

Proof By Theorem 9.20, the condition $\gamma^b_{2e+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1$ or $D_e(\boldsymbol{X}, \boldsymbol{Y}) \geq 2le + 1$ shows the necessary and sufficient condition of the $l_{e/b}$EC codes, and therefore this is also the condition of the $l_{e/b}$EL codes.

If the condition $\gamma^e_1(\boldsymbol{X}, \boldsymbol{Y}) = 0$ is satisfied for the vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$, there exists the possibility to have the relation $\boldsymbol{X} + E_x = \boldsymbol{Y} + E_y$ for $\exists E_x$ and $\exists E_y$, where $d_H(X_i, Y_i)$ is no less than $e + 1$ or zero. This shows that $E_{x_i} \neq 0$ and $E_{y_i} \neq 0$ for the erroneous i-th bytes in $\boldsymbol{X}$ and $\boldsymbol{Y}$ with relation $d_H(X_i, Y_i) \geq e + 1$, or $E_{x_i} = E_{y_i}$ for the i-th bytes in $\boldsymbol{X}$ and $\boldsymbol{Y}$ that satisfy $d_H(X_i, Y_i) = 0$, where E_{x_i} and E_{y_i} show the i-th byte of E_x and E_y, respectively. By Lemma 9.6, the above satisfies the conditions of the $l_{e/b}$EL codes.

Conversely, we consider the conditions $\gamma^b_{2e+1}(\boldsymbol{X}, \boldsymbol{Y}) = 0$, $\gamma^e_1(\boldsymbol{X}, \boldsymbol{Y}) \geq 1$, and $D_e(\boldsymbol{X}, \boldsymbol{Y}) < 2le + 1$. In this case there always exist the bytes that satisfy $1 \leq d_H(X_i, Y_i) \leq e$. For these bytes we can select the errors E_x and E_y in the error set

comprising of l or less e/b-errors where $\boldsymbol{X} + E_x = \boldsymbol{Y} + E_y$ for $E_{x_i} \neq 0$ and $E_{y_i} = 0$. Therefore, they do not satisfy the conditions of Lemma 9.6. Q.E.D.

Conditions for Codes with Combination of Code Functions Here we consider the codes with two code functions such as error correction and error detection, error correction and error location, and error location and error detection.

Table 9.3 demonstrates the necessary and sufficient conditions of the $t_{e_1/b}$EC-$m_{e_3/b}$ED codes, the $t_{e_1/b}$EC-$l_{e_2/b}$EL codes, and the $l_{e_2/b}$EL-$m_{e_3/b}$ED codes, where $t < l < m$. The conditions of the $t_{e_1/b}$EC-$l_{e_2/b}$EL codes, for example, can be obtained by taking logical AND of the following conditions:

1. Conditions of the $t_{e_1/b}$EC codes,
2. Conditions of the $l_{e_2/b}$EL codes,
3. Conditions to discriminate the t e_1/b-errors and the l e_2/b-errors.

This table also includes the conditions of the $t_{e/b}$EC codes, the $m_{e/b}$ED codes and the $l_{e/b}$EL codes given in Theorems 9.20, 9.21, and 9.22, respectively.

TABLE 9.3 Necessary and Sufficient Conditions of Codes

Codes	Conditions
$t_{e/b}$EC code	$\gamma^b_{2e+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1$, or $D_e(\boldsymbol{X}, \boldsymbol{Y}) \geq 2te + 1$
$l_{e/b}$EL code	$\gamma^b_{2e+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1, \gamma^e_1(\boldsymbol{X}, \boldsymbol{Y}) = 0$, or $D_e(\boldsymbol{X}, \boldsymbol{Y}) \geq 2le + 1$
$m_{e/b}$ED code	$\gamma^b_{e+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1$, or $D_e(\boldsymbol{X}, \boldsymbol{Y}) \geq me + 1$
$t_{e_1/b}$EC-$m_{e_3/b}$ED code	$\gamma_{e^b_1+e^*_{1,3}+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1$, $(D_{e_1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 2te_1 + 1 \wedge \gamma^b_{e_3}(\boldsymbol{X}, \boldsymbol{Y}) \geq t + 1)$, $D_{e^*_{1,3}}(\boldsymbol{X}, \boldsymbol{Y}) \geq (t + m)e^*_{1,3} + 1$, $(\gamma^b_{e_1+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq m + 1 \text{ if } e_1 \leq e_3)$, or $(D_e(\boldsymbol{X}, \boldsymbol{Y}) \geq 2te_1 + 1 \wedge \gamma^b_{e_1+e_3+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1 \text{ if } e_1 > e_3)$
$t_{e_1/b}$EC-$l_{e_2/b}$EL code	$\gamma^b_{2e^*_{1,2}+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1$, $(D_{e_1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 2te_1 + 1 \wedge \gamma^b_{e_2+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq t + 1 \wedge \gamma^{e_2}_1(\boldsymbol{X}, \boldsymbol{Y}) = 0)$, $(D_{e_1}(\boldsymbol{X}, \boldsymbol{Y}) \geq (t + l)e_1 + 1 \wedge D_{e_2}(\boldsymbol{X}, \boldsymbol{Y}) \geq 2le_2 + 1)$, $(\gamma^{e_2}_1(\boldsymbol{X}, \boldsymbol{Y}) = 0 \wedge \gamma^b_{e_1+e_2+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1 \text{ if } e_1 \leq e_2)$, $(D_{e_1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 2te_1 + 1 \wedge \gamma^b_{e_1+e_2+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1 \text{ if } e_1 > e_2)$, $\{D_{e_1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 2te_1 + 1 \wedge \gamma^b_{e_2+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq t + 1 \wedge (\gamma^b_{2e_2+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1$, or $D_{e_2}(\boldsymbol{X}, \boldsymbol{Y}) \geq 2le_2 + 1) \text{ if } e_1 > e_2\}$, or $\{D_{e_1}(\boldsymbol{X}, \boldsymbol{Y}) \geq (t + l)e_1 + 1 \wedge (\gamma^b_{2e_2+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1 \text{ or } \gamma^{e_2}_1(\boldsymbol{X}, \boldsymbol{Y}) = 0) \text{ if } e_1 > e_2\}$
$l_{e_2/b}$EL-$m_{e_3/b}$ED code	$\gamma^b_{e_2+e^*_{2,3}+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1$, $(D_{e_2}(\boldsymbol{X}, \boldsymbol{Y}) \geq 2le_1 + 1 \wedge \gamma^b_{e_3+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq l + 1)$, $\gamma^{e^*_{2,3}}_1(\boldsymbol{X}, \boldsymbol{Y}) = 0$, $D_{e^*_{2,3}}(\boldsymbol{X}, \boldsymbol{Y}) \geq (l + m)e^*_{2,3} + 1$, $(\gamma^b_{e_2+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq m + 1 \text{ if } e_2 \leq e_3)$, or $(D_{e_2}(\boldsymbol{X}, \boldsymbol{Y}) \geq 2le_2 + 1 \wedge \gamma^b_{e_2+e_3+1}(\boldsymbol{X}, \boldsymbol{Y}) \geq 1 \text{ if } e_2 > e_3)$

Source: *[KITA97]. © 1997 IEICE Japan.*

Note: $* : e_{i,j} = \max(e_i, e_j) = \begin{cases} e_i & \text{if } e_i \geq e_j, \\ e_j & \text{if } e_i < e_j. \end{cases}$

TABLE 9.4 Code Conditions of Existing Codes

Codes	Code conditions	t	l	m	e_1	e_2	e_3
SEC code	$D_1(\boldsymbol{X}, \boldsymbol{Y}) \geq 3$	1	—	—	1	—	—
DED code	$D_1(\boldsymbol{X}, \boldsymbol{Y}) \geq 3$	—	—	1 2	—	—	2^* 1
SEC-DED code	$D_1(\boldsymbol{X}, \boldsymbol{Y}) \geq 4$	1	—	1 2	1	—	2^* 1
t_bEC-m_bED code	$D_b(\boldsymbol{X}, \boldsymbol{Y}) \geq (t+m)b+1$	t	—	m	b	—	b
SEC-SbED code	$\gamma_2^b(\boldsymbol{X}, \boldsymbol{Y}) \geq 2$, or $D_b(\boldsymbol{X}, \boldsymbol{Y}) \geq 2b+1$	1	—	1	1	—	b
SEC-DED-SbED code	$\gamma_2^b(\boldsymbol{X}, \boldsymbol{Y}) \geq 2$, $(D_b(\boldsymbol{X}, \boldsymbol{Y}) \geq 2b+1 \wedge \gamma_3^b(\boldsymbol{X}, \boldsymbol{Y}) \geq 1)$, or $(D_b(\boldsymbol{X}, \boldsymbol{Y}) \geq 2b+1 \wedge D_1(\boldsymbol{X}, \boldsymbol{Y}) \geq 4)$	1	—	1 2 1	1	—	2^* 1 b
SbEC-DED code	$D_b(\boldsymbol{X}, \boldsymbol{Y}) \geq 3b+1$, or $(\gamma_2^b(\boldsymbol{X}, \boldsymbol{Y}) \geq 2 \wedge D_b(\boldsymbol{X}, \boldsymbol{Y}) \geq 2b+1)$	1	—	2	1	—	1
SEC-S$_{e/b}$EL code	$\gamma_{2e+1}^b(\boldsymbol{X}, \boldsymbol{Y}) \geq 1$, $(\gamma_{e+2}^b(\boldsymbol{X}, \boldsymbol{Y}) \geq 1 \wedge \gamma_1^e(\boldsymbol{X}, \boldsymbol{Y}) = 0)$, $(\gamma_2^b(\boldsymbol{X}, \boldsymbol{Y}) \geq 2 \wedge \gamma_1^e(\boldsymbol{X}, \boldsymbol{Y}) = 0)$, or $D_e(\boldsymbol{X}, \boldsymbol{Y}) \geq 2e+1$	1	1	—	1	e	—
SEC-DED-S$_{e/b}$EL code	$\gamma_{2e+1}^b(\boldsymbol{X}, \boldsymbol{Y}) \geq 1$, $D_e(\boldsymbol{X}, \boldsymbol{Y}) \geq 3e+1$, $(\gamma_1^e(\boldsymbol{X}, \boldsymbol{Y}) = 0 \wedge \gamma_{e+2}^b(\boldsymbol{X}, \boldsymbol{Y}) \geq 1)$, or $\{D_e(\boldsymbol{X}, \boldsymbol{Y}) \geq 2e+1 \wedge (\gamma_{e+2}^b(\boldsymbol{X}, \boldsymbol{Y}) \geq 1,$ $\gamma_1^e(\boldsymbol{X}, \boldsymbol{Y}) = 0$, or $\gamma_2^b(\boldsymbol{X}, \boldsymbol{Y}) \geq 2)\}$	1	1	1 2	1	e	2^* 1
$S_{b/p\times b}$EL code	$\gamma_3^p(\boldsymbol{X}, \boldsymbol{Y}) \geq 1, \gamma_1^1(\boldsymbol{X}, \boldsymbol{Y}) = 0$, or $D_p(\boldsymbol{X}, \boldsymbol{Y}) \geq 2p+1$	—	1	—	—	1	—
SbEC-S$_{i\times b/p\times b}$ED code	$\gamma_{i+2}^p(\boldsymbol{X}, \boldsymbol{Y}) \geq 1, \gamma_{i+1}^p(\boldsymbol{X}, \boldsymbol{Y}) \geq 2$, or $D_i(\boldsymbol{X}, \boldsymbol{Y}) \geq 2i+1$	1	—	1	1	—	i

Source: *[KITA97]. © 1997 IEICE Japan.*
Note: The $*$ means that the code conditions of the DED code, for example, are induced by taking logical ANDing the conditions of the $m_{e_3/b}$ED codes given by substituting two cases of $(m=1, e_3=2)$ and $(m=2, e_3=1)$.

Table 9.4 shows the conditions of the existing codes induced by substituting appropriate values to the parameters of t, l, m, e_1, e_2, and e_3 of the codes in Table 9.3.

9.6.3 Relation between Error Locating Codes and Error Correcting / Detecting Codes

The necessary and sufficient conditions of the error locating codes include the condition of $\gamma_1^e(\boldsymbol{X}, \boldsymbol{Y}) = 0$ in addition to the conditions of the error correcting codes. The necessity of including this condition makes the number of codewords of error locating codes greater than that of the error correcting codes. The same is true for the relation between the $l_{e_2/b}$EL-$m_{e_3/b}$ED codes and the $t_{e_1/b}$EC-$m_{e_3/b}$ED codes.

As the value of e becomes equal to b, the number of codewords that satisfy $\gamma_1^e(\boldsymbol{X}, \boldsymbol{Y}) = 0$ becomes small, and therefore the error locating codes has to be equal to the error correcting codes. If $e = b$, the following theorem holds.

Theorem 9.23 *If $e = b$, an $l_{e/b}$EL code is an $l_{e/b}$EC code.*

Theorem 9.2 indicates a particular case of the theorem above where $l = 1$.

The number of bytes in the vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$, where X_i and Y_i are i-th bytes of $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively, that satisfy $d_H(X_i, Y_i) \geq e + 1$, can be expressed as

$$\sum_{j=e+1}^{b} \binom{b}{j}.$$

As the value of e becomes equal to b, the value above rapidly becomes small, and therefore the error locating codes rapidly become equal to the error correcting codes. So we have the following theorem on the relation between the error locating codes and the error detecting codes.

Theorem 9.24 *If byte length is equal to code length, an error locating code is an error detecting code.*

Proof By Theorems 9.21 and 9.22, the conditions of the $l_{e/b}$EL codes and the $m_{e/b}$ED codes are expressed as $\gamma_1^e(\boldsymbol{X}, \boldsymbol{Y}) = 0$ and $\gamma_{e+1}^b(\boldsymbol{X}, \boldsymbol{Y}) \geq 1$, respectively. If there exists only one byte in the codeword, it is apparent that these conditions are equivalent, and therefore an error locating code is an error detecting code. Q.E.D.

EXERCISES

9.1 Derive the bound expressed in (9.1).

9.2 Let $\boldsymbol{C_D}$ be a binary (5, 1) quadruple-bit error detecting code generated by the polynomial $\mathbf{g}(x) = x^4 + x^3 + x^2 + x + 1$, and let C_c be a (7, 5) single-symbol error correcting code over the field generated by the polynomial $\mathbf{g}(x)$. Design the $\mathbf{H}$ matrix of the (35, 27) $S_{4/5}$EL code using the tensor product of the $\mathbf{H}$ matrices of the $\boldsymbol{C_D}$ and $\boldsymbol{C_c}$ codes.

9.3 Prove Theorem 9.2.

9.4 Prove that the SEC-$S_{b/p\times b}$EL code of design method I can locate an erroneous block that includes single b-bit burst error.

9.5 Design the $\mathbf{H}$ matrix of the SEC-$S_{2/3\times 2}$EL code of design method I with $R = 6$ bits.

9.6 Prove Lemma 9.1. Obtain the eight nonsingular odd-weight 4×4 square matrices.

9.7 Design two (115, 105) SEC-$S_{3/7\times 3}$EL codes by applying the design methods of I and II.

9.8 The following $\mathbf{H}$ matrix shows a (36, 30) SEC-$S_{3/4\times 3}$EL code of design type II.

$$\mathbf{H} = \left[\begin{array}{cccc|cccc|cccc} 000 & 111 & 000 & 111 & 100 & 100 & 100 & 100 & 100 & 100 & 100 & 100 \\ 000 & 111 & 111 & 000 & 010 & 010 & 010 & 010 & 010 & 010 & 010 & 010 \\ 000 & 000 & 111 & 111 & 001 & 001 & 001 & 001 & 001 & 001 & 001 & 001 \\ \hline 100 & 100 & 100 & 100 & 000 & 111 & 000 & 111 & 100 & 001 & 011 & 110 \\ 010 & 010 & 010 & 010 & 000 & 111 & 111 & 000 & 010 & 101 & 010 & 101 \\ 001 & 001 & 001 & 001 & 000 & 000 & 111 & 111 & 001 & 011 & 110 & 100 \end{array}\right].$$

(a) The following received words W_1 and W_2 have a single-bit error and a single-byte error, respectively. Locate or correct the errors in W_1 and W_2 by the code shown above.

$$W_1 = [\ 101\ \ 011\ \ 100\ \ 011\ |\ 000\ \ 010\ \ 100\ \ 010\ |\ 101\ \ 101\ \ 010\ \ 111\ ]$$
$$W_2 = [\ 011\ \ 010\ \ 100\ \ 110\ |\ 011\ \ 110\ \ 101\ \ 110\ |\ 101\ \ 001\ \ 011\ \ 100\ ]$$

(b) Design the error locating circuit of the code.

9.9 Prove Theorem 9.9.

9.10 Prove Theorem 9.11.

9.11 Prove Theorem 9.13.

9.12 Find the maximum code length (in bits) of the SEC-$S_{(b-2)/b}$EL code having $2b - 2$ check bits.

9.13 Design the **H** matrix of a (136, 128) SEC-$S_{2/8}$EL code.

9.14 Design a (40, 32) SEC-B_3EL code and its decoder circuit.

9.15 Derive the necessary and sufficient conditions of the $t_{e_1/b}$EC-$l_{e_2/b}$EL code and the $l_{e_2/b}$EL-$m_{e_3/b}$ED code.

REFERENCES

[BOSE92] B. Bose and S. Al-Bassam, "Byte Unidirectional Error Correcting and Detecting Codes," *IEEE Trans. Comput.*, C-41 (December 1992): 1601–1606.

[CHAN65] S.-H. Chang and L.-J. Weng, "Error Locating Codes," *IEEE Int. Conv. Rec.*, Pt. 7 (1965): 252–258.

[DASS82] B. K. Dass, "Burst Error Locating Linear Codes," *J. Info. Optimization Sci.*, 3 (January 1982): 77–80.

[DUNN89] L. A. Dunning, G. Dial, and M. R. Varanasi, "Unidirectional 9-bit Byte Error Detecting Codes for Computer Memory Systems," *Dig. 19th IEEE Int. Symp. Fault-Tolerant Computing*, FTCS-19 (June 1989): 216–221.

[FIRE59] P. Fire, "A Class of Multiple-Error Correcting Binary Codes for Non-independent Errors," *Sylvania Report*, RSL-E-2 (1959).

[FUJI94] E. Fujiwara and M. Kitakami, "A Class of Error-Locating Codes for Byte-Organized Memory Systems," *IEEE Trans. Info. Theory*, 40 (November 1994): 1857–1865.

[GOET67] J. M. Goethals, "Cyclic Error-Locating Codes," *Info. Contr.*, 10 (1967): 378–385.

[HONG72] S. J. Hong and A. M. Patel, "A General Class of Maximal Codes for Computer Applications," *IEEE Trans. Comput.*, C-21 (December 1972): 1322–1331.

[KAIS02] T. Kaise and M. Kitakami, "Single-Bit Error Correcting and Burst Error Locating Codes," *Proc. 2002 IEEE Int. Symp. on Information Theory* (June 2002): 117.

[KASA63] T. Kasami, "Optimum Shortened Cyclic Codes for Burst Error Correction," *IEEE Trans. Info. Theory*, IT-9 (April 1963): 105–109.

[KITA95] M. Kitakami and E. Fujiwara, "A Class of Error Locating Codes—SEC-$S_{e/b}$EL codes—," *IEICE Trans. Fundamentals*, E78-A (September 1995): 1086–1091.

[KITA97] M. Kitakami, S. Jiang, and E. Fujiwara, ''Metrics of Error Locating Codes,'' *IEICE Trans. Fundamentals*, E80-A (November 1997): 2117–2122.

[KITA98] M. Kitakami and E. Fujiwara, ''A Class of Burst Error Locating Codes,'' *Proc. IEEE Int. Symp. Information Theory and Its Applications* (October 1998): 435–438.

[KITA05] M. Kitakami and J. Sano, ''Code Design and Decoding Method for Burst Error Locating Codes,'' *IEEE Pacific Rim Int. Symp. on Dependable Computing* (2005): 125–132.

[MCWL77] F. J. Mcwilliams and N. J. A. Sloan, *The Theory of Error Correcting Codes*, North-Holland (1977).

[PETE72] W. W. Peterson and E. J. Weldon Jr., *Error-Correcting Codes*, 2d ed., MIT Press (1972).

[TO89] L. P. To and K. Sakaniwa, ''A Discussion on *m*-Ary Error Discriminating Codes,'' *Dig. 12th IEEE Symp. Information Theory and Its Application* (December 1989): 121–125.

[VAID92] N. H. Vaidya and D. K. Pradhan, ''A New Class of Bit- and Byte Error Control Codes,'' *IEEE Trans. Info. Theory*, IT-38 (September 1992): 1617–1623.

[WOLF63] J. K. Wolf and B. Elspas, ''Error-Locating Codes—A New Concept in Error Control,'' *IEEE Trans. Info. Theory*, IT-9 (April 1963): 113–117.

[WOLF65a] J. K. Wolf, ''On Codes Derivable from the Tensor Product of Check Matrices,'' *IEEE Trans. Info. Theory*, IT-11 (April 1965): 281–284.

[WOLF65b] J. K. Wolf, ''On an Extended Class of Error-Locating Codes,'' *Info. Contr.*, 8 (1965): 163–169.

CONTENTS

10

Codes for Unequal Error Control / Protection (UEC / UEP)

In almost all applications the error control capability of a code is characterized by the probability of correct reception of the information over the entire codeword, where every position in the codeword requires an equal error control level against errors. We find, however, that some applications of these coding systems leave some positions in a codeword with a higher error rate than others [LO96]. Some such situations may be caused by low reliability devices or by certain error-sensitive positions in a codeword that are vulnerable to external noises or have a low noise margin. Further some types of computer words or communication messages have a structure whereby the information included in a part of the word is more important or more valuable than that in other parts of the word [MASN67]. Control and address information in computer / communication messages and pointer information in database words are good examples. In these cases any errors in the control information or in the pointer information cause serious damage to the subsequent processes in the system.

This chapter presents a new class of codes, called *unequal error control (UEC) codes* and *unequal error protection (UEP) codes*, that has different error control levels in a codeword such that some part of the word is more strongly controlled from errors than other parts [FUJI98, FUJI95, MASN67]. The discussion is organized as follows: Error models for the UEC codes and the UEP codes are clarified first. Then this chapter demonstrates some types of UEC codes and UEP codes and their applications to holographic memories and to lossless compressed text data, respectively.

10.1 ERROR MODELS FOR UEC CODES AND UEP CODES

Before proceeding to the UEC codes, we consider in this section a similar class of codes called *unequal error protection (UEP) codes* [MASN67]. These codes are designed

Code Design for Dependable Systems: Theory and Practical Applications, by Eiji Fujiwara

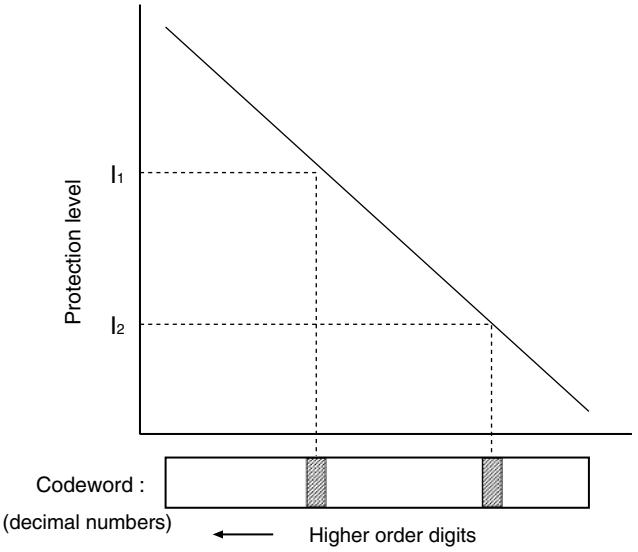


Figure 10.1 *Unequal error protection (UEP) model in numeral codeword (decimal numbers).*

on the concept that some information digits in a codeword are protected against a higher number of errors than other information digits. For example, during the processing of digital data using conventional decimal numbers or measurement data, errors in the higher order digits of numbers can yield more serious effects on the subsequent processes in the digital systems than errors in the lower order digits. The UEP codes are defined such that some of the digits in a codeword, which should be strongly protected, are correctly decoded if I_1 or fewer errors occur, and others are correctly decoded only if I_2 or fewer errors occur, where $I_1 > I_2$. Figure 10.1 shows this model. The UEP codes have the property that the strongly protected area should not be miscorrected by I_1 errors occurring outside the protected area. I_1 errors that occur inside the strongly protected area errors should all be corrected.

In this chapter we will later deal mainly with a different class of error control codes from the UEP codes. Unlike the UEP codes, the codes have distinct error control levels in a codeword such that some part of the word is more strongly controlled from errors than other parts. The binary codewords used in computer systems won't necessarily require different error control level in each bit of the words. As we will see, binary codewords rather require uniform level in the clustered bit positions in the word. These positions are called a *byte*, and as these positions in the word are determined in advance, they are called a *fixed-byte*. In general, the codeword is assumed to be constructed from some fixed-bytes with different error control levels. In order to apply an error control code to tolerate errors occurring in this type of computer / communication word, we must transform the error control level into a code function. The higher the level, the stronger the code function that must be applied. Figure 10.2 gives an example of this unequal error control model with two levels in the codeword. This UEC model is much simpler than the existing model shown in Figure 10.1, and therefore we will use this model to design more practical and simpler codes for computer / communication systems.

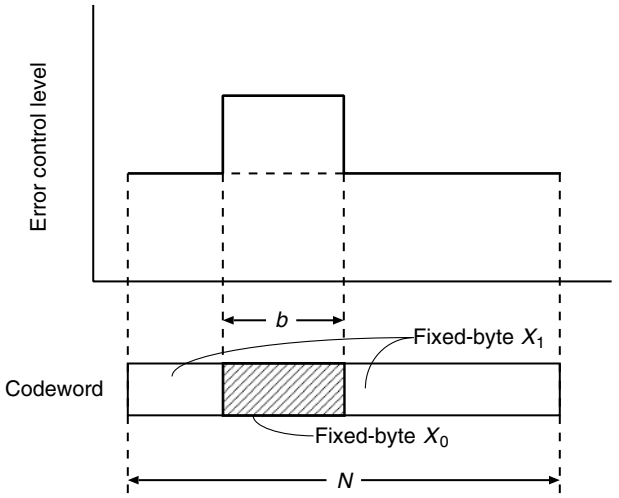


Figure 10.2 *Two-level unequal error control (2-level UEC) model in codeword.* Source: *[FUJI98]. © 1998 IEEE.*

In generalized form, the definitions of a class of UEC codes are presented as follows.

Definition 10.1 Let X be a codeword divided into L fixed-bytes, $X_0, X_1, \cdots, X_{L-1}$, and let N_i be the length of the i-th fixed-byte X_i, where N_i's are not necessarily equal to each other. Also let F_i be the error control function in X_i. The *L-level UEC code* is defined as the one with $F_i \neq F_j$ for $0 \leq i \neq j \leq L-1$, and it is expressed as $F_0|F_1|\cdots|F_{L-1}$ *code* with code length $N = N_1 + N_2 + \cdots + N_{L-1}$. □

If we consider random bit error correction and detection, the error control function F_i can be expressed by t_i-bit error correction and d_i-bit error detection, where $t_i < d_i \leq N_i$. In this case the error control level F_i is measured by the minimum Hamming distance of the code with error control level F_i (i.e., $t_i + d_i + 1$), and therefore F_i is stronger than F_j if and only if $t_i + d_i + 1 > t_j + d_j + 1$.

Definition 10.2 Let X be a codeword that is divided according to Definition 10.1, and let F_{ij} be the error control function against errors occurring in X_i and X_j simultaneously. The code with functions F_{ij}, $0 \leq i \neq j \leq L-1$, and F_k, $0 \leq k \neq i,j \leq L-1$, is also defined as the UEC code. □

Definition 10.2 can be extended to the code with error control functions against errors occurring in more than two fixed-bytes simultaneously.

In this chapter we have some optimal 2-level UEC codes, for example, *Fixed b-bit byte Error Correcting | Single-bit Error Correcting (FbEC | SEC) codes* and *Fixed b-bit byte error correcting | Single-bit Error Correcting and Double-bit Error Detecting (FbEC | SEC-DED) codes*. The former codes have two error control functions in a codeword: the stronger one is a fixed-byte error correction (F*b*EC) that corrects any error in the fixed-byte X_0, and the other is a single-bit error correction (SEC) in X_1. The latter codes have also two error control functions in a codeword, and in particular, the area X_1 has the SEC-DED code function.

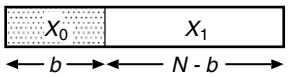


Figure 10.3 *Codeword with 2-UEC levels.*

From the previous definitions, the FbEC|SEC code is a 2-level UEC code with $N_0 = b$ and $N_1 = N - b$, where F_0 is a b-bit byte error correction in X_0 and F_1 is a single-bit error correction in X_1. The FbEC|SEC-DED code is also a 2-level UEC code with $N_0 = b$ and $N_1 = N - b$ that has an error control function F_0 of correcting b-bit byte errors in X_0, and also F_1 of correcting single-bit errors and detecting double-bit errors in X_1.

Let $\boldsymbol{C}$ be a code whose codewords consist of two parts as shown in Figure 10.3: the fixed-byte area X_0 and the remaining fixed-byte area X_1. Without loss of generality, we assume that X_0 always precedes X_1 in a codeword. Since check bits are not always used in a decoded output, we assume that the whole check bits are included in X_1.

Let us recall the relation between the UEC codes and the UEP codes. If the errors occurring outside the strongly controlled area X_0 in the codeword are detected and are not miscorrected as errors in X_0, then the UEC codes approach in function the UEP codes. In the case of the FbEC|SEC codes, for example, if a b-bit burst error has occurred in X_1 and is very highly detected, then the codes can be regarded as fixed-byte error protection codes. The same may be said about applications of the UEC codes with multiple-levels and their relation to existing UEP codes.

It easily follows that many types of 2-level UEC codes can be designed by combining the basic code functions of single-bit error correction (SEC), double-bit error detection (DED), fixed b-bit byte error detection (FbED), fixed b-bit byte error correction (FbEC), byte plus single-bit error detection ((Fb+S)ED), and byte plus single-bit error correction ((Fb+S)EC). Here, the code functions of (A + B)ED and (A + B)EC must allow the code to detect A errors and B errors simultaneously, and correct A errors and B errors simultaneously, respectively; that is, '(A + B)E' means that A errors and B errors occur simultaneously. Figure 10.4 provides an overview of the 2-level UEC

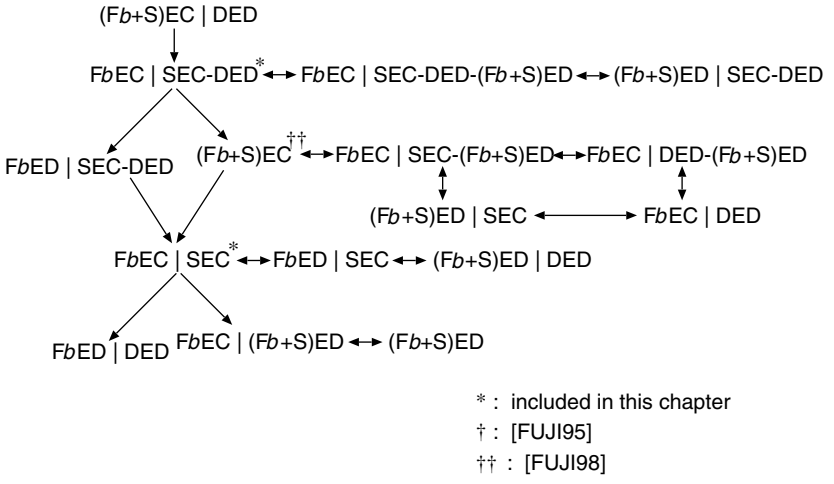


Figure 10.4 *Overview of basic two-level unequal error control functions.* Source: *[FUJI98]. © 1998 IEEE.*

codes combined with the indicated pieces of basic code functions [FUJI98, RITT96]. In this figure the single-barb arrow (→) means that the upper code function includes the lower one, and the double-barb arrow (↔) means that the code functions at both ends are equivalent. For example, the F*b*EC|SEC-DED code function includes the (F*b*+S)EC code function as well as the F*b*ED|SEC-DED code function. It is also equivalent to the code function of F*b*EC|SEC-DED-(F*b*+S)ED. While we can combine some other pieces of basic code functions, from the practical stand point, the combination of the basic code functions shown in Figure 10.4 are sufficient. Note that in this figure the codes presented in this chapter are treated as the nuclei of the basic 2-level UEC codes.

10.2 FIXED-BYTE ERROR CONTROL UEC CODES

10.2.1 Optimal Fixed-Byte Error Correcting | Single-Bit Error Correcting (Optimal F*b*EC|SEC) Codes

As one of the optimal 2-level UEC codes, we discuss the fixed b-bit byte error correcting | single-bit error correcting (F*b*EC|SEC) code, which corrects b-bit byte errors in X_0 and also corrects single-bit errors in X_1.

Code Conditions and Bounds Here we consider two sets of errors, $\boldsymbol{E}_0$ and $\boldsymbol{E}_1$, where $\boldsymbol{E}_0$ is the error set caused by all possible errors in the fixed-byte X_0 (i.e., byte errors in X_0) and $\boldsymbol{E}_1$ the error set caused by single-bit errors in X_1, and then $\boldsymbol{E}_0 \cap \boldsymbol{E}_1 = \phi$, where ϕ is the empty set. The following theorem provides the necessary and sufficient condition for a linear code that can correct all patterns in $\boldsymbol{E}_0$ and also correct all error patterns in $\boldsymbol{E}_1$.

Theorem 10.1 *A binary linear code, described by the parity-check matrix* $\boldsymbol{H}$*, corrects all errors in* $\boldsymbol{E}_0$ *as well as all errors in* $\boldsymbol{E}_1$*, if and only if:*

1. $E \cdot \boldsymbol{H}^T \neq 0 \quad \text{for all } E \in \{\boldsymbol{E}_0 \cup \boldsymbol{E}_1\}$,
2. $E_i \cdot \boldsymbol{H}^T \neq E_j \cdot \boldsymbol{H}^T \quad \text{for all } E_i, E_j \in \boldsymbol{E}_1,\ E_i \neq E_j$,
3. $E_p \cdot \boldsymbol{H}^T \neq E_q \cdot \boldsymbol{H}^T \quad \text{for all } E_p, E_q \in \boldsymbol{E}_0,\ E_p \neq E_q$,
4. $E_p \cdot \boldsymbol{H}^T \neq E_i \cdot \boldsymbol{H}^T \quad \text{for all } E_p \in \boldsymbol{E}_0 \text{ and for all } E_i \in \boldsymbol{E}_1$.

Proof The theorem can be easily proved such that conditions 1 and 2 are the necessary and sufficient conditions for single-bit error correction, conditions 1 and 3 for fixed-byte error correction, and condition 4 guarantees error correction both of single-bit errors and of fixed-byte X_0 errors independently. Q.E.D.

Without loss of generality, the fixed-byte X_0 is assumed to be the area with the highest error control level and to be located at the beginning of the word, as shown in Figure 10.3. The check-bit part is assumed to be always located in the byte with the lowest error control level. Here the $\mathbf{H}$ matrix of the code is divided into two submatrices shown in Eq. (10.1). That is, $\mathbf{H}_0$ contains b columns and $\mathbf{H}_1$ contains the remaining columns in $\mathbf{H}$, which amounts to $N - b$ columns. The submatrix $\mathbf{H}_0$ shows the matrix corresponding to the fixed-byte X_0 having b-bit length, and the adjacent

submatrix $\mathbf{H}_1$ shows the remaining matrix corresponding to X_1 having $(N-b)$-bit length.

$$\mathbf{H} = [\mathbf{H}_0 \,|\, \mathbf{H}_1]. \tag{10.1}$$

Theorem 10.2 *The maximum code length (in bits) of an* $(N,\ N-r)$ *FbEC|SEC code is*

$$N_{max} = 2^r - 2^b + b.$$

Proof We consider two sets of syndromes. One syndrome is that caused by single-bit errors in X_1 and the other syndrome is that caused by byte errors in X_0. The maximum size of the former syndrome is $2^r - 1$, where r is the check-bit length. The number of all possible distinct syndromes in the fixed-byte X_0 having length b (i.e., the maximum size of the latter set) equals $\sum_{i=1}^{b} {}_bC_i = 2^b - 1$. In the F$b$EC|SEC code with code length N and check-bit length r whose $\mathbf{H}$ matrix is shown in Eq. (10.1), these two syndrome sets should be disjoint, and therefore the number of columns in $\mathbf{H}_1$ (i.e., $N-b$), satisfies the following relation:

$$N - b \le (2^r - 1) - (2^b - 1).$$

From this we have $N_{max} = 2^r - 2^b + b$. Q.E.D.

Substituting $b = 1$ in the equation shown in Theorem 10.2 gives the maximum length of the SEC codes, which is $N_{max} = 2^r - 1$. Table 10.1 lists the maximum information-bit lengths of the FbEC|SEC codes (i.e., $N_{max} - r$) for fixed-byte X_0 length b and check-bit length r.

Lemma 10.1 *In the FbEC | SEC codes the byte errors in* X_0 *and single-bit errors in* X_1 *occurring simultaneously, which are denoted as byte errors in* X_0 *plus single-bit errors in* X_1*, are not miscorrected as the errors occurring in the fixed-byte* X_0*.*

Proof If byte errors in X_0 plus single-bit errors in X_1 lead to miscorrection of fixed-byte X_0 in the codewords of the FbEC|SEC codes, there exist such errors E_i, E_p, and E_q having the following relation:

$$E_p \cdot \mathbf{H}^T + E_i \cdot \mathbf{H}^T = E_q \cdot \mathbf{H}^T, \tag{10.2}$$

TABLE 10.1 Bounds on Information-Bit Lengths of FbEC|SEC Codes

r	b							
	3	4	5	6	7	8	10	12
$b+1$	7	15	31	63	127	255	1,023	4,095
$b+2$	22	46	94	190	382	766	3,070	12,286
$b+3$	53	109	221	445	893	1,789	7,165	28,669
$b+4$	116	236	476	956	1,916	3,836	15,356	61,436
$b+5$	243	491	987	1,979	3,963	7,931	31,739	126,971

Source: [FUJI98]. © 1998 IEEE.

where E_p, E_q, $E_p \neq E_q$, are byte errors in X_0, and E_i is single-bit error in X_1. Since $E_p - E_q = E_{p'}$ is a byte error in X_0, the relation shown in Eq. (10.2) is equivalent to the following relation:

$$E_i \cdot \mathbf{H}^T + E_{p'} \cdot \mathbf{H}^T = 0.$$

This relation contradicts condition 4 of Theorem 10.1. So the fixed-byte X_0 is not miscorrected by the byte errors in X_0 plus the single-bit errors in X_1. Q.E.D.

Design for Optimal FbEC|SEC Codes

Theorem 10.3 *The following* $\boldsymbol{H}$ *matrix shows a systematic FbEC|SEC code satisfying the bounds on the code length given by Theorem 10.2:*

$$\boldsymbol{H} = \left[\begin{array}{c|cccccc|cccccc} \boldsymbol{I}_b & \boldsymbol{M}_0 & \boldsymbol{M}_3 & \cdots & \boldsymbol{M}_i & \cdots & \boldsymbol{M}_{N_e} & \boldsymbol{M}_1 & \boldsymbol{M}_2 & \cdots & \boldsymbol{M}_j & \cdots & \boldsymbol{M}_{N_o} \\ \boldsymbol{P}_{(r-b)\times b} & \boldsymbol{Q}_e & \boldsymbol{Q}_e & \cdots & \boldsymbol{Q}_e & \cdots & \boldsymbol{Q}_e & \boldsymbol{Q}_o & \boldsymbol{Q}_o & \cdots & \boldsymbol{Q}_o & \cdots & \boldsymbol{Q}_o \end{array}\right],$$

where

$\boldsymbol{I}_b$: $b \times b$ *identity matrix*,
$\boldsymbol{P}_{(r-b)\times b}$: $(r-b) \times b$ *matrix with all 1's*,
i: *integer whose binary representation has even weight*,
j: *integer whose binary representation has odd weight*,
N_e: *maximum integer i having value no greater than* $2^b - 1$,
N_o: *maximum integer j having value no greater than* $2^b - 1$,
$\boldsymbol{M}_i(\boldsymbol{M}_j)$: $b \times (2^{r-b} - 1)$ *matrix whose binary column vector of b-bit length indicates integer i(j)*,
$\boldsymbol{Q}_e$: $(r-b) \times (2^{r-b} - 1)$ *matrix whose distinct column vectors indicate integers from 1 to* $2^{r-b} - 1$,
$\boldsymbol{Q}_o$: $(r-b) \times (2^{r-b} - 1)$ *matrix whose distinct column vectors indicate integers from zero to* $2^{r-b} - 2$.

Proof Since nonzero column vectors in $\mathbf{H}$ are all distinct, the code satisfies conditions 1 and 2 of Theorem 10.1 for the error set $\boldsymbol{E}_1$, and therefore the code has an SEC function. Since matrix $\mathbf{I}_b$ is nonsingular, the code satisfies conditions 1 and 3 for the error set $\boldsymbol{E}_0$.

Let the upper b bits of syndrome S be S_F and the lower $r - b$ bits of S be S_p. For the byte errors in X_0, S_p is an all-0 or an all-1 vector. If S_p is an all-0 vector, then X_0 includes an even number of bit errors and S_F is of even weight. If S_p is an all-1 vector, then X_0 has an odd number of bit errors and S_F is of odd weight. Syndromes caused by single-bit errors occurring outside X_0 are different from those caused by byte errors in X_0. This is because in the case of single-bit errors, S_p is not all-0 for S_F with even weight while S_p is not all-1 for S_F with odd weight. So condition 4 of Theorem 10.1 is satisfied. Hence the $\mathbf{H}$ matrix shown in the theorem is an FbEC|SEC code.

The maximum number of columns in $\mathbf{H}_1$ shown in the previous $\mathbf{H}$ is $2^b \times (2^{r-b} - 1)$, and hence the maximum code length in bits equals

$$\begin{aligned} N_{max} &= b + 2^b \times (2^{r-b} - 1) \\ &= 2^r - 2^b + b. \end{aligned}$$

This code length is equal to the maximum code length of the FbEC|SEC code shown in Theorem 10.2. Q.E.D.

We see from the proof that the code indicated in Theorem 10.3 is optimal.

Example 10.1 Systematic (27, 22) F3EC|SEC Code

$$\mathbf{H} = \left[\begin{array}{c|cccc:cccc} 100 & 000 & 000 & 111 & 111 & 000 & 000 & 111 & 111 \\ 010 & 000 & 111 & 000 & 111 & 000 & 111 & 000 & 111 \\ \underline{001} & \underline{000} & \underline{111} & \underline{111} & \underline{000} & \underline{111} & \underline{000} & \underline{000} & \underline{111} \\ 111 & 011 & 011 & 011 & 011 & 001 & 001 & 001 & 001 \\ 111 & 101 & 101 & 101 & 101 & 010 & 010 & 010 & 010 \\ & \uparrow\uparrow & & & & \uparrow & \uparrow & \uparrow & \end{array}\right].$$

The place of the check bits is indicated by the upward-pointing arrow (↑).

Decoding Procedure Single-bit error correction can be easily performed such that if the nonzero syndrome is equal to one of the column vectors in $\mathbf{H}$, the corresponding bit is inverted and then corrected as a single-bit error. If the nonzero syndrome is not equal to any column vector in $\mathbf{H}$, then byte errors in X_0 can be assumed to exist. Let the upper b bits of the syndrome S be S_F and the lower $r - b$ bits of S be S_p. Further let calculation of $S_F \cdot (\mathbf{I}_b)^{-1}$ be E_p, meaning $E_p = S_F$. If $E_p \cdot \mathbf{P}^T_{(r-b)\times b} = S_p$, then E_p is a byte-error pattern, which is added to the original fixed-byte information of X_0. This provides a correction of the erroneous fixed-byte X_0. If $E_p \cdot \mathbf{P}^T_{(r-b)\times b} \neq S_p$, then we can assume that there exist multiple-bit errors in the word other than single-bit errors in X_1 and fixed-byte errors in X_0 (i.e., uncorrectable errors) that are finally detected by the code.

The FbEC|SEC code does not require large decoding hardware augmentation compared to the existing SEC-DED code. For example, the decoder of the $(72,\ 64)$ F7EC|SEC code requires only 11.6% hardware augmentation compared to that of the (72, 64) SEC-DED code.

Evaluation The FbEC|SEC codes are evaluated by their check-bit lengths and error detection capabilities. Figure 10.5 shows the relation between the information-bit lengths and the check-bit lengths of the FbEC|SEC codes for the fixed-byte X_0 with lengths $b = 4, 6, 7, 8$, and 10 bits. For comparison, the lengths of SEC codes are indicated in the figure as well. Note that the F4EC|SEC code has almost the same check-bit length as that of the SEC code.

Figure 10.6 provides an example of (72, 64) F7EC|SEC code in a shortened version of the original (135, 127) F7EC|SEC code. In the obtained $\mathbf{H}$ matrix of the (135, 127)

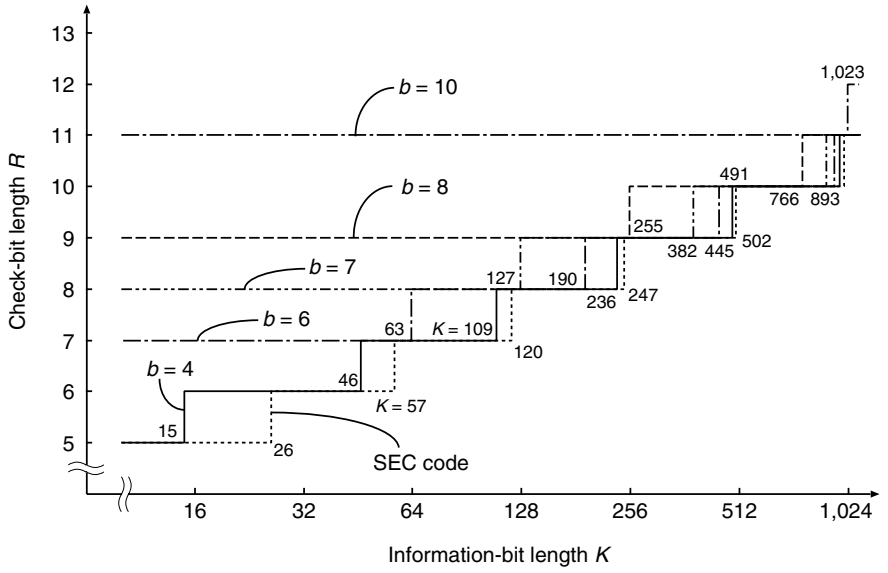


Figure 10.5 *Check-bit lengths compared with information-bit lengths of the FbEC|SEC codes.* Source: *[FUJI98]. © 1998 IEEE.*

$$\mathbf{H}' = \left[\begin{array}{l|l|l}
1000000 & 00000000000000011111100000000000000000000011111111111111111 & 10000000 \\
0100000 & 00000000011111000001000000000011111111110000000000111111 & 01000000 \\
0010000 & 00000111100001000010000011111100000011110000001111000011 & 00100000 \\
0001000 & 00011100010001000100011100011100011100010001110001000101 & 00010000 \\
0000100 & 01100100100010000100010110110010110010010011001001000 1001 & 00001000 \\
0000010 & 10101001000100001000011011010101010100100101010010001 0000 & 00000100 \\
0000001 & 11010010001000010000011101101001101001000110100100010 0000 & 00000010 \\
1111111 & 11111111111111111111100000000000000000000000000000000000 & 00000001
\end{array}\right]$$

Figure 10.6 *Example of the (72, 64) F7EC|SEC code.*

F7EC|SEC code, 63 column vectors having a larger number of 1's are deleted, and then the code shown in Figure 10.6 is obtained.

Table 10.2 lists the error detection capabilities of some shortened $(k + r,\ k)$ FbEC|SEC codes for two types of errors: random double-bit errors in the entire word and byte errors in X_0 plus single-bit errors in X_1 that are beyond the original error correction capabilities of

TABLE 10.2 Error Detection Capabilities of (k+r, k) FbEC | SEC Codes

b	k	r	Double-bit errors (%)	Byte errors in X_0 plus single-bit errors in X_1 (%)
6	32	7	17.41	49.21
6	64	8	45.58	65.75
7	64	8	6.57	49.61
8	128	9	6.40	49.80

TABLE 10.3 Miscorrection Rates of $(k + r,\ k)$ F*b*EC|SEC Codes

b	*k*	*r*	Double-bit errors occurring outside the fixed-byte X_0	
			Miscorrect single bit in X_1 (%)	Miscorrect fixed-byte X_0 (%)
6	32	7	11.34	71.26
6	64	8	26.56	27.86
7	64	8	12.05	81.38
8	128	9	4.98	88.62

the codes. The shortening method of the codes in this table is same as that of the (72, 64) F7EC|SEC codes shown in Figure 10.6.

10.2.2 Optimal Fixed-Byte Error Correcting | Single-Bit Error Correcting and Double-Bit Error Detecting (Optimal F*b*EC|SEC-DED) Codes

The F*b*EC|SEC code does not strongly protect the fixed-byte X_0. Table 10.3 lists the miscorrection tendencies of the $(k + r,\ k)$ F*b*EC|SEC codes for random double-bit errors occurring outside the fixed-byte. This table gives a more precise picture of the miscorrection of double-bit errors in Table 10.2. The percentage of errors miscorrected in the fixed-byte X_0 is high. As is evident, linear codes are needed that give the fixed-byte X_0 strong error protection against a large number of errors that occurr outside X_0 such as random double-bit errors, and burst errors. The 2-level UEC code shown here offers this required strong error protection of the fixed-byte X_0. The *FbEC|SEC-DED code* corrects b-bit byte errors in X_0 and, in addition, corrects single-bit errors in X_1 and detects double-bit errors in X_1.

Code Conditions and Bounds

Theorem 10.4 *A binary linear code, described by the parity-check matrix* $\boldsymbol{H}$*, corrects any error in the fixed-byte* X_0 *and, in addition, corrects single-bit errors and detects double-bit errors in* X_1*, if and only if:*

1. $E \cdot \boldsymbol{H}^T \neq 0 \quad \text{for all } E \in \{\boldsymbol{E}_0 \cup \boldsymbol{E}_1\},$
2. $E_i \cdot \boldsymbol{H}^T \neq E_j \cdot \boldsymbol{H}^T \quad \text{for all } E_i, E_j \in \boldsymbol{E}_1,\ E_i \neq E_j,$
3. $E_p \cdot \boldsymbol{H}^T \neq E_q \cdot \boldsymbol{H}^T \quad \text{for all } E_p, E_q \in \boldsymbol{E}_0,\ E_p \neq E_q,$
4. $E_p \cdot \boldsymbol{H}^T \neq E_i \cdot \boldsymbol{H}^T \quad \text{for all } E_p \in \boldsymbol{E}_0 \text{ and for all } E_i \in \boldsymbol{E}_1,$
5. $(E_i + E_j) \cdot \boldsymbol{H}^T \neq E_p \cdot \boldsymbol{H}^T \quad \text{for all } E_i, E_j \in \boldsymbol{E}_1 \text{ and for all } E_p \in \boldsymbol{E}_0, E_i \neq E_j,$
6. $(E_i + E_j) \cdot \boldsymbol{H}^T \neq E_k \cdot \boldsymbol{H}^T \quad \text{for all } E_i, E_j, E_k \in \boldsymbol{E}_1,\ E_i \neq E_j \neq E_k \neq E_i,$

where $\boldsymbol{E}_0$ *is the error set caused by all possible errors in* X_0 *and* $\boldsymbol{E}_1$ *the error set caused by single-bit errors in* X_1*, and* $\boldsymbol{H}^T$ *is the transpose of* $\boldsymbol{H}$*.*

Proof It is apparent that conditions 1, 2, and 6, and conditions 1 and 3 are necessary and sufficient conditions for single-bit error correction and double-bit error detection in X_1, and also the conditions for byte error correction in X_0, respectively. In addition to these, single-bit error correction in X_1 and byte error correction in X_0 can be performed

independently by condition 4. Double-bit error detection in X_1 and byte error correction in X_0 can be performed independently by adding condition 5. So conditions 1 to 6 are the necessary and sufficient conditions for the FbEC|SEC-DED code. Q.E.D.

Theorem 10.5 *The maximum code length (in bits) of an* $(N,\ N-r)$ *FbEC|SEC-DED code is*

$$N_{max} = 2^{r-b} + b - 1. \tag{10.3}$$

Proof We first consider the minimum number of distinct syndromes necessary for satisfying the conditions of the FbEC|SEC-DED code:

Case 1. Number of syndromes by single-bit errors occurring outside X_0 : $N - b$
Case 2. Number of syndromes for byte error correction in X_0 : $2^b - 1$

Any byte error E_p in X_0 included in $\boldsymbol{E}_0$ can be expressed as $E_p = E'_p + E''_p$, where E'_p and E''_p are distinct byte errors included in $\boldsymbol{E}_0$. As a result, condition 5 of Theorem 10.4 is equivalent to the following condition:

5′. $(E'_p + E_i) \cdot \mathbf{H}^T \neq (E''_p + E_j) \cdot \mathbf{H}^T$ for all $E'_p, E''_p \in \boldsymbol{E}_0,\ E_i \neq E_j,\ E'_p \neq E''_p$ and for all $E_i, E_j \in \boldsymbol{E}_1$.

Condition 5′ says that the syndromes caused by distinct byte plus single-bit errors are not equal to each other.

Case 3. Number of syndromes caused by byte plus single-bit errors: $(2^b - 1)(N - b)$
Case 4. Number of syndromes caused by double-bit errors occurring outside X_0 : $N - b - 1$

The syndrome space of case 4 can all be included in that of case 3. From conditions 4 and 5 of Theorem 10.4, the syndrome spaces of cases 1 to 3 should be distinct, and therefore the following relation holds:

$$2^r - 1 \geq (N - b) + (2^b - 1) + (2^b - 1)(N - b).$$

So we have $N_{max} = 2^{r-b} + b - 1$. Q.E.D.

Equation (10.3) can be used to express the maximum information-bit length, K_{max}, as

$$\begin{aligned} K_{max} = N_{max} - r &= 2^{r-b} + b - 1 - r \\ &= 2^x - x - 1, \end{aligned}$$

where $x = r - b$. We see that K_{max} is treated as a function of $x = r - b$. Table 10.4 presents the relation between $r = b + x$ and K_{max}.

TABLE 10.4 Relation between $r = b + x$ and K_{max} of the FbEC|SEC-DED Codes

$r = b + x$	$b+2$	$b+3$	$b+4$	$b+5$	$b+6$	$b+7$	$b+8$	$b+9$	$b+10$
$K_{max} = 2^x - x - 1$	1	4	11	26	57	120	247	502	1,013

Source: [FUJI98]. © 1998 IEEE.

Theorem 10.6 *A linear FbEC|SEC-DED code can detect byte plus single-bit errors; that is, it can detect byte errors in X_0 and single-bit errors in X_1 simultaneously.*

Theorem 10.6 can be proved by conditions 4 and 5 of Theorem 10.4.

The next theorem holds for the byte errors in X_0 and the double-bit errors in X_1 occurring simultaneously, which are denoted as *byte plus double-bit errors*.

Theorem 10.7 *In a linear FbEC|SEC-DED code the byte plus double-bit errors do not miscorrect any bits in the fixed-byte X_0.*

Proof Assume that the byte plus double-bit errors are miscorrected as byte errors. Then the following relation holds:

$$(E_p + E_i + E_j) \cdot \mathbf{H}^T = E_q \cdot \mathbf{H}^T,$$

where $E_p, E_q \in \boldsymbol{E}_0$, and $E_i, E_j \in \boldsymbol{E}_1$. This can be transformed into the following relation:

$$(E_p + E_i) \cdot \mathbf{H}^T = (E_q + E_j) \cdot \mathbf{H}^T,$$

which contradicts condition $5'$ shown in the proof of Theorem 10.5. Hence the byte plus double-bit errors are not miscorrected as byte errors in the FbEC|SEC-DED code. Q.E.D.

Design for Optimal FbEC|SEC-DED Codes Without loss of generality, the fixed-byte X_0 is assumed to be located at the beginning of the word, as was noted previously in Section 10.1. Here the $\mathbf{H}$ matrix of the code is divided into two submatrices shown in Eq. (10.1), with $\mathbf{H}_0$ having b columns and $\mathbf{H}_1$ the remaining columns in $\mathbf{H}$, that is, $N - b$ columns.

Theorem 10.8 *The following $\mathbf{H}$ matrix shows an FbEC|SEC-DED code satisfying the bounds on code length in bits shown in Theorem 10.5:*

$$\boldsymbol{H} = \left[\boldsymbol{H}_0 \;\middle|\; \boldsymbol{H}_1\right] = \left[\begin{array}{c|c|c} \boldsymbol{I}_b & \boldsymbol{Q} & \boldsymbol{O} \\ \hline \boldsymbol{P} & \boldsymbol{M}_e & \boldsymbol{M}_o \end{array} \middle| \begin{array}{c} \\ + \\ \end{array} \boldsymbol{I}_r\right],$$

where

$$\boldsymbol{H}_0 = \left[\begin{array}{c} \boldsymbol{I}_b \\ \hline \boldsymbol{P} \end{array}\right], \quad \boldsymbol{H}_1 = \left[\begin{array}{c|c} \boldsymbol{Q} & \boldsymbol{O} \\ \hline \boldsymbol{M}_e & \boldsymbol{M}_o \end{array} + \boldsymbol{I}_r\right],$$

$\boldsymbol{I}_b(\boldsymbol{I}_r)$: *$b \times b$ $(r \times r)$ identity matrix*,

$\boldsymbol{O}$: *zero matrix*,

$\boldsymbol{P}$: *$(r-b) \times b$ matrix having distinct b nonzero even-weight columns*,

$\boldsymbol{Q}$: *matrix whose columns are repetitions of the first column in $\boldsymbol{I}_b$*,

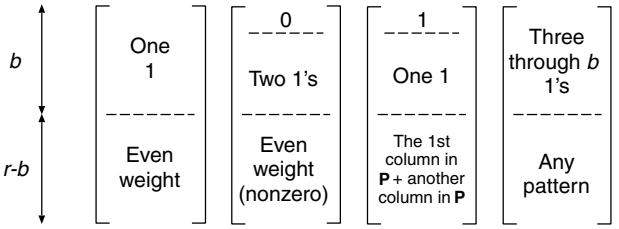
$\boldsymbol{M}_e$: *matrix having $(r-b)$-bit nonzero even-weight columns except for $b-1$ columns obtained by adding the first column in $\boldsymbol{P}$ to each remaining column in $\boldsymbol{P}$ as well as for the first column in $\boldsymbol{P}$*,

$\boldsymbol{M}_o$: *matrix having $(r-b)$-bit odd-weight columns except for weight-one columns.*

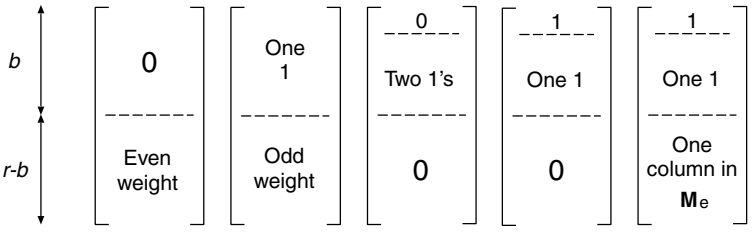
Proof Since nonzero column vectors in **H** are all distinct and odd weight, the code satisfies conditions 1, 2, and 6 of Theorem 10.4 for the error set $\boldsymbol{E}_1$; therefore the code has the SEC-DED function. Since the matrix $\mathbf{I}_b$ is nonsingular, the code satisfies conditions 1 and 3 in the fixed-byte X_0 for the error set $\boldsymbol{E}_0$. It is easy to see that the syndrome caused by any byte error in X_0 is not equal to the syndrome caused by any single-bit error in X_1. So condition 4 holds. Figure 10.7 shows the syndrome patterns caused by byte errors in X_0 and by double-bit errors in X_1. Note that the syndromes caused by byte errors are distinct from those caused by double-bit errors, so condition 5 holds. Based on the above, the **H** matrix given in this theorem satisfies all the conditions of Theorem 10.4 and therefore is the parity-check matrix of the FbEC|SEC-DED code.

Since the matrices $\mathbf{M}_e$ and $\mathbf{M}_o$ have the maximum number of columns, $2^{r-b-1}-b-1$ and $2^{r-b-1}-(r-b)$, respectively, the maximum length (in bits) of this code can be expressed as

$$\begin{aligned} N_{max} &= b + (2^{r-b-1} - b - 1) + (2^{r-b-1} - (r-b)) + r \\ &= 2^{r-b} + b - 1. \end{aligned}$$



(a) Syndrome patterns of byte errors in X_0



(b) Syndrome patterns of double-bit errors occurring outside X_0

Figure 10.7 *Syndrome patterns caused by byte errors in X_0 and also by double-bit errors occurring outside X_0.* Source: *[FUJI98]. © 1998 IEEE.*

This is equal to the maximum code length (in bits) of the F*b*EC|SEC-DED code given in Theorem 10.5. Q.E.D.

Consequently the code indicated in Theorem 10.8 is optimal.

Example 10.2 (19, 11) F4EC|SEC-DED Code

$$\mathbf{H} = \left[\begin{array}{c|c|c|c} 1000 & 111 & 0000 & 10000000 \\ 0100 & 000 & 0000 & 01000000 \\ 0010 & 000 & 0000 & 00100000 \\ 0001 & 000 & 0000 & 00010000 \\ \hline 0110 & 101 & 1110 & 00001000 \\ 1101 & 011 & 1101 & 00000100 \\ 0011 & 110 & 1011 & 00000010 \\ 1000 & 000 & 0111 & 00000001 \end{array}\right].$$

Evaluation The F*b*EC|SEC-DED codes are evaluated from the perspectives of error detection capabilities and decoder hardware amount.

The F*b*EC|SEC-DED codes sometimes miscorrect b-bit burst errors and random triple-bit errors, some of which are beyond the original error correction / detection capabilities of the code, as single-bit errors in X_1 or as fixed-byte errors in X_0. Table 10.5 shows the miscorrection tendencies of the example codes with information-bit lengths $k = 32$, 64, and 128, which are shortened $(k + r, k)$ F*b*EC|SEC-DED codes with $k = 32$, 64, and 128 bits. Here the reader should recall Theorems 10.6 and 10.7 that the byte plus single-bit or double-bit errors do not miscorrect any bits in the fixed-byte X_0 in the F*b*EC|SEC-DED code. The table indicates that the miscorrection of fixed-byte X_0 is very small, and therefore the fixed-byte X_0 has strong error protection against multiple-bit errors.

The F*b*EC|SEC-DED code does not require large decoding hardware augmentation compared to the existing SEC-DED code. For example, the decoder of the (75, 64) F4EC|SEC-DED code requires 15.49% larger hardware than the (72, 64)SEC-DED code.

TABLE 10.5 Miscorrection Rates of (*k*+*r*, *k*) F*b*EC|SEC-DED Codes

			b-Bit burst errors		Triple-bit errors	
b	*k*	*r*	Miscorrect single bit in X_1 (%)	Miscorrect fixed-byte X_0 (%)	Miscorrect single bit in X_1 (%)	Miscorrect fixed-byte X_0 (%)
4	32	10	17.55	0.63	35.11	1.17
8	32	14	17.13	1.23	19.06	0.91
12	32	18	10.38	1.33	12.50	0.75
4	64	11	16.81	0.51	43.02	0.63
8	64	15	21.97	0.89	29.93	0.56
12	64	19	18.60	0.81	20.57	0.50
4	128	12	24.21	0.09	52.39	0.29
8	128	16	31.27	0.41	43.58	0.27
12	128	20	28.94	0.41	36.20	0.27

Source: [FUJI98]. © 1998 IEEE.

10.3 BURST ERROR CONTROL UEC / UEP CODES

10.3.1 Burst Error Control UEC Codes — $B_lEC|SEC$ Codes —

Another class of UEC codes is the $B_lEC|SEC$ codes. The $B_lEC|SEC$ codes correct l-bit burst errors in the X_0 area having n_0-bit length, where $l < n_0$, as well as correct single-bit errors in the X_1 area having n_1-bit length [NAMB03]. This UEC code type can be extended such that burst errors with larger lengths of $L = p \times l$ bits can be corrected in X_0 as well as burst errors with lengths of p bits by applying the interleaving method of degree p to the $B_lEC|SEC$ code.

Code Conditions and Bounds The relation between the codeword and the corresponding parity-check matrix $\mathbf{H}$ of the $B_lEC|SEC$ codes is shown in Figure 10.8. The matrix $\mathbf{H}$ is constituted by an $r \times n_0$ submatrix $\mathbf{H}_{\mathrm{BEC}}$ and an $r \times n_1$ submatrix $\mathbf{H}_{\mathrm{SEC}}$.

Theorem 10.9 *A binary linear code, described by the parity-check matrix $\boldsymbol{H}$, corrects l-bit burst errors in X_0 and, in addition, corrects single-bit errors in X_1, if and only if:*

1. $E \cdot \boldsymbol{H}_{BEC}^T \neq 0$ *for all* $E \in \boldsymbol{E}_B$
2. $E \cdot \boldsymbol{H}_{BEC}^T \neq E' \cdot \boldsymbol{H}_{BEC}^T$ *for all* $E, E' \in \boldsymbol{E}_B, E \neq E'$,
3. $E \cdot \boldsymbol{H}_{SEC}^T \neq 0$ *for all* $E \in \boldsymbol{E}_b$,
4. $E \cdot \boldsymbol{H}_{SEC}^T \neq E' \cdot \boldsymbol{H}_{SEC}^T$ *for all* $E, E' \in \boldsymbol{E}_b, E \neq E'$
5. $E \cdot \boldsymbol{H}_{BEC}^T \neq E' \cdot \boldsymbol{H}_{SEC}^T$ *for all* $E \in \boldsymbol{E}_B$, *and for all* $E' \in \boldsymbol{E}_b$,

where $\boldsymbol{E}_B$ and $\boldsymbol{E}_b$ are error sets of l-bit burst errors in X_0 and single-bit errors in X_1, respectively.

The proof of this theorem is left to the reader.

Theorem 10.10 *An $(n_0 + n_1, n_0 + n_1 - r)$ $B_lEC|SEC$ code satisfies the following relations:*

$$r \geq min(2l,\, n_0), \tag{10.4}$$

$$r \geq \lceil log_2\{(n_0 - l + 2) \cdot 2^{l-1} + n_1\}\rceil, \tag{10.5}$$

where min(x, y) means x if $x \leq y$ and y if $x > y$, and $\lceil z \rceil$ is the smallest integer larger than or equal to z.

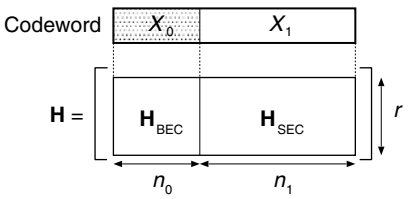


Figure 10.8 *Parity-check matrix of the $B_lEC|SEC$ code corresponding to the 2-level UEC codeword.* Source: *[NAMB03]. © 2003 IEICE Japan.*

Proof In general, l-bit burst error correcting codes satisfy the Reiger bound [REIG60] written as $r \geq 2l$. However, for $n_0 < 2l$, by condition 2 of Theorem 10.9, every column vector in $\mathbf{H}_{\mathrm{BEC}}$ should be linearly independent, so $r \geq n_0$. Therefore a $\mathrm{B}_l\mathrm{EC}|\mathrm{SEC}$ code should satisfy the relation (10.4).

In the area X_0, there exist $(n_0 - l + 2) \cdot 2^{l-1} - 1$ distinct l-bit burst errors, and also these exist n_1 single-bit errors in X_1. The syndromes of these errors should be distinct, and thus the following relation holds:

$$2^r - 1 \geq (n_0 - l + 2) \cdot 2^{l-1} + n_1 - 1.$$

This satisfies the relation (10.5). Q.E.D.

Code Design I

Theorem 10.11 *The following parity-check matrix* $\boldsymbol{H}$ *shows a* $B_lEC|SEC$ *code:*

$$\boldsymbol{H} = [\boldsymbol{H}_{\boldsymbol{BEC}} \mid \boldsymbol{H}_{\boldsymbol{SEC}}],$$

$$\boldsymbol{H}_{BEC} = \begin{bmatrix} \boldsymbol{I}_{n_0} \\ \boldsymbol{O} \end{bmatrix} \begin{matrix} \updownarrow n_0 \\ \updownarrow r - n_0, \end{matrix}$$

$$\boldsymbol{H}_{SEC} = \begin{bmatrix} \boldsymbol{M}' & \boldsymbol{M} & \boldsymbol{M} & \cdots & \boldsymbol{M} \\ \boldsymbol{O} & \boldsymbol{Q}_1 & \boldsymbol{Q}_2 & \cdots & \boldsymbol{Q}_{2^{r-n_0}-1} \end{bmatrix} \begin{matrix} \updownarrow n_o \\ \updownarrow r - n_0, \end{matrix}$$

where

$\boldsymbol{I}_{n_0}$: $n_0 \times n_0$ *identity matrix*,
$\boldsymbol{O}$: $(r - n_0) \times n_0$ *zero matrix*,
$\boldsymbol{Q}_i$: $(r - n_0) \times 2^{n_0}$ *matrix whose columns are all equal, and its column patterns are binary representation of integer i, and* $\boldsymbol{Q}_i \neq \boldsymbol{Q}_j$ *where* $i \neq j$, *and* $1 \leq \mathrm{i} \leq 2^{\mathrm{r}-\mathrm{n}_0} - 1$,
$\boldsymbol{M}$: $n_0 \times 2^{n_0}$ *matrix whose columns are distinct with each other*,
$\boldsymbol{M}'$: *matrix constituted by the columns of* $\boldsymbol{M}$ *from which columns with zero pattern and l-bit burst pattern are excluded.*

This theorem can be easily proved, and therefore the proof is omitted.

Example 10.3 (56, 50) $\mathrm{B}_3\mathrm{EC}|\mathrm{SEC}$ code with $n_0 = 4$

In this code we have to pay attention to the construction of the matrix $\mathbf{M}'$. That is, we have to delete zero pattern columns and 3-bit burst pattern columns, surrounded by the four-sided figures in $\mathbf{M}$ in the code of Figure 10.9.

Code Design II In this code design we use the parity-check matrix of the Fire code for the submatrix $\mathbf{H}_{\mathrm{BEC}}$. We assume that the code can correct *cyclic burst error patterns*. These cyclic burst error patterns of l-bit length exist as w-bit burst errors at the beginning

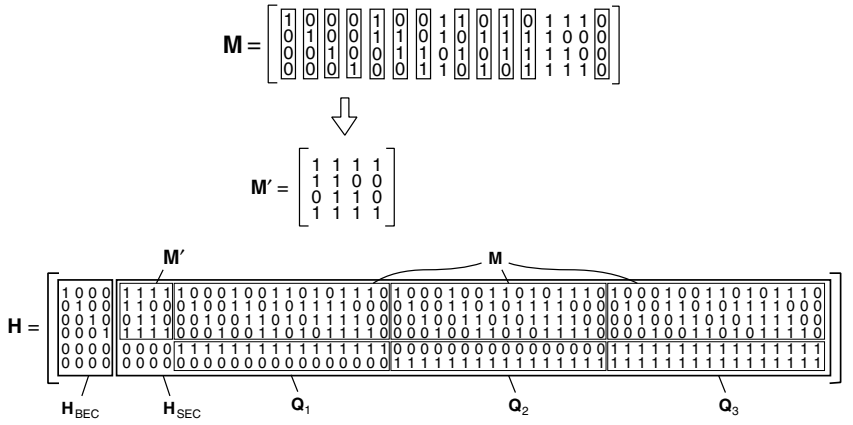


Figure 10.9 *Parity-check matrix of the (56, 50) $B_3EC|SEC$ code with $n_0 = 4$.* Source: *[NAM03]. © 2003 IEICE Japan.*

of the pattern and $(l - w)$-bit burst errors at the end of the pattern, and in between there are no errors so that $0 < w < l$.

The following shows the parity-check matrix of the l-bit cyclic burst error correcting Fire code generated by $\mathbf{g}(x) = (x^c - 1) \cdot \mathbf{p}(x)$, where $\mathbf{p}(x)$ is a primitive polynomial with degree m $(\geq l)$, c is an integer larger than or equal to $2l - 1$, and $n_e = \mathrm{LCM}(2^m - 1,\ c)$ where $\mathrm{LCM}(y,\ z)$ is the least common multiple of y and z:

$$\widetilde{\mathbf{H}}_i = \left[\begin{array}{c|c|c|c} \mathbf{I}_c & \mathbf{I}_c & \cdots & \mathbf{I}_c \\ \hline \alpha^i \ \cdots \ \alpha^{i+c-1} & \alpha^{i+c} \ \cdots \ \alpha^{i+2c-1} & \cdots & \alpha^{i+n_e-c} \ \cdots \ \alpha^{i+n_e-1} \end{array}\right] \begin{array}{l} c \\ \\ m \end{array} \quad (10.6)$$

In this case, α is a root of $\mathbf{p}(x)$, and i is an arbitrary integer.

From the preparation above, we have another type of $B_lEC|SEC$ code shown in the following theorem.

Theorem 10.12 *The following parity-check matrix $\boldsymbol{H}$ shows a $B_lEC|SEC$ code:*

$$\boldsymbol{H} = [\boldsymbol{H}_{BEC} \,|\, \boldsymbol{H}_{SEC}],$$

where

$\boldsymbol{H}_{BEC}$: $(c + m) \times n_0$ *matrix deleting the last* $(n_e - n_0)$ *columns in* $\widetilde{\boldsymbol{H}}_0$ *shown in* (10.6) *with* $i = 0$,

$\boldsymbol{H}_{SEC} = [\boldsymbol{H}_A \ \boldsymbol{H}_B \ \boldsymbol{H}_\Gamma]$,

$$\boldsymbol{H}_A = \left[\begin{array}{c|c|c|c|c|c} \boldsymbol{W}_c & \boldsymbol{O} & \boldsymbol{Y}_0 & \boldsymbol{Y}_1 & \cdots & \boldsymbol{Y}_{2^c - c \cdot 2^{l-1} - 2} \\ \hline \boldsymbol{O} & \boldsymbol{W}_m & \boldsymbol{W}_m & \boldsymbol{W}_m & \cdots & \boldsymbol{W}_m \\ \leftarrow 2^c - 1 \rightarrow & \leftarrow 2^m - 1 \rightarrow & \leftarrow 2^m - 1 \rightarrow & \leftarrow 2^m - 1 \rightarrow & & \longleftarrow 2^m - 1 \longrightarrow \end{array}\right] \begin{array}{l} c \\ m \end{array},$$

$\boldsymbol{W}_x : x \times (2^x - 1)$ *matrix having distinct nonzero column vectors,*

$\boldsymbol{Y}_j : [y_j\ y_j\ \cdots\ y_j] \updownarrow c,$

$\leftarrow 2^m - 1 \rightarrow$

$y_j\ (0 \le j < 2^c - c \cdot 2^{l-1} - 1)$: *nonzero column vectors with lenght c, not equal to the vectors with l-bit cyclic burst patterns,*

$$\boldsymbol{H}_B = [\boldsymbol{H}_1^*\ \boldsymbol{H}_2^* \cdots \boldsymbol{H}_{GCD(c,2^m-1)-1}^*],$$

$$\boldsymbol{H}_i^* = \widetilde{\boldsymbol{H}}_i \cdot \boldsymbol{Z} \cdot \boldsymbol{P},$$

$$\boldsymbol{Z} = \left[\begin{array}{c|c} \boldsymbol{I}_{n_e} & \begin{matrix}\boldsymbol{I}_{l-1}\\ \boldsymbol{O}\end{matrix} \end{array}\right] \begin{matrix} \\ \updownarrow n_e \end{matrix},$$

$$\overleftrightarrow{n_e + l - 1}$$

$$\boldsymbol{P} = \left[\begin{array}{cccc} \boldsymbol{Q} & & \boldsymbol{O} & \\ & \boldsymbol{Q} & & \\ & & \boldsymbol{Q} & \\ \boldsymbol{O} & & & \boldsymbol{Q} \end{array}\right] \begin{matrix} \\ \\ \\ \updownarrow n_e - l - 1, \end{matrix}$$

1 bit

$$\overleftrightarrow{n_e\, 2^{l-1}}$$

$$\boldsymbol{Q} = \begin{bmatrix} \cdots \\ - \end{bmatrix},$$

$\boldsymbol{M}_{l-1} : (l-1) \times 2^{l-1}$ *matrix with distinct column vectors,*

$$\boldsymbol{H}_\Gamma = \widetilde{\boldsymbol{H}}_0 \cdot \boldsymbol{Z} \cdot \boldsymbol{P}',$$

$\boldsymbol{P}'$: *matrix deleting the cokumn vectors whose bottom* $n_e - n_0 + l - 1$ *bits are all 0's in the matrix* $\boldsymbol{P}$.

The proof is complicated and the reader is recommended to refer to [NAM 03].

Example 10.4 [NAMB03]: (116, 109) $B_2EC|SEC$ Code for $n_0 = 12$

Figure 10.10 shows the **H** matrix of the code with submatrices $\widetilde{\mathbf{H}}_0,\ \widetilde{\mathbf{H}}_1,\ \widetilde{\mathbf{H}}_2,$ **P**, and **Z**.

Evaluation Figure 10.11 shows the burst error lengths and the check-bit lengths of codes I and II discussed above, for $n_0 = 64$ bits and $n_1 = 1,024$ bits. The figure also shows the bound given in Theorem 10.10. Note that code II is superior in the region $l < n_0/3$ to code I, whereas code I is superior to code II in the region $l \ge n_0/3$ and is equal to the bound in the region $l > n_0/2$. This relation is applicable to the codes with other code parameters.

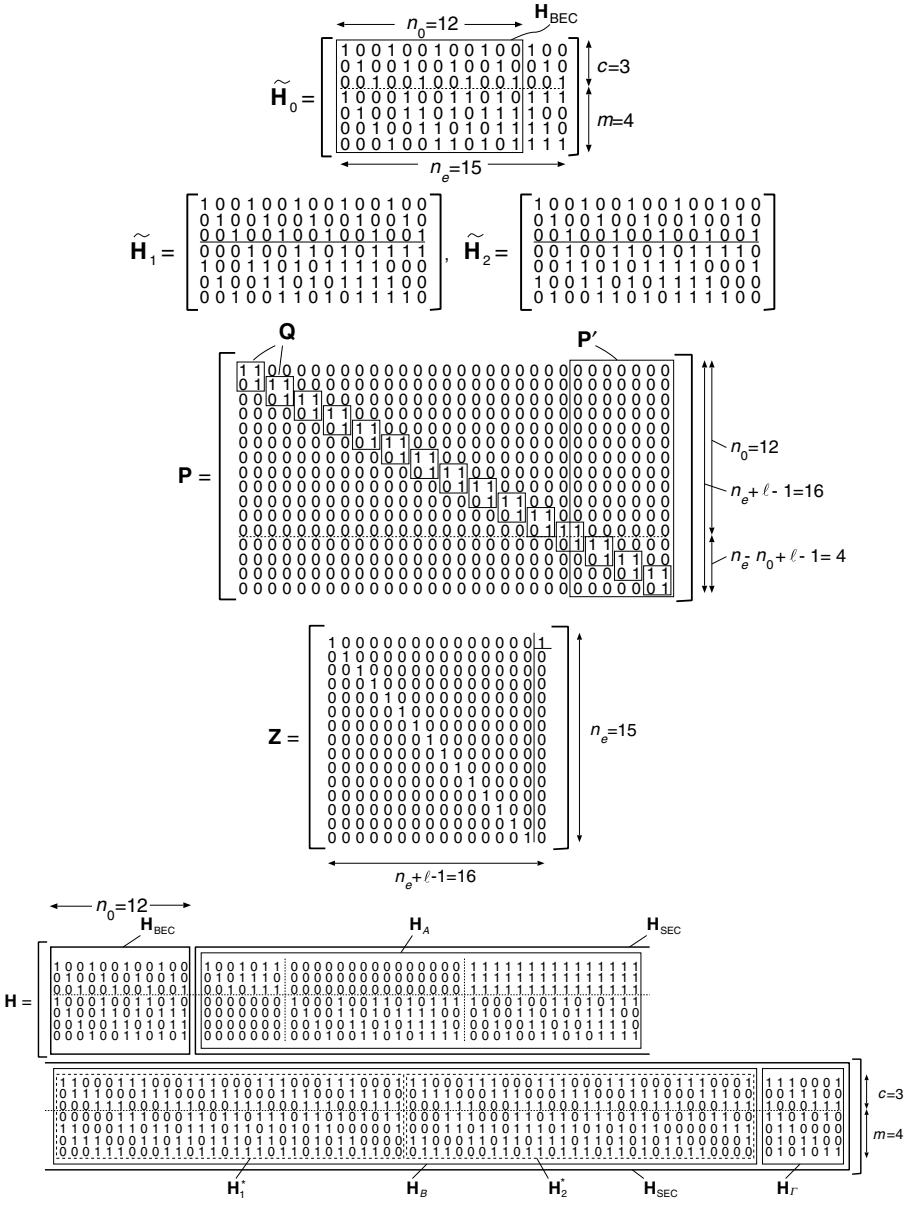


Figure 10.10 *Parity-check matrix of the (116, 109) $B_2EC|SEC$ code with $n_0 = 12$.* Source: *[NAMB03]. © 2003 IEICE Japan.*

10.3.2 Burst Error Control UEP Codes — $(B_lEC)_{n_o}$-$(SEC)_{n_1}$ UEP Codes —

There is a class of burst error control UEP codes with two different levels of error correction capabilities in a codeword: $l(\geq 2)$-bit burst error correction in an important area of the word and single-bit error correction in the remaining area of it [NAMB02]. A class of byte error control UEP codes has already been presented in [HAYA00, IWAS97]. The

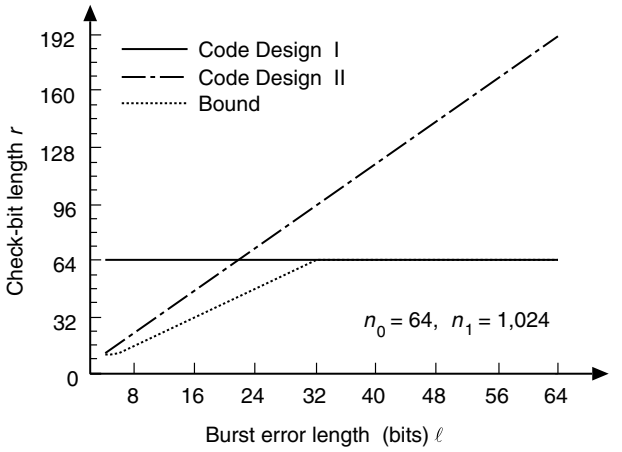


Figure 10.11 *Comparison of burst error lengths and check-bit lengths of the $B_lEC|SEC$ codes.* Source: *[NAMB03]. © 2003 IEICE Japan.*

byte error control UEP codes correct any single-bit errors in the whole word and also correct single-byte errors in an important area of the word.

Code Conditions and Bounds Let **C** be a code whose codewords consist of two parts: an important part X_0 and the other less important part X_1. Without loss of generality, assume that X_0 always precedes X_1 in a codeword and check bits are included in X_1. Let n_0 and n_1 be lengths of X_0 and X_1, respectively. If **C** is capable of correcting l-bit burst errors (B_lEC) in X_0 and of correcting single-bit errors (SEC) in X_1, then **C** is denoted as a $(B_lEC)_{n_0}$-$(SEC)_{n_1}$ UEP code, which is a different notation from the UEC codes. It is noted that the B_lEC function includes an SEC function, and therefore the codes correct single-bit errors not only in X_1 but also in the whole word.

Figure 10.12 shows the code functions of the $(B_lEC)_{n_0}$-$(SEC)_{n_1}$ UEP codes. The latter two cases—that is, the one where the l-bit burst errors are spanned over X_0 and X_1, and the other where the l-bit burst errors are in X_1—are important to note. In the first case, the errors included in X_0 should be corrected, and the remaining errors in X_1 should not miscorrect any bits in X_0. This is because the X_0 part should be strongly protected from

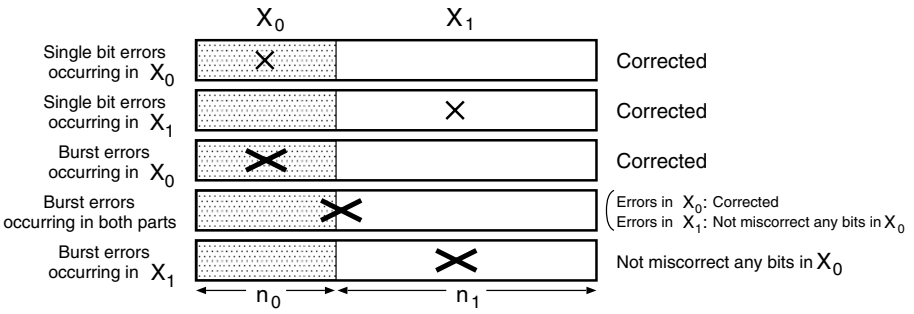


Figure 10.12 *Code functions of the $(B_lEC)_{n_0}$-$(SEC)_{n_1}$ UEP code.* Source: *[NAMB02]. © 2002 IEICE Japan.*

errors that occurr in any other part. Similarly, in the second case, the l-bit burst errors in X_1 should not miscorrect any bits in X_0.

To better see this consider four sets of errors, $\mathbf{E}_{B0}$, $\mathbf{E}_{B1}$, $\mathbf{E}_B$, and $\mathbf{E}_b$, where $\mathbf{E}_{B0}$ and $\mathbf{E}_{B1}$ are the error sets caused by l-bit burst errors in X_0 and in X_1, respectively, $\mathbf{E}_B$ the error set caused by l-bit burst errors spanned over two parts X_0 and X_1, and $\mathbf{E}_b$ the error set caused by single-bit errors. Let $X_0(E)$ and $X_1(E)$ represent the error E in the former part X_0 of the word and in the latter part X_1, respectively.

The following theorem provides the necessary and sufficient conditions of the $(\mathrm{B}_l\mathrm{EC})_{n_0}$-$(\mathrm{SEC})_{n_1}$ UEP code.

Theorem 10.13 *A binary linear code, described by the parity-check matrix $\boldsymbol{H}$, is a $(B_lEC)_{n_0}$-$(SEC)_{n_1}$ UEP code if and only if:*

1. $E \cdot \boldsymbol{H}^T \neq 0$ *for all* $E \in \{\boldsymbol{E}_{B0} \cup \boldsymbol{E}_B \cup \boldsymbol{E}_b\}$,
2. $E \cdot \boldsymbol{H}^T \neq E' \cdot \boldsymbol{H}^T$ *for all* $E, E' \in \boldsymbol{E}_{B0}$, $E \neq E'$,
3. $E \cdot \boldsymbol{H}^T \neq E' \cdot \boldsymbol{H}^T$ *for all* $E \in \boldsymbol{E}_{B0}$, *and for all* $E' \in \boldsymbol{E}_B$, $X_0(E) \neq X_0(E')$,
4. $E \cdot \boldsymbol{H}^T \neq E' \cdot \boldsymbol{H}^T$ *for all* $E, E' \in \boldsymbol{E}_B$, $X_0(E) \neq X_0(E')$,
5. $E \cdot \boldsymbol{H}^T \neq E' \cdot \boldsymbol{H}^T$ *for all* $E, E' \in \boldsymbol{E}_b$, $E \neq E'$,
6. $E \cdot \boldsymbol{H}^T \neq E' \cdot \boldsymbol{H}^T$ *for all* $E \in \boldsymbol{E}_{B0}$, *and for all* $E' \in \boldsymbol{E}_{B1}$,
7. $E \cdot \boldsymbol{H}^T \neq E' \cdot \boldsymbol{H}^T$ *for all* $E \in \boldsymbol{E}_B$, *and for all* $E' \in \boldsymbol{E}_{B1}$,

where $\boldsymbol{H}^T$ is the transpose of $\boldsymbol{H}$.

Proof It can be easily proved such that conditions 1 and 2 are the necessary and sufficient conditions for correcting l-bit burst errors in X_0, conditions 1, 3, and 4 for correcting errors in X_0 when l-bit burst errors corrupting both X_0 and X_1, conditions 1 and 5 for correcting single-bit errors, and conditions 6 and 7 for preventing l-bit burst errors in X_1 from being mistaken as correctable errors in X_0. Q.E.D.

Next the theoretical lower bounds on the check-bit length of a linear $(\mathrm{B}_l\mathrm{EC})_{n_0}$-$(\mathrm{SEC})_{n_1}$ UEP code are presented.

Theorem 10.14 *A linear $(n_0 + n_1,\ n_0 + n_1 - r)$ $(B_lEC)_{n_0}$-$(SEC)_{n_1}$ UEP code must satisfy the following inequalities:*

$$r \geq 2\,l, \tag{10.7}$$

$$r \geq log_2\{(n_0 - l + 4) \cdot 2^{l-1} + n_1 - l - 1\}. \tag{10.8}$$

Proof In general, l-bit burst error correcting codes satisfy the Reiger bound [REIG60] presented in relation (10.7).

Correctable errors of the $(\mathrm{B}_l\mathrm{EC})_{n_0}$-$(\mathrm{SEC})_{n_1}$ UEP codes have distinct nonzero syndromes, and hence we can count the number of all correctable errors in the following three cases:

Case 1. Number of l-bit burst errors in X_0 excluding single-bit errors: $(n_0 - l + 2) \cdot 2^{l-1} - n_0 - 1$

Case 2. Number of l-bit burst errors spanning over X_0 and X_1 whose Hamming weight is one in X_1: $\sum_{i=1}^{l-1}(2^i - 1) = 2^l - l - 1$

Case 3. Number of single-bit errors: $n_0 + n_1$

Hence we have the following inequality in the syndrome space:

$$\begin{aligned} 2^r - 1 &\geq \left\{ (n_0 - l + 2) \cdot 2^{l-1} - n_0 - 1 \right\} \\ &\quad + \left(2^l - l - 1 \right) + (n_0 + n_1) \\ &= (n_0 - l + 4) \cdot 2^{l-1} + n_1 - l - 2, \end{aligned}$$

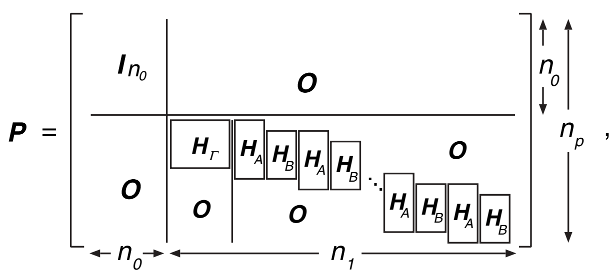
which expresses the relation (10.8). Q.E.D.

Design for the $(B_lEC)_{n_0}$-$(SEC)_{n_1}$ UEP Code We will use here a matrix **P** that converts the error pattern E with length $n_0 + n_1$, including l-bit burst errors in X_0 and single-bit errors in X_1 in the output word, into an error pattern $E \cdot \mathbf{P}^T$ with length n_p, including uniform l-bit burst errors. That is, we will use a matrix **P** that is an $n_p \times (n_0 + n_1)$ conversion matrix: matrix **P** converts l-bit burst errors in X_0 into the same l-bit burst errors in the former part with length n_0 of the output word, and also converts single-bit errors in X_1 into l-bit burst errors in the latter part with length $n_p - n_0$ of the output word. These converted errors in the output word with length n_p can be corrected by applying an $(n_p, n_p - r)$ l-bit burst error correcting code. In order to prevent l-bit burst errors in X_1 from being mistaken as correctable errors in X_0, the matrix **P** converts the burst errors in X_1 to the l-bit burst errors, that is, correctable errors, in the latter part of the output word.

Theorem 10.15 *Let $\boldsymbol{H}_{BEC}$ be a parity-check matrix of an $(n_p, n_p - r)$ l-bit burst error correcting code. Then the null space of*

$$\boldsymbol{H} = \boldsymbol{H}_{BEC} \cdot \boldsymbol{P}$$

is an $(n_0 + n_1,\ n_0 + n_1 - r)$ $(B_lEC)_{n_0}$-$(SEC)_{n_1}$ UEP code, where



$\boldsymbol{I}_{n_0}$ *: $n_0 \times n_0$ identity matrix,*

$\boldsymbol{O}$ *: zero matrix,*

$$\boldsymbol{H}_A = \begin{bmatrix} 1 \cdots 1 \\ \boldsymbol{Q} \\ 1 \cdots 1 \end{bmatrix}_{l \times q},$$

$$\boldsymbol{H}_B = \begin{bmatrix} \boldsymbol{Q} \\ 1 \cdots 1 \end{bmatrix}_{(l-1) \times q},$$

$\boldsymbol{Q}$: $(l-2) \times q$ *matrix having distinct* q *binary columns where* $l - 1 \leq q \leq 2^{l-2}$,

$\boldsymbol{H}_\Gamma$: $(l-1) \times q'$ *matrix having distinct* q' *binary columns where the element* $h_{i,j}$ *in* $\boldsymbol{H}_\Gamma$ *equals zero for* $i > j$ *and* $l - 1 \leq q' \leq 2^{l-1} - 1$.

Proof The syndrome caused by error E is expressed as

$$E \cdot \mathbf{H}^T = E \cdot (\mathbf{H}_{\text{BEC}} \cdot \mathbf{P})^T = \left(E \cdot \boldsymbol{P}^T\right) \cdot \mathbf{H}_{\text{BEC}}^T,$$

where E is an error vector with length $n_0 + n_1$ and output of $E \cdot \mathbf{P}^T$ is called a converted error of E.

From the structure of the matrix $\mathbf{P}$, any error that occurs in X_0 is always converted to the original error E in $E \cdot \mathbf{P}^T$, because there exists an identity submatrix $\mathbf{I}_{n_0}$ in $\mathbf{P}$. That is, the l-bit burst errors occurring in X_0 give the converted errors including the input l-bit burst error e' in the former part of $E \cdot \mathbf{P}^T$:

$$E \cdot \mathbf{P}^T = (\underbrace{0 \cdots 0 \overbrace{e'}^{l} 0 \cdots 0}_{n_0} \vdots \underbrace{0 \cdots 0}_{n_p - n_0}).$$

These converted errors can be corrected by the l-bit burst error correcting code expressed by $\mathbf{H}_{\text{BEC}}$. Hence the code satisfies conditions 1 and 2 of Theorem 10.13 for the error set $\boldsymbol{E}_{B0}$.

When l-bit burst errors corrupt both X_0 and X_1, that is

$$E = (\underbrace{0 \cdots 0 \overbrace{e_0}^{w} \vdots \overbrace{e_1}^{l-w}}_{n_0} \underbrace{0 \cdots 0}_{n_1}),$$

then the former part of the error, e_0, with length w ($< l$) bits is converted to the original error e_0 and the remaining part of the error, e_1, with a length of $l - w$ bits in X_1 is converted to an error $e^\dagger$ with a length of at most $l - w$ bits because of the property that $h_{i,j} = 0$ for $i > j$ in the matrix $\mathbf{H}_\Gamma$. That is, the converted error is an l-bit burst error, shown as

$$E \cdot \mathbf{P}^T = (\underbrace{0 \cdots 0 \overbrace{e_0}^{l}}_{n_0} \vdots \underbrace{e^\dagger 0 \cdots 0}_{n_p - n_0}).$$

Especially in this case, the former part of the converted errors can be properly corrected, but the remaining part is not guaranteed to be properly corrected in X_1. Hence the code satisfies conditions 1, 3, and 4 of Theorem 10.13 for the error set $\boldsymbol{E}_B$.

Nonzero column vectors in $\mathbf{P}$ are all distinct, and therefore every single-bit error E is always converted to different pattern of $E \cdot \mathbf{P}^T$. Single-bit errors in X_0 are always converted to single-bit errors, and the errors in X_1 are converted to l-bit burst errors because of the matrix structure of $\mathbf{H}_\Gamma$, $\mathbf{H}_A$, and $\mathbf{H}_B$, which must have l or fewer nonzero rows. Because these converted errors can be properly corrected, the code satisfies conditions 1 and 5 of Theorem 10.13 for the error set $\boldsymbol{E}_b$.

When l-bit burst errors occur in X_1, the latter part of $E \cdot \mathbf{P}^T$ with length $(n_p - n_0)$ can include a zero pattern or a nonzero pattern of e^* having at most l 1's:

$$E \cdot \mathbf{P}^T = (\underbrace{0 \cdots 0}_{n_0} \vdots \underbrace{0 \cdots 0 \overbrace{e^*}^{l} 0 \cdots 0}_{n_p - n_0}).$$

It is apparent that the syndrome caused by l-bit burst errors in X_1 is different from the syndrome due to errors in X_0, and the syndrome due to errors spanned over X_0 and X_1. Hence the code satisfies conditions 6 and 7 of Theorem 10.13. Consequently the $\mathbf{H}$ matrix indicated in the theorem satisfies the conditions of Theorem 10.13, so the code is a $(\mathrm{B}_l\mathrm{EC})_{n_0}$-$(\mathrm{SEC})_{n_1}$ UEP code. Q.E.D.

Example 10.5 [NAMB02]: (55, 44) $(\mathrm{B}_4\mathrm{EC})_8$-$(\mathrm{SEC})_{47}$ UEP Code

The parity-check matrix of a (16, 5) 4-bit burst error correcting Fire code is shown below:

$$\mathbf{H}_{\mathrm{BEC}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{matrix} \uparrow \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \downarrow \end{matrix} r = 11.$$

$$\longleftarrow \qquad n_p = 16 \qquad \longrightarrow$$

The conversion matrix $\mathbf{P}$ and the resultant parity-check matrix of the (55, 44) $(\mathrm{B}_4\mathrm{EC})_8$-$(\mathrm{SEC})_{47}$ UEP code in a binary form are shown in Figures 10.13 and 10.14, respectively.

Theorem 10.16 *The code parameters l, n_p, n_0, and n_1 in Theorem 10.15 satisfy the following inequality:*

$$n_p \geq n_0 + \left\lceil \frac{n_1 + 1}{2^{l-1}} \right\rceil + l - 2, \tag{10.9}$$

where $\lceil x \rceil$ shows the smallest integer larger than or equal to x.

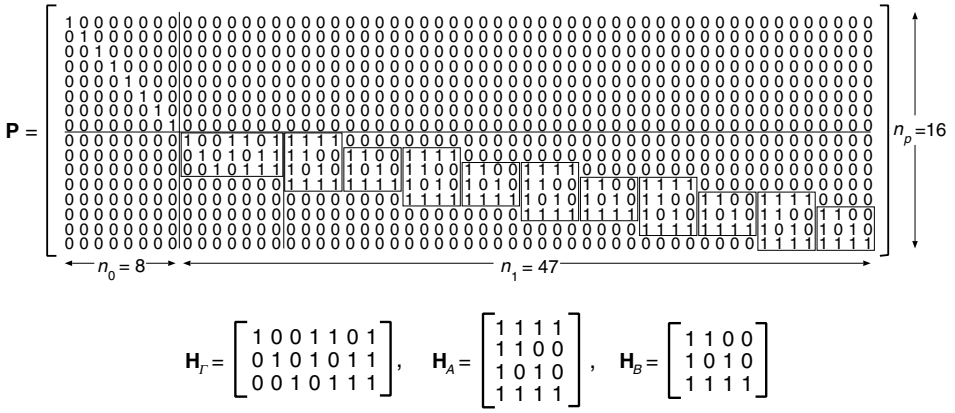


Figure 10.13 *The conversion matrix* ***P****.* Source: *[NAMB02]. © 2002 IEICE Japan.*

Proof Length n_1 is less than or equal to the sum of the following three values: the product value of the number of $\mathbf{H}_A$ matrices in matrix $\mathbf{P}$ and the number of columns in $\mathbf{H}_A$, the product value of the number of $\mathbf{H}_B$ matrices in matrix $\mathbf{P}$ and the number of columns in $\mathbf{H}_B$, and the number of columns in $\mathbf{H}_\Gamma$. The maximum number of columns in $\mathbf{H}_A$ is equal to that in $\mathbf{H}_B$, meaning 2^{l-2}. The number of $\mathbf{H}_A$ matrices in $\mathbf{P}$ is equal to that of $\mathbf{H}_B$ matrices in $\mathbf{P}$, meaning $n_p - n_0 - l + 1$. The maximum number of columns in $\mathbf{H}_\Gamma$ is $2^{l-1} - 1$. Consequently we have the inequality

$$n_1 \leq \left(n_p - n_0 - l + 2\right) \cdot 2^{l-1} - 1.$$

By simply re-arranging the variables, we obtain the inequality (10.9). Q.E.D.

Decoding Procedure The decoding proceduce is demonstrated for the $(\mathrm{B}_l\mathrm{EC})_{n_0}$-$(\mathrm{SEC})_{n_1}$ UEP code. The syndromes for the received word V that may include error E are expressed as

$$\begin{aligned} S &= \left(V \cdot \mathbf{P}^T\right) \cdot \mathbf{H}_{\mathrm{BEC}}^T \\ &= (V_0 + E) \cdot \mathbf{P}^T \cdot \mathbf{H}_{\mathrm{BEC}}^T \\ &= \left(E \cdot \mathbf{P}^T\right) \cdot \mathbf{H}_{\mathrm{BEC}}^T, \end{aligned}$$

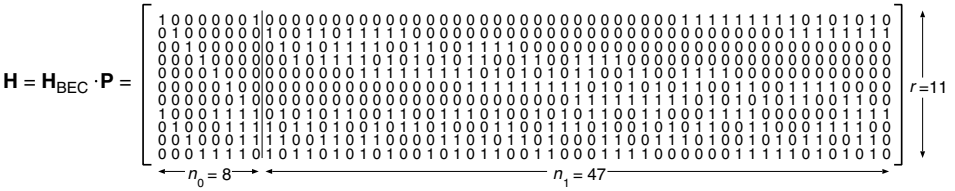


Figure 10.14 *Parity-check matrix of the (55, 44)* $(B_4EC)_8$*-*$(SEC)_{47}$ *UEP code.* Source: *[NAMB02]. © 2002 IEICE Japan.*

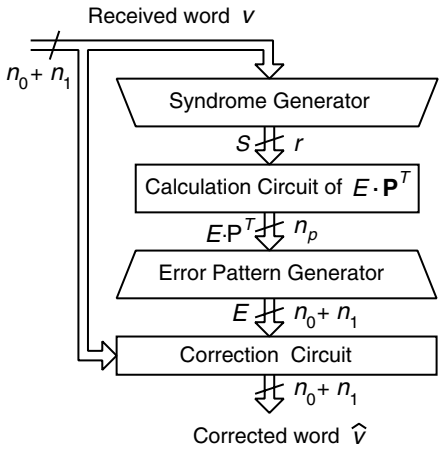


Figure 10.15 *Block diagram of the decoding circuit.* Source: *[NAMB02]. © 2002 IEICE Japan.*

where V_0 is a transmitted word. The decoding is performed by the following four steps:

Step 1. Generate syndrome $S = \left(V \cdot \mathbf{P}^T\right) \cdot \mathbf{H}_{\mathrm{BEC}}^T$.
Step 2. Calculate $E \cdot \mathbf{P}^T$ from syndrome S.
Step 3. Calculate the correctable error pattern E from $E \cdot \mathbf{P}^T$.
Step 4. Correct the received word by $\hat{V} = V \oplus E$.

These operations are performed in parallel and implemented by combinational circuits. Figure 10.15 shows the block diagram of the decoding circuit.

Evaluation Figure 10.16 shows the relation between the information-bit length $k_1 = n_1 - r$ and the check-bit length r of the $(\mathrm{B}_4\mathrm{EC})_{32}$-$(\mathrm{SEC})_{n_1}$ UEP code with $l = 4$ bits and $n_0 = 32$ bits. Figure 10.16 also shows the bound given in Theorem 10.16.

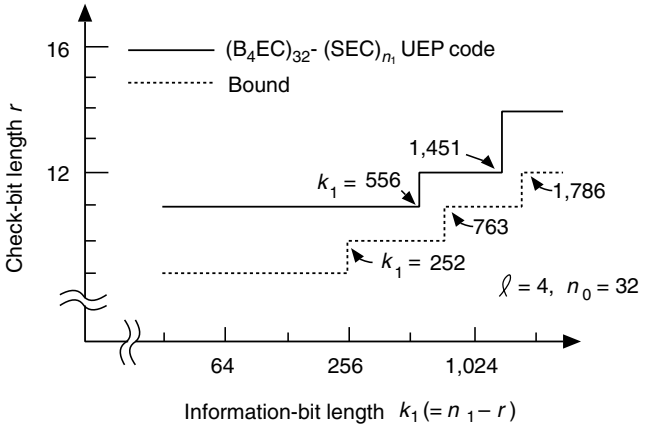


Figure 10.16 *Comparison of the information-bit lengths and check-bit lengths of the $(B_4EC)_{32}$-$(SEC)_{n_1}$ UEP code.* Source: *[NAMB02]. © 2002 IEICE Japan.*

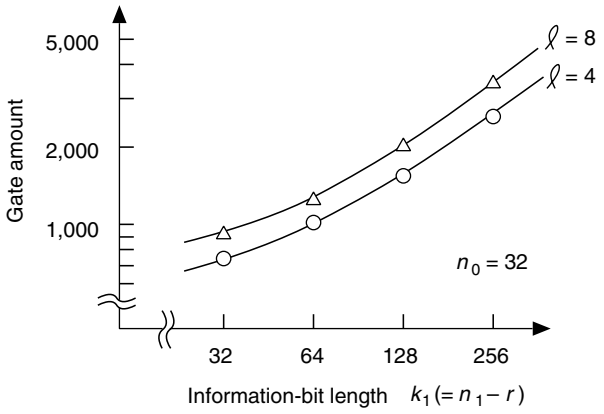


Figure 10.17 *Parallel decoder gate amount of the $(B_lEC)_{32}$-$(SEC)_{n_1}$ UEP code.* Source: *[NAMB02]. © 2002 IEICE Japan.*

Figure 10.17 shows the amount of decoder hardware for the $(B_lEC)_{32}$-$(SEC)_{n_1}$ UEP code with $n_0 = 32$ bits, $l = 4$ and 8 bits. In this case, a four-input AND/OR gate is counted as one gate and a two-input exclusive-OR (XOR) gate as 1.5 gates.

10.4 APPLICATION OF THE UEC / UEP CODES

10.4.1 Application of *q*-Ary UEC Codes to Holographic Memories

In two-dimensional storage media the bit error rate (BER) is not equal in general in overall area of the media. For example, in holographic memories the BER of readout data from the edge of the media is much higher than that from the center of the media [CHOU98, BETZ98, ASHL00]. This type of error can be effectively corrected using unequal error control (UEC) codes. This subsection demonstrates two classes of optimal q-ary UEC codes whose codewords have two disjoint areas with distinct error control capabilities [KANE03]. Each q-ary symbol in a codeword of the UEC code is mapped to a codeword of a block modulation code.

Conventional error control codes, such as Reed-Solomon codes, BCH codes, and Fire codes, are not suitable for such two-dimensional storage media because they require a large number of check bits to provide a uniform error correction strong enough to correct errors in the edge of the media. In contrast, two-level binary unequal error control (UEC) codes are suitable for this type of storage media because a codeword of the UEC code has two disjoint areas with distinct error control capabilities: the area X_0 with a strong error control capability and the remaining area X_1 with a moderate error control capability.

In addition to the error control codes, optical storage systems employ modulation codes to improve the signal-to-noise ratio (SNR) of the readout signals and finally to reduce the BER of the readout data. Typical modulation codes applied to the holographic memories are the *balanced codes*, the *low-pass filtering codes* [BURR97], and the *sparse modulation codes* [KING00]. A balanced code is a block code of length n whose codewords have a constant Hamming weight $\lfloor n/2 \rfloor$, where $\lfloor x \rfloor$ shows the largest integer less than or equal to x. These codes are effective in reducing the errors that occur in the binarization stage of the readout gray-scale signal. That is, these gray-scale signals can be binarized without using

an explicit threshold value; for example, the brightest $\lfloor n/2 \rfloor$ pixels can be set to 1 and the remaining ones set to 0. A low-pass filtering code tolerates high-frequency error-prone recording patterns such as a dark pixel surrounded by bright pixels, and thus improves the SNR of the readout signals. A sparse modulation code generates binary sequences with low Hamming weight; this reduces the number of bright pixels in a recording medium, thereby improving the SNR of the holographic memories [KING00].

Error control coding and block modulation coding can be combined using q-ary error control codes, where each q-ary symbol in a codeword of the error control code is mapped to a codeword of the block modulation code [TILB89]. Here a q-ary symbol is simply referred to as a symbol. This subsection deals with a new class of two-level q-ary UEC codes whose codeword consists of two disjoint areas X_0 and X_1. As indicated in Section 10.2, X_0 and X_1 correspond to the areas with high symbol error rate (SER) and low SER, respectively. The area X_0 is called here a *fixed-area*, and any error confined to X_0 is referred to here as a *fixed-area error*. Without loss of generality, X_0 is located in the leftmost l symbols of a codeword, where l is the length (in symbols) of the fixed-area, and X_1 has the remaining $n - l$ symbols. This subsection presents the following two classes of two-level q-ary UEC codes:

1. *Fixed l-symbol Error Correcting | Single-symbol Error Correcting codes* ($\mathrm{F}_l\mathrm{EC}|\mathrm{SEC}$ codes). These codes are capable of correcting fixed-area errors in X_0 as well as correcting single-symbol errors in X_1, and
2. *Fixed l-symbol plus Single-symbol Error Correcting codes* ($(\mathrm{F}_l+\mathrm{S})\mathrm{EC}$ codes). These codes are capable of correcting both fixed-area errors in X_0 and single-symbol errors in X_1 simultaneously.

1. Combination of Error Control Coding and Block Modulation Coding for Holographic Memories

Before designing q-ary UEC codes for holographic memory systems, an encoding process that employs a q-ary UEC code combined with a block modulation code is clarified here. Let $\mathbf{C}_M$ be a block modulation code having $|\mathbf{C}_M|$ codewords, let q be the largest prime or power of prime satisfying $q \leq |\mathbf{C}_M|$, and also let b be the largest integer satisfying $2^b \leq q$. A binary information word $\mathbf{D}$ composed of k vectors is expressed as

$$\mathbf{D} = (D_0\ D_1\ \ldots\ D_{k-1}),$$

where $D_i = (d_{i,0}\ d_{i,1}\ \ldots\ d_{i,b-1})$, $0 \leq i \leq k-1$, is a binary vector with length b. The information word $\mathbf{D}$ thus has a total length of bk bits. The encoding process for $\mathbf{D}$ is constituted by the following steps [KANE03]:

Step 1. Each binary vector D_i in the information word $\mathbf{D}$ is mapped to an element of $GF(q)$, and hence an information word over $GF(q)$ is constituted by these mapped elements. Note that the mapping from the set of binary vectors D_i to $GF(q)$ should be injective.

Step 2. The information word over $GF(q)$ is encoded by a q-ary UEC code, which reduces to a codeword over $GF(q)$.

Step 3. Each q-ary symbol in the codeword over $GF(q)$ is mapped to a codeword of the modulation code $\mathbf{C}_M$. The mapping from $GF(q)$ to $\mathbf{C}_M$ should be injective. The binary sequence of the resulting word is the recording pattern.

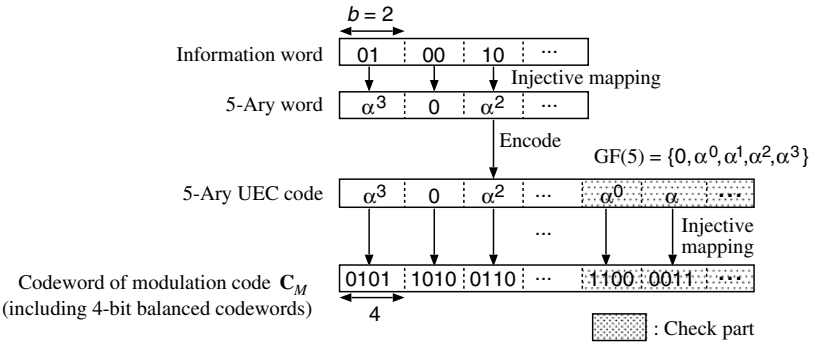


Figure 10.18 *Combination of error control coding and modulation coding by using 5-ary UEC code.* Source: *[KANE03]. © 2003 IEEE.*

Figure 10.18 illustrates an example of the encoding process, where $\mathbf{C}_M$ is the balanced code with length 4 bits, $|\mathbf{C}_M| = 6$, $q = 5$, $b = 2$, and α is a primitive element in $GF(5)$.

2. q-Ary $F_l EC|SEC$ Codes

Let's design the linear q-ary F_lEC|SEC codes capable of correcting any fixed-area errors in X_0 as well as correcting single-symbol errors in X_1 [KANE03].

Preliminaries We first consider the necessary and sufficient conditions of the linear q-ary F_lEC|SEC codes and then derive an upper bound on code length.

Let $\mathbf{E}_0$ and $\mathbf{E}_1$ be sets of vectors over $GF(q)$ having length n and defined as follows:

$$\mathbf{E}_0 = \{(x_0\ x_1\ \ldots\ x_{n-1}) \mid 1 \le w_{\mathrm{H}}(x_0\ \ldots\ x_{l-1}) \le l,\ (x_l\ \ldots\ x_{n-1}) = o\},$$
$$\mathbf{E}_1 = \{(x_0\ x_1\ \ldots\ x_{n-1}) \mid (x_0\ \ldots\ x_{l-1}) = o,\ w_{\mathrm{H}}(x_l\ \ldots\ x_{n-1}) = 1\},$$

where $x_i \in GF(q)$ for $0 \le i \le n-1$, $o = (0\ \ldots\ 0)$, and $w_{\mathrm{H}}(X)$ is the Hamming weight of vector X. Here $\mathbf{E}_0$ represents the set of fixed-area error patterns in X_0, and $\mathbf{E}_1$ represents the set of single-symbol error patterns in X_1. The following theorem gives the necessary and sufficient conditions for the linear q-ary F_lEC|SEC codes.

Theorem 10.17 *The null space of a parity-check matrix $\boldsymbol{H}$ over $GF(q)$ is a linear q-ary $F_l EC|SEC$ code if and only if:*

1. $E \cdot \boldsymbol{H}^T \neq o$ *for all* $E \in (\boldsymbol{E}_0 \cup \boldsymbol{E}_1)$,
2. $E_p \cdot \boldsymbol{H}^T \neq E_q \cdot \boldsymbol{H}^T$ *for all* $E_p, E_q \in \boldsymbol{E}_0, E_p \neq E_q$,
3. $E_i \cdot \boldsymbol{H}^T \neq E_j \cdot \boldsymbol{H}^T$ *for all* $E_i, E_j \in \boldsymbol{E}_1, E_i \neq E_j$,
4. $E_p \cdot \boldsymbol{H}^T \neq E_i \cdot \boldsymbol{H}^T$ *for all* $E_p \in \boldsymbol{E}_0$, *and for all* $E_i \in \boldsymbol{E}_1$.

The proof is left to the reader.

Theorem 10.18 *The code length in symbols n and the number of check symbols r of a linear q-ary $F_l EC|SEC$ code satisfy the following inequality:*

$$n \le \frac{q^r - q^l}{q - 1} + l. \tag{10.10}$$

TABLE 10.6 Upper Bounds on the Information-Symbol Lengths of the $F_\ell EC|SEC$ Codes over *GF* (5)

r	l					
	3	4	5	6	7	8
4	124	—	—	—	—	—
5	748	624	—	—	—	—
6	3,872	3,748	3,124	—	—	—
7	19,496	19,372	18,748	15,624	—	—
8	97,620	97,496	96,872	93,748	78,124	—
9	488,244	488,120	487,496	484,372	468,748	390,624

Note: r: number of check symbols

Proof Condition 2 of Theorem 10.17 says that all syndromes caused by fixed-area errors in X_0 should be distinct, where there exist $q^l - 1$ fixed-area error patterns. Condition 3 says that all syndromes for single-symbol errors in X_1 should be distinct, where there exist $(q-1)(n-l)$ single-symbol error patterns. Therefore the total number of nonzero syndromes (i.e., $q^r - 1$) should satisfy the following inequality:

$$q^r - 1 \geq (q^l - 1) + (q-1)(n-l).$$

After re-arranging this inequality , the relation (10.10) is derived. Q.E.D.

Table 10.6 shows the upper bound on the information-symbol length $k = n - r$ of the $F_lEC|SEC$ codes over $GF(5)$ for $4 \leq r \leq 9$ and $3 \leq l \leq 8$.

Code Design Let $\mathbf{H}'_{r-l}$ be a parity-check matrix of a single-symbol error correcting Hamming code over $GF(q)$ with $r - l$ check symbols, defined as

$$\mathbf{H}'_{r-l} = [h'_0 \quad h'_1 \quad \cdots \quad h'_{n'-1}]_{(r-l)\times n'},$$

where h'_i is a column vector over $GF(q)$ with length $r - l$ for $0 \leq i \leq n' - 1$, and $n' = (q^{r-l} - 1)/(q-1)$. Note that $\mathbf{H}' = [1]$ for $r - l = 1$. By using one column vector in $\mathbf{H}'_{r-l}$, the matrix $\mathbf{Q}_i$ is defined as

$$\mathbf{Q}_i = [h'_i \quad h'_i \quad \cdots \quad h'_i]_{(r-l)\times q^l},$$

where $i \in \{0,\ 1,\ \ldots,\ n' - 1\}$.

Theorem 10.19 *The null space of*

$$\boldsymbol{H} = [\boldsymbol{H}_0 \,|\, \boldsymbol{H}_1] = \left[\begin{array}{c|cccc} \boldsymbol{I}_l & \boldsymbol{M} & \boldsymbol{M} & \cdots & \boldsymbol{M} \\ \boldsymbol{O}_{(r-l)\times l} & \boldsymbol{Q}_0 & \boldsymbol{Q}_1 & \cdots & \boldsymbol{Q}_{n'-1} \end{array}\right]$$

is an optimal linear $F_lEC|SEC$ code over $GF(q)$ satisfying the upper bound on code length given by the relation (10.10), where submatrices in $\boldsymbol{H}$ are defined as follows:

$\boldsymbol{I}_l$: $l \times l$ *identity matrix*,
$\boldsymbol{O}_{(r-l)\times l}$: $(r-l) \times l$ *zero matrix*,

$\boldsymbol{M}$: $l \times q^l$ *matrix with all distinct column vectors over GF(q),*

$$\boldsymbol{H}_0 = \begin{bmatrix} \boldsymbol{I}_l \\ \boldsymbol{O}_{(r-l)\times l} \end{bmatrix},$$

$$\boldsymbol{H}_1 = \begin{bmatrix} \boldsymbol{M} & \boldsymbol{M} & \cdots & \boldsymbol{M} \\ \boldsymbol{Q}_0 & \boldsymbol{Q}_1 & \cdots & \boldsymbol{Q}_{n'-1} \end{bmatrix}$$

Proof This theorem can be proved such that the code satisfies conditions 1 to 4 of Theorem 10.17. Since $\mathbf{I}_l$ is a nonsingular matrix and all column vectors in $\mathbf{H}_1$ are nonzero, conditions 1 and 2 are satisfied. Condition 3 is also satisfied because $\mathbf{H}_1$ is a parity-check matrix of a shortened single-symbol error correcting Hamming code. In order to prove that the code satisfies condition 4, let $S = (s_0\ s_1\ \ldots\ s_{l-1}\ s_l\ \ldots\ s_{r-1}) = E \cdot \mathbf{H}^T$ be a syndrome caused by an error pattern E. If $E \in \mathbf{E}_0$, then $(s_l\ s_{l+1}\ \ldots\ s_{r-1}) = (0\ 0\ \ldots\ 0)$, but, if $E \in \mathbf{E}_1$, then $(s_l\ s_{l+1}\ \ldots\ s_{r-1}) \neq (0\ 0\ \ldots\ 0)$. Therefore condition 4 is satisfied because the last $r - l$ symbols of the syndrome S caused by fixed-area errors in X_0 are different from those of the single-symbol errors in X_1.

The maximum code length n of the code can be determined by counting the number of column vectors in $\mathbf{H}$, which is written as

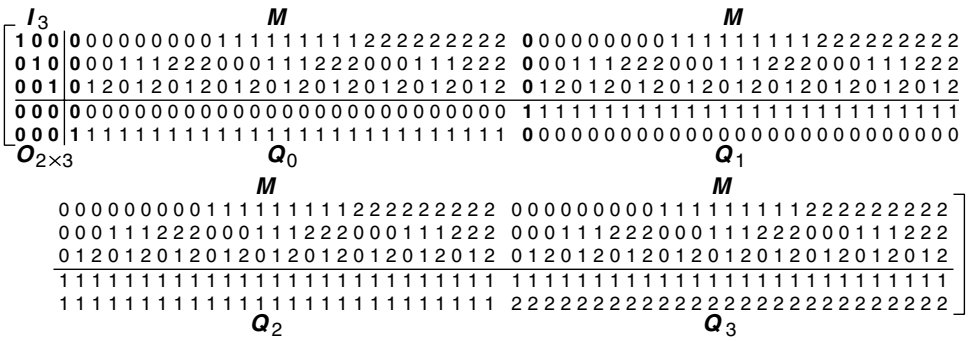
$$n = \frac{q^{r-l} - 1}{q - 1} \times q^l + l = \frac{q^r - q^l}{q - 1} + l.$$

Therefore the code is optimal from the relation (10.10). Q.E.D.

The F_lEC|SEC code given in Theorem 10.19 is systematic because $\mathbf{H}$ has r distinct column vectors, each having Hamming weight one. The F_lEC|SEC codes over $GF(q)$ coincide with the binary F_bEC|SEC codes when $q = 2$, mentioned in Subsection 10.2.1.

Example 10.6

Figure 10.19 shows the parity-check matrix of the (111, 106) F_3EC|SEC code over $GF(3)$ designed by using the following parity-check matrix of the single-symbol error



Column vectors indicated by bold font correspond to check part

Figure 10.19 *Parity-check matrix of the (111, 106) F_3EC|SEC code over $GF(3)$.*

correcting Hamming code over $GF(3)$,

$$\mathbf{H}_2' = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 \end{bmatrix}.$$

In Fig. 10.19, the column vectors indicated by bold font correspond to check part.

Decoding Procedure For a received word $U' = (u_0' \; u_1' \; \ldots \; u_{n-1}')$, the syndrome S is defined by

$$\begin{aligned} S &= (S_F \; S_P) \\ &= (s_0 \; s_1 \; \ldots \; s_{l-1} \; s_l \; s_{l+1} \; \ldots \; s_{r-1}) = U' \cdot \mathbf{H}^T, \end{aligned}$$

where $S_F = (s_0 \; s_1 \; \ldots \; s_{l-1})$ and $S_P = (s_l \; s_{l+1} \; \ldots \; s_{r-1})$. The received word U' is then decoded as follows:

Step 1. If $S = 0$, then U' has no error.

Step 2. If $S_F \neq 0$ and $S_P = 0$, then a fixed-area error exists in X_0 and is corrected using the error pattern given by S_F.

Step 3. If $S_P \neq 0$, then a single-symbol error exists in X_1. In this case, S is a multiple of a column vector in $\mathbf{H}_1$, in particular, $\exists i \in \{0, 1, \cdots, n-l-1\}$ such that $S = wv_i$, where v_i is the i-th column vector in $\mathbf{H}_1$ and $w \in GF(q) - \{0\}$. Since any two column vectors in $\mathbf{H}_1$ are linearly independent, w and v_i can be uniquely determined from S, and thus a single-symbol error in X_1 can be corrected.

3. q-Ary (F_l+S)EC Codes

Next we turn to design the linear q-ary (F_l+S)EC codes capable of correcting both fixed-area errors in X_0 and single-symbol errors in X_1 that occur simultaneously, called *q-ary fixed-area plus single-symbol error correcting codes* [KANE03].

Preliminaries Let us consider the necessary and sufficient conditions of the linear (F_l+S)EC codes, and derive the upper bound on code length.

Theorem 10.20 *The null space of a parity-check matrix $\boldsymbol{H}$ over GF(q) is a linear q-ary (F_l+S)EC code if and only if:*

1. $E \cdot \boldsymbol{H}^T \neq o \quad$ *for all* $E \in (\boldsymbol{E}_0 \cup \boldsymbol{E}_1)$,
2. $E_p \cdot \boldsymbol{H}^T \neq E_q \cdot \boldsymbol{H}^T \quad$ *for all* $E_p, E_q \in \boldsymbol{E}_0, \; E_p \neq E_q$,
3. $E_i \cdot \boldsymbol{H}^T \neq E_j \cdot \boldsymbol{H}^T \quad$ *for all* $E_i, E_j \in \boldsymbol{E}_1, \; E_i \neq E_j$,
4. $E_p \cdot \boldsymbol{H}^T \neq E_i \cdot \boldsymbol{H}^T \quad$ *for all* $E_p \in \boldsymbol{E}_0$, *and for all* $E_i \in \boldsymbol{E}_1$,
5. $(E_i + E_p) \cdot \boldsymbol{H}^T \neq (E_j + E_q) \cdot \boldsymbol{H}^T \quad$ *for all* $E_p, E_q \in \boldsymbol{E}_0$, *and for all* $E_i, E_j \in \boldsymbol{E}_1$, $E_p \neq E_q, \; E_i \neq E_j$,

where $\boldsymbol{E}_0$ and $\boldsymbol{E}_1$ are the sets of q-ary vectors defined in the previous F_lEC|SEC codes.

Proof Conditions 1 and 2 are necessary and sufficient conditions for correcting fixed-area errors in X_0, conditions 1 and 3 are those for correcting single-symbol errors in X_1, and condition 4 is that for discriminating between the fixed-area errors and the single-symbol errors. Condition 4 also provides those for discriminating between the fixed-area X_0 errors and the fixed area plus single-symbol errors because it follows from the condition that $E'_p \cdot \mathbf{H}^T \neq (E_i + E''_p) \cdot \mathbf{H}^T$, where $E_p, E''_p \in \mathbf{E}_0$ and $E_i \in \mathbf{E}_1$. Condition 5 provides those for correcting errors in the fixed-area plus single-symbol errors; it also provides those for discriminating between the single-symbol errors and the errors in the fixed-area plus single-symbol errors because it includes the relation $E_i \cdot \mathbf{H}^T \neq (E_j + E'_q) \cdot \mathbf{H}^T$, where $E'_q \in \mathbf{E}_0$ and $E_i, E_j \in \mathbf{E}_1$. Therefore conditions 1 through 5 are necessary and sufficient conditions of the linear q-ary (F_l+S)EC codes. Q.E.D.

Theorem 10.21 *The code length in symbols n and the number of check-symbols r of a liner q-ary (F_l+S)EC code satisfy the following inequality:*

$$n \leq \frac{q^{r-l} - 1}{q - 1} + l. \tag{10.11}$$

Proof Theorem 10.20 says that all syndromes for fixed-area errors in X_0, single-symbol errors in X_1, and fixed-area plus single-symbol errors should be distinct. There exist $q^l - 1$ error patterns in X_0, $(q-1)(n-l)$ single-symbol error patterns in X_1, and $(q^l - 1) \times (q-1)(n-l)$ fixed-area plus single-symbol error patterns. Thus the following inequality holds:

$$q^r - 1 \geq (q^l - 1) + (q-1)(n-l) + (q^l - 1)(q-1)(n-l).$$

By re-arranging this inequality, the relation (10.11) is derived. Q.E.D.

Table 10.7 shows the upper bound on information-symbol length $k = n - r$ of the (F_l+S)EC codes over $GF(5)$ for $5 \leq r \leq 12$ and $3 \leq l \leq 7$.

TABLE 10.7 Upper Bounds on Information-Symbol Length of (F_l+S)EC Codes over *GF*(5)

r	ℓ				
	3	4	5	6	7
5	4	—	—	—	—
6	28	4	—	—	—
7	152	28	4	—	—
8	776	152	28	4	—
9	3,900	776	152	28	4
10	19,524	3,900	776	152	28
11	97,648	19,524	3,900	776	152
12	488,272	97,648	19,524	3,900	776

Note: r: number of check symbols.

Code Design

Theorem 10.22 *The null space of*

$$\boldsymbol{H} = \left[\boldsymbol{H}_0 \,\middle\|\, \boldsymbol{H}_1\right] = \left[\begin{array}{c\|c|c|c} \boldsymbol{I}_l & \boldsymbol{O}_{l\times(n-l-r)} & \boldsymbol{I}_l & \boldsymbol{O}_{l\times(r-l)} \\ \hline \boldsymbol{P} & \boldsymbol{Q} & \boldsymbol{O}_{(r-l)\times l} & \boldsymbol{I}_{r-l} \end{array}\right]$$

is an optimal linear (F_l+S)EC code over $GF(q)$ satisfying the upper bound on code length given by the relation (10.11), where the submatrices in $\boldsymbol{H}$ are defined as follows:

$\boldsymbol{I}_x$: $x \times x$ *identity matrix*,
$\boldsymbol{O}_{y\times z}$: $y \times z$ *zero matrix*,
$[\boldsymbol{P} \mid \boldsymbol{Q} \mid \boldsymbol{I}_{r-l}]$: *parity-check matrix of a systematic single-symbol error correcting Hamming code over $GF(q)$ having $r - l$ check symbols*,

$$\boldsymbol{H}_0 = \left[\frac{\boldsymbol{I}_l}{\boldsymbol{P}}\right],$$

$$\boldsymbol{H}_1 = \left[\begin{array}{c|c|c} \boldsymbol{O}_{l\times(n-l-r)} & \boldsymbol{I}_l & \boldsymbol{O}_{l\times(r-l)} \\ \hline \boldsymbol{Q} & \boldsymbol{O}_{(r-l)\times l} & \boldsymbol{I}_{r-l} \end{array}\right].$$

Proof This can be proved such that the code satisfies conditions 1 through 5 of Theorem 10.20. Conditions 1, 2, and 3 are satisfied because $\mathbf{I}_l$ is a nonsingular matrix, and any two column vectors in $\mathbf{H}_1$ are linearly independent. In order to prove that the code satisfies condition 4, let $S = (S_F\ S_P) = (s_0\ s_1\ \ldots\ s_{l-1}\ s_l\ \ldots\ s_{r-1}) = E \cdot \mathbf{H}^T$ be a syndrome caused by an error pattern E, where $S_F = (s_0\ s_1\ \ldots\ s_{l-1})$ and $S_P = (s_l\ s_{l+1}\ \ldots\ s_{r-1})$. Syndromes caused by fixed-area errors $E \in \boldsymbol{E}_0$ satisfy the following condition:

$$[w_{\mathrm{H}}(S_F) \geq 2] \vee [\,[w_{\mathrm{H}}(S_F) = 1] \wedge [w_{\mathrm{H}}(S_P) \geq 2]].$$

Note that the latter condition holds because the Hamming weight of a column vector in $\mathbf{P}$ is greater than or equal to two. In contrast, syndromes caused by single-symbol errors $E \in \mathbf{E}_1$ satisfy the following condition:

$$[w_{\mathrm{H}}(S_F) = 0] \vee [[w_{\mathrm{H}}(S_F) = 1] \wedge [w_{\mathrm{H}}(S_P) = 0]].$$

Therefore the fixed-area errors in X_0 can be discriminated from the single-symbol errors in X_1, thus satisfying condition 4. Condition 5 can be proved by contradiction; we assume that the condition does not hold, then for $\exists E_p, E_q \in \boldsymbol{E}_0$ and $\exists E_i, E_j \in \boldsymbol{E}_1$ the following relation holds:

$$\left(E_p - E_q\right) \cdot \mathbf{H}^T = \left(-E_i + E_j\right) \cdot \mathbf{H}^T, \tag{10.12}$$

where $E_p \neq E_q$ and $E_i \neq E_j$. Each row vector $(E_p - E_q) \cdot \mathbf{H}^T$ and $(-E_i + E_j) \cdot \mathbf{H}^T$ is divided into two parts as follows:

$$(S_F\ S_P) = (E_p - E_q) \cdot \mathbf{H}^T,$$
$$(S'_F\ S'_P) = (-E_i + E_j) \cdot \mathbf{H}^T,$$

where S_F and S'_F are row vectors each having length l, and S_P and S'_P are also row vectors each having length $r - l$. For $(S_F\ S_P)$, exactly one of the following conditions holds:

a. $[w_{\mathrm{H}}(S_F) = 1] \wedge [S_P^T = \text{a column vector in } \mathbf{P}]$,
b. $[w_{\mathrm{H}}(S_F) = 2] \wedge [S_P \neq o]$,
c. $[w_{\mathrm{H}}(S_F) \geq 3]$.

For $(S'_F\ S'_P)$, exactly one of the following conditions holds:

a'. $[w_{\mathrm{H}}(S'_F) = 0]$,
b'. $[w_{\mathrm{H}}(S'_F) = 1] \wedge [S_P'^T = \text{a column vector in } \mathbf{Q}]$,
c'. $[w_{\mathrm{H}}(S'_F) = 1] \wedge [S_P'^T = \text{a column vector in } \mathbf{I}_{r-l}]$,
d'. $[w_{\mathrm{H}}(S'_F) = 2] \wedge [S'_P = o]$.

It follows from these two sets of conditions that $(S_P\ S_F) \neq (S'_P\ S'_F)$, which contradicts Eq. (10.12), and the assumption that condition 5 does not hold. Therefore condition 5 is satisfied.

The maximum code length n of the code can be determined by counting the number of column vectors in $\mathbf{H}$, which is written as

$$n = \frac{q^{r-l} - 1}{q - 1} + l.$$

Therefore the code is optimal by relation (10.11). Q.E.D.

Example 10.7

The following shows the parity-check matrix of the (16, 10) (F_3+S)EC code over $GF(3)$:

$$\mathbf{H} = \left[\begin{array}{c||c|c|c} \mathbf{I}_3 & \mathbf{O}_{3\times 7} & \mathbf{I}_3 & \mathbf{O}_{3\times 3} \\ \hline \mathbf{P} & \mathbf{Q} & \mathbf{O}_{3\times 3} & \mathbf{I}_3 \end{array}\right]$$

$$= \left[\begin{array}{c||c|c|c} 100 & 0000000 & 100 & 000 \\ 010 & 0000000 & 010 & 000 \\ 001 & 0000000 & 001 & 000 \\ \hline 001 & 1111111 & 000 & 100 \\ 110 & 0111222 & 000 & 010 \\ 121 & 2012012 & 000 & 001 \end{array}\right],$$

where $[\,\mathbf{P} \,|\, \mathbf{Q} \,|\, \mathbf{I}_3\,]$ is the parity-check matrix of the (13, 10) Hamming code over $GF(3)$.

Decoding Procedure By performing the row operations, the matrix $\mathbf{H}$ defined in Theorem 10.22 can be transformed into the following matrix $\mathbf{H}'$:

$$\mathbf{H}' = \left[\begin{array}{c||c|c|c} \mathbf{I}_l & \mathbf{O}_{l\times(n-l-r)} & \mathbf{I}_l & \mathbf{O}_{l\times(r-l)} \\ \hline \mathbf{O}_{(r-l)\times l} & \mathbf{Q} & -\mathbf{P} & \mathbf{I}_{r-l} \end{array}\right].$$

Note that the null space of $\mathbf{H}$ is identical to that of $\mathbf{H}'$, and therefore the matrices $\mathbf{H}$ and $\mathbf{H}'$ are the parity-check matrices of the identical ($F_l + S$)EC code. Decoding is performed by the matrix $\mathbf{H}'$. For a received word $U' = (u'_0\ u'_1\ \ldots\ u'_{n-1})$, the syndrome S is expressed as

$$\begin{aligned} S &= (S_F\ S_P) \\ &= (s_0\ s_1\ \ldots\ s_{l-1}\ s_l\ s_{l+1}\ \ldots\ s_{r-1}) = U' \cdot \mathbf{H}'^T, \end{aligned}$$

where $S_F = (s_0\ s_1\ \ldots\ s_{l-1})$ and $S_P = (s_l\ s_{l+1}\ \ldots\ s_{r-1})$. The received word U' is then decoded as follows:

Step 1. If $S = 0$, then U' has no error.

Step 2. If $S_F \neq 0$ and $S_P = 0$, then a fixed-area error exists in X_0 and is corrected by the error pattern given by S_F.

Step 3. If $S_P \neq 0$, then a single-symbol error or a fixed-area plus single-symbol error exists in U'. In this case the syndrome is expressed as follows:

$$S = (S_F\ S_P) = (E_p + E_i) \cdot \mathbf{H}'^T,$$

where $E_p \in \boldsymbol{E}_0 \cup \{o\}$ and $E_i \in \boldsymbol{E}_1$. Since any two column vectors in $[\mathbf{Q} \mid -\mathbf{P} \mid \mathbf{I}_{r-l}]$ are linearly independent, the single-symbol error pattern E_i in X_1 can be uniquely determined and thus corrected by S_P, regardless of whether or not a fixed-area error exists in X_0. In addition a fixed-area error pattern E_p can be determined simultaneously by the equation above because the first l columns in $\mathbf{H}'$ are also linearly independent.

4. Evaluation

The designed codes are evaluated in terms of the probability of *correct decoding*, that is, the probability of the received word U' being decoded correctly as the original codeword U. For the q-ary UEC codes and the RS codes over $GF(q)$, we are interested in the correct decoding of a received word. We denote, P_0 and P_1 as the symbol error rates in X_0 and in X_1, respectively.

***q*-Ary F_lEC|SEC Code** A received word U' is decoded correctly if and only if (1) X_1 has no error or (2) X_0 has no error and X_1 has a single-symbol error. Therefore the probability of correct decoding U' by the q-ary F_lEC|SEC code is

$$P_a = (1 - P_1)^{n-l} + (1 - P_0)^l \times (n - l) \times P_1 \times (1 - P_1)^{n-l-1}.$$

***q*-Ary (F_l+S)EC Code** A received word U' is decoded correctly if and only if (1) X_1 has no error or (2) X_1 has a single-symbol error. Therefore the probability of correct decoding of U' by the q-ary (F_l+S)EC code is

$$P_b = (1 - P_1)^{n-l} + (n - l) \times P_1 \times (1 - P_1)^{n-l-1}.$$

***t*-Symbol Error Correcting RS Code** A received word U' is decoded correctly if and only if the total number of errors in U' is less than or equal to t. Therefore the

probability of correct decoding of U' by the t-symbol error correcting RS code is

$$P_c(t) = \sum_{i=0}^{t} \sum_{j=0}^{i} \left\{ \binom{l}{j} P_0^j (1-P_0)^{l-j} \binom{n-l}{i-j} P_1^{i-j} (1-P_1)^{n-l-i+j} \right\},$$

where $\binom{x}{y}$ denotes a binomial coefficient defined as

$$\binom{x}{y} = \frac{x!}{y!(x-y)!}.$$

Comparison of Error Correction Capabilities Under the code parameters $q = 67$, $n = 66$, and $l = 6$, Figure 10.20 shows the probability of correct decoding for the following codes:

(a) 67-ary $(66, 59)$ F_6EC|SEC code,
(b) 67-ary $(66, 58)$ (F_6+S)EC code,
(c) 3-symbol error correcting $(66, 60)$ RS code over $GF(67)$,
(d) 4-symbol error correcting $(66, 58)$ RS code over $GF(67)$,

where $P_0 = 0.05$. The F_6EC|SEC code (a) has higher probability of correct decoding than the 3-symbol error correcting RS code (c) for $P_1 \leq 5.0 \times 10^{-6}$, and also higher than the 4-symbol error correcting RS code (d) for $P_1 \leq 1.0 \times 10^{-7}$. The (F_6+S)EC code (b) has higher probability of correct decoding than the 3-symbol error correcting RS code (c) for $P_1 \leq 2.5 \times 10^{-4}$, and also higher than the 4-symbol error correcting RS code (d) for

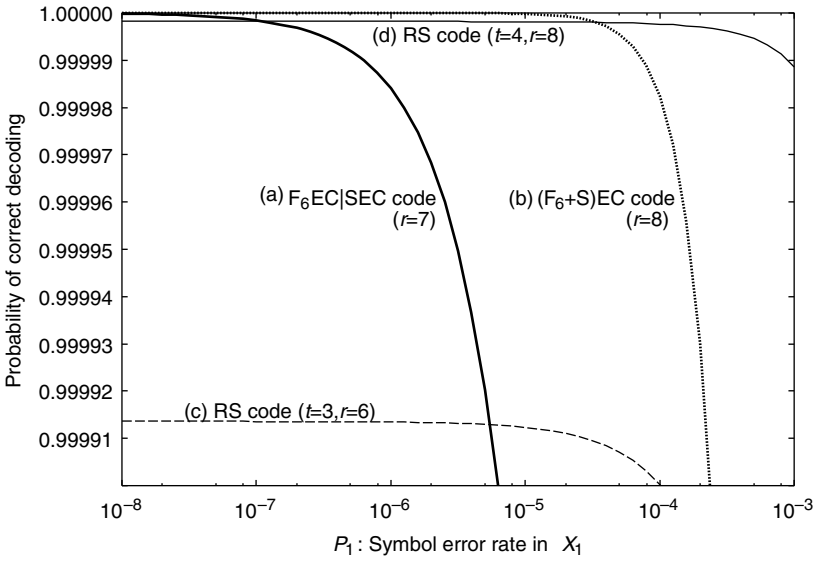


Figure 10.20 *Probability of correct decoding of the codes with code parameters $q = 67$, $n = 66$, $l = 6$, and $P_0 = 0.05$.*

$P_1 \leq 3.2 \times 10^{-5}$. Therefore the q-ary UEC codes are suitable for the unequal error model in which P_1 is much smaller than P_0, as is the case of the readout data in the holographic memories.

10.4.2 Application of UEP Scheme to Lossless Compressed Data

This subsection deals with an application of UEP coding scheme to error tolerance in lossless compressed text data [FUJI03].

Data compression is popularly applied to computer systems and communication systems in order to save storage area and communication bandwidth [BELL90]. Lossless compression, which can obtain the same decompressed data as the source data, is suitable to the text data. Ziv-Lempel coding is a typical class of lossless compression based on LZ77 coding [ZIV77] and LZW coding [WELC84], which is a modified version of LZ78 coding [ZIV78]. This class of coding uses an adaptive dictionary that encodes future segments of the source data via maximum-length copying from a dictionary containing the recent past output.

This subsection shows the influence of errors that occur in the text data compressed by Ziv-Lempel coding. From theoretical analysis and computer simulation we know that errors in the error-sensitive part of compressed data give more serious damage to the decompressed data than errors in the other parts. In LZW coding, the part of the data used to build the dictionary is sensitive to error, while in the LZ77 coding, the matched length part is sensitive to error. Therefore the UEP scheme, which protects the error-sensitive part of the compressed data more strongly than the other parts, is applied to the compressed data. Computer simulation tells us that the UEP scheme can recover from the errors in compressed data more effectively than a method using the existing burst error correcting Fire codes applied uniformly to the compressed data.

1. Lossless Text Data Compression

First, lossless compression algorithms of LZW coding and LZ77 coding are briefly introduced.

LZW Coding The LZW algorithm is organized by a translation table (i.e., a dictionary) that maps the strings of input characters onto the fixed-length words. The LZW string table has a prefix property where, for every string in the table, a corresponding prefix string is also in the table and is initialized to contain all single-character strings. Each string in the table is assigned a sequential index, called an *output code*, that represents the compressed strings. LZW coding adopts a ''greedy'' parsing algorithm, whereby the input string is examined character-serially in one pass, and the longest recognized string is parsed off each time. Each parsed input string extended by its next input character forms a new string added to the dictionary if the number of strings in the dictionary does not reach its limit. The following algorithm explains the compression process:

Step 1. Let the dictionary contain all single-character strings.

Step 2. Parse the longest string that matches a character string in the dictionary. Output the index of the string in $\lceil \log_2 m \rceil$ bits binary expression, where $\lceil x \rceil$ is the smallest integer not less than x and m is the maximum number of the strings the dictionary can store.

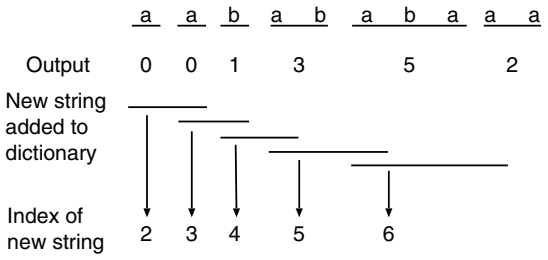


Figure 10.21 *Compression process of LZW coding for an input string "aabababaa."* Source: *[FUJI03]. © 2003 IEICE Japan.*

Step 3. If the number of strings in the dictionary does not reach its limit, add the parsed string extended by its next input character as a new string.

Step 4. Repeat steps 2 and 3 until the input string is exhausted.

Example 10.8 [FUJI03]

Let "aababababaaa" be the input string and $\{a, b\}$ be the input alphabet. Initially the dictionary contains two strings, "a" and "b." The first parsed string is "a" and its index "0" is the output. Then the string extended by its next input character forms a new string "aa" and is added to the dictionary. The index of the new string is "2." The compression is illustrated in Figure 10.21, and Table 10.8 lists the dictionary.

LZ77 Coding The LZ77 coding consists of a rule for parsing strings of symbols from a finite alphabet into substrings, or words, and a coding scheme that maps these substrings sequentially onto uniquely decodable codewords of fixed length. The LZ77 coding uses an n-symbol buffer. The buffer contains a part of the input string that starts at the $(n - L_s - 1)$-th symbol and ends after L_s symbols from the point. That is, the former $n - L_s$ symbols, which have already been encoded, are in the dictionary and the remaining L_s symbols, which are going to be encoded, are in the lookahead buffer. The input string is assumed to be preceded by $n - L_s$ zeros, the first symbol of the source alphabet. The LZ77 searches the dictionary to find the longest match with the beginning of the lookahead buffer. The matched string in the lookahead buffer and the following symbol are parsed and encoded into a fixed-length word that consists of three elements: the offset of the matched string from the lookahead buffer, the length of the match, and the last symbol of

TABLE 10.8 Dictionary of LZW Coding for Input String "aababababa"

Index	String
0	a
1	b
2	aa
3	ab
4	ba
5	aba
6	abaa

Source: [FUJI03]. © 2003 IEICE Japan.

the parsed string. Then the buffer is shifted so that the new coding point is located at the $(n - L_s)$-th symbol of the buffer.

The following algorithm shows the compression process:

Step 1. Let the former $n - L_s$ symbols of the buffer be $n - L_s$ copies of zero, the first symbol of the input alphabet, and let the remaining L_s symbols be the first L_s symbols of the input string.

Step 2. Search the dictionary to find the longest match with the beginning of the lookahead buffer. Parse the matched string in the lookahead buffer and the following symbol. Let the offset of the matched string and the length of the match be p_i and l_i, respectively. If the matched string cannot be found, let $p_i = 1,\ l_i = 0$.

Step 3. Output the fixed length word $C_i = C_{i,1}C_{i,2}C_{i,3}$, where $C_{i,1},\ C_{i,2}$, and $C_{i,3}$ are binary expressions of $p_i - 1,\ l_i$, and the last symbol of the parsed string, with lengths $\lceil \log_2(n - L_s) \rceil,\ \lceil \log_2 L_s \rceil$, and $\lceil \log_2 M \rceil$, respectively, where M is the cardinality of the source alphabet.

Step 4. Shift the buffer by $l_i + 1$ symbols, and put the following $l_i + 1$ symbols of the source string into the lookahead buffer.

Step 5. Repeat steps 2, 3, and 4 until the last source symbol is compressed.

Example 10.9 [FUJI03]

Let "ccabcaa" be the input string, let $\{a, b, c\}$ be the input alphabet, and let $n = 8,\ L_s = 4$. Initially the dictionary contains four copies of "a" and the lookahead buffer has "ccab," the first 4 symbols of the input string. Since any strings in the dictionary do not match with the content of the lookahead buffer, only "c," the first symbol of the lookahead buffer, is parsed and the output is "00c." The buffer is shifted by one symbol and the following source symbol "c" is put into the buffer. In this case the string, including the last symbol of the dictionary, matches with the content of the lookahead buffer and "ca" is parsed. Since the offset is 4 and the length of the match is 1, the output is "31a." The compression steps are illustrated in nonbinary expression in Figure 10.22.

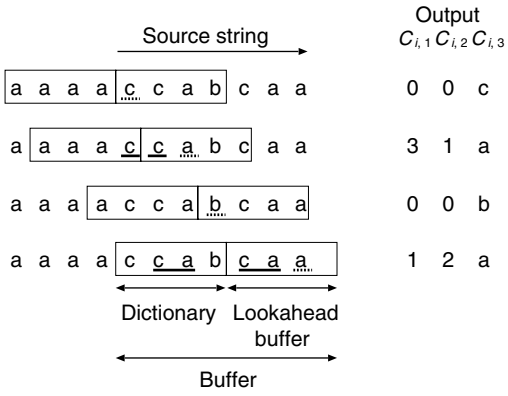


Figure 10.22 *Compression by LZ77 coding for the input string "ccabcab."* Source: *[FUJI03]. © 2003 IEICE Japan.*

2. UEP Scheme

(1) UEP in LZW Coding

LZW coding algorithm is organized by a translation table (i.e., a dictionary) that maps strings of input characters into fixed-length words. In the decompression process of LZW coding, the same dictionary used in the compression is constructed. Until the number of strings in the dictionary reaches the limit, a new string is added to the dictionary when a compressed word is decompressed. The error in the compressed word that is added to the dictionary corrupts the dictionary and can seriously damage the following decompression. The error in the compressed word that is not added to the dictionary, however, does not seriously damage the following decompression. Therefore the errors in the former part of the compressed data can more seriously damage the decompression than errors in the remaining latter part. The size of the former part is given by $(m - M)\lceil \log_2 m \rceil$ bits, where m is the size of the dictionary, M is the cardinality of the source alphabet, and $\lceil x \rceil$ is the smallest integer larger than or equal to x.

In order to verify the analysis above, the relation between an error location in the compressed data and the influence of the error is tested by computer simulation. Figure 10.23 shows the simulation result for a source file "paper1," the standard source file for compression [BELL90], having $m = 8,\ 192$ string dictionary and $M = 256$ symbols source alphabet. The effect of the error is obtained by comparing the data decompressed from the compressed data, which include 30-bit burst errors, with the original source data. The error value is expressed as the ratio of erroneous lines (i.e., lines in the decompressed data that differ from those in the source data) to total lines. Here we use a "return mark" to separate each line of a string of characters from each other. Note in the figure that errors in the first $(m - M)\lceil \log_2 m \rceil = 103,168$ bits in the compressed data have much more effect than errors in the remaining bits. This is why the former part of the compressed data should be protected more strongly than the latter part.

Encoding Method The scheme divides the compressed data into two parts: the former part with $(m - M)\lceil \log_2 m \rceil$ bits and the remaining part. So the error control codes are

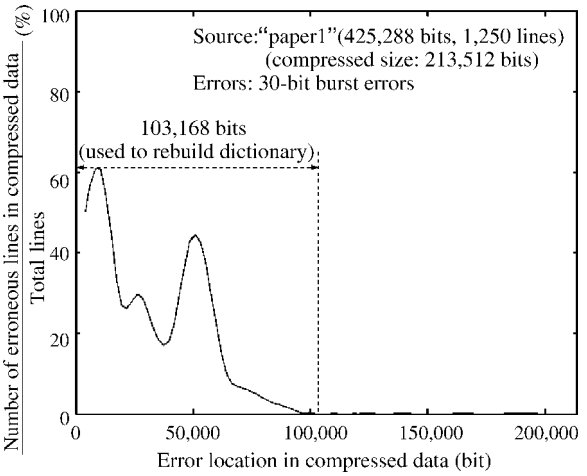


Figure 10.23 *Effect of errors in data compressed by LZW coding.* Source: *[FUJI03]. © 2003 IEICE Japan.*

applied to these parts separately. The code function in the former part should be stronger than that in the latter part. The following lists the encoding method.

Step 1. Compress the source data by ordinary LZW coding.

Step 2. Divide the compressed data into two parts. The former part has the first $(m - M)\lceil \log_2 m \rceil$ bits, and the latter part does the remaining bits in the compressed data.

Step 3. Apply l_1-bit and l_2-bit burst error correcting codes, where $l_1 > l_2$, to the former part and the latter part, respectively. Let the check-bit parts of the codes be C_1 and C_2, respectively.

Step 4. Output the compressed data by LZW coding, and the check-bit parts C_1 and C_2.

Evaluation Figure 10.24 shows the error recovery capability of the scheme for source file ''paper1''. The parameters of dictionary size, the cardinality of the source alphabets, and the injected burst error lengths are same as those in Figure 10.23. The former part of the compressed data employs a burst error correcting Fire code of $l_1 = 30$ bits and the latter part does a burst error correcting Fire code of $l_2 = 10$ bits. The proposed scheme requires 89 check bits for 213,512 bits of compressed data. Also injected are 30 bits of burst errors. For comparison, this figure includes the case of a 20-bit burst error correcting Fire code with 64-check bits, which is applied to the compressed data uniformly, and also includes a no error control code, denoted as ''without ECC.'' Note that the average ratio of the erroneous lines in the indicated UEP scheme is about 0.005% while that in the method using the conventional Fire code applied uniformly to the compressed data is about 3.83%.

Simulation results for other source files [BELL90] lead to a similar conclusion.

(2) UEP in LZ77 Coding

The compressed data by LZ77 coding have fixed-length words, each consisting of three elements—the offset of the matched string, the matched length, and the last symbol of the parsed string. Here we consider the influence of the errors that occur in

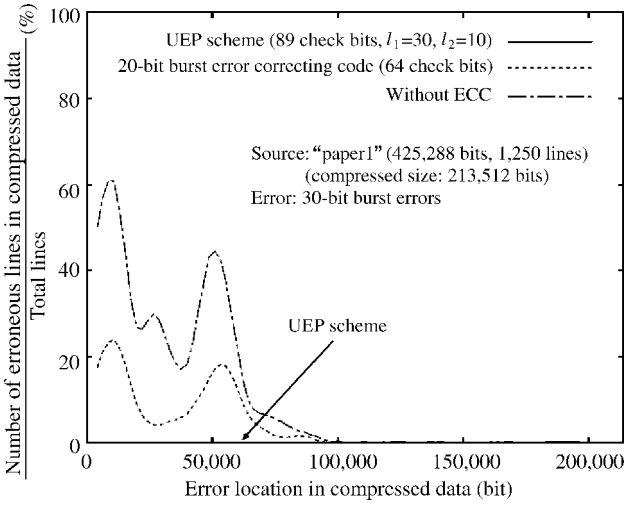


Figure 10.24 *Error recovery capability of the UEP scheme for "paper 1" LZW coding.* Source: *[FUJI03]. © 2003 IEICE Japan.*

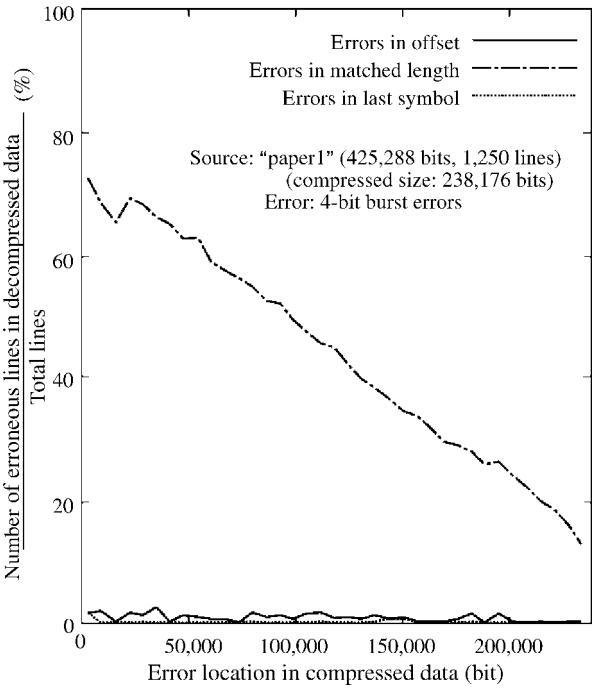


Figure 10.25 *Influence of errors in data compressed by LZ77 coding.* Source: *[FUJI03]. © 2003 IEICE Japan.*

these three elements. Errors in the matched length corrupt the shift operations of the buffer and affect the decoding of the following words. Errors in the offset and the last symbol corrupt the decoding only of the corresponding word; only the strings that depend on the erroneous words are decompressed incorrectly. So errors in the matched length cause more serious damage to the decompression than those in the offset and the last symbol.

This result is verified by computer simulation. Figure 10.25 shows the relation between the error location in the compressed data and the influence of the error to the decompressed data. The source file is "paper1," and the influence of the error is evaluated in the same manner as in Figure 10.23. The lengths of the offset, the matched length, and the last symbol in the compressed data are 12, 4, and 8 bits, respectively. In the three cases, 4-bit burst errors are injected in the offset, the matched length, and the last symbol of the compressed data. This says that errors in the matched length give more serious damage to the decompression than those in the offset and the last symbol. Also errors in the former part of the matched length corrupt larger amounts of compressed data than those in the latter part. Therefore the matched length in the compressed words, especially its former part of the compressed data, should be strongly protected from errors.

Encoding Method The encoding scheme divides the compressed data into three sets. The first set consists of the offsets and the last symbols. The second and the third sets are the former and latter halves of the collected matched lengths, respectively. These three sets are encoded by error control codes separately. Since errors in the second set give more

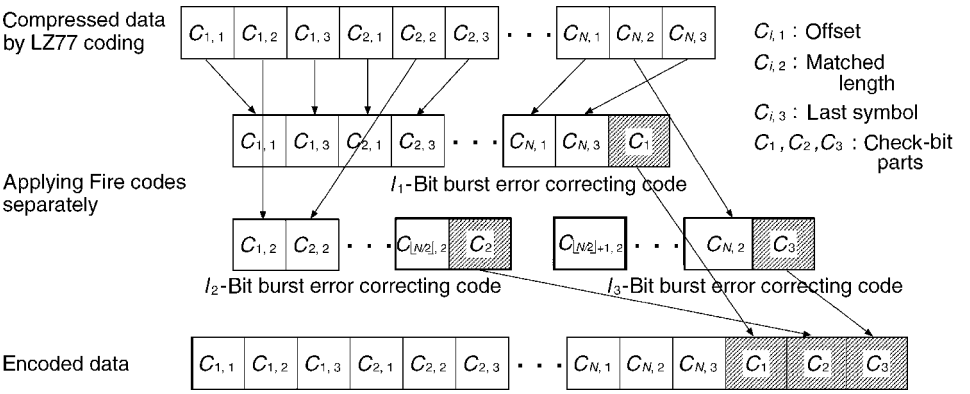


Figure 10.26 *Encoding process in LZ77 coding.* Source: *[FUJI03]. © 2003 IEICE Japan.*

serious damage to the decompressed data, the code in this set has stronger error control capability than the codes in the other sets. The following shows the encoding method, which is illustrated in Figure 10.26.

Step 1. Compress by LZ77 coding, and let the compressed data be $C_{1,1}\ C_{1,2}\ C_{1,3}\ C_{2,1}\ C_{2,2}\ C_{2,3}\ \cdots\ C_{N,1}\ C_{N,2}\ C_{N,3}$, where $C_{i,1}$, $C_{i,2}$, and $C_{i,3}$ are the offset of the matched string in the lookahead buffer, the matched length, and the last symbol of the parsed string, respectively, in the i-th word for $1 \leq i \leq N$.

Step 2. Divide the compressed data into three sets. The first set consists of the offsets and the last symbols, $C_{1,1}\ C_{1,3}\ C_{2,1}\ C_{2,3}\ \cdots\ C_{N,1}\ C_{N,3}$. The second and the third sets are the former and the latter half of the collected matched lengths, $C_{1,2}\ C_{2,2}\ \cdots\ C_{\lfloor N/2 \rfloor,2}$ and $C_{\lfloor N/2 \rfloor+1,2}\ C_{\lfloor N/2 \rfloor+2,2}\ \cdots\ C_{N,2}$, respectively.

Step 3. Apply l_1, l_2, and l_3 bits burst error correcting codes to theses three sets, respectively, and also let the check-bit parts of the codes be C_1, C_2, and C_3, respectively.

Step 4. Output the compressed data by LZ77 coding and the check-bit parts C_1, C_2, and C_3.

Evaluation Figure 10.27 shows relation between the error location in the compressed data and the error recovery capability of the scheme. In this case, C_1, C_2, and C_3 are determined by $l_1 = 16$ bits, $l_2 = 12$ bits, and $l_3 = 8$ bits burst error correcting Fire codes, respectively. Because the combined length of the offset and the last symbol is much larger than the total length of the matched length in this model, the burst error correction length l_1 takes rather large value. The source file is "paper1." The check-bit length is 105 bits for the 238,176 bits of compressed data and 48-bit burst errors are injected. The length of the offset, the matched length, and the length of the last symbol are same as those in Figure 10.25. Note in the comparison of Figure 10.27 the dramatic difference between the case of the 40-bit burst error correcting Fire code with 119 check bits applied to compressed data uniformly and the case of no error control code, denoted as "without ECC." Also note that the UEP scheme is the more powerful method to control errors than the existing burst error control code method applied uniformly to the compressed data. The simulations using other source files [BELL90] yielded similar results.

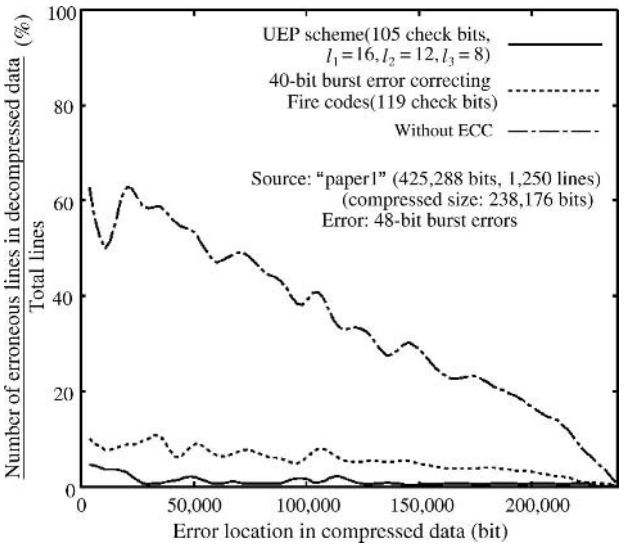


Figure 10.27 *Error recovery capability of the UEP scheme for "paper 1" in LZ77 coding.* Source: *[FUJI03].* *© 2003 IEICE Japan.*

EXERCISES

10.1 Design a (37, 21) F5EC|SEC code.

10.2 Design the decoding circuit of the (27, 22) F3EC|SEC code shown in Example 10.1.

10.3 Prove that the FbEC|SEC code is equivalent to the (Fb+S)ED|DED code.

10.4 Design a (36, 26) F5EC|SEC-DED code.

10.5 Prove Theorem 10.6.

10.6 Assume that the codewords contain separated fixed-bytes with equal level, meaning multiple parts (i.e., fixed bytes) not located adjacently and having equal error rate. Then explain how to design the fixed-byte error control UEC codes for this model.

10.7 Recall that a fixed b-bit byte error correcting plus single-bit error correcting ((Fb+S)EC) code corrects b-bit byte errors in X_0 and corrects single-bit errors in X_1, and in addition corrects byte errors in X_0 as well as single-bit errors in X_1 occurring simultaneously.

(a) Provide the necessary and sufficient conditions of the code.

(b) Prove that the maximal code length of an $(N,\ N-r)$ (Fb+S)EC code is shown as $N_{max} = 2^{r-b} + b - 1$.

(c) Prove that the FbEC|SEC-DED code includes the (Fb+S)EC code.

(d) Prove that the following $\mathbf{H}$ matrix reflects an (Fb+S)EC code satisfying the bounds on code length shown in **(b)**:

$$\boldsymbol{H} = [\mathbf{H}_0 \mid \mathbf{H}_1] = \left[\begin{array}{c|c:c} \mathbf{I}_b & \mathbf{O} & \\ \hline \mathbf{P} & \mathbf{Q} & \mathbf{I}_r \end{array}\right],$$

where

$$\mathbf{H}_0 = \left[\begin{array}{c} \mathbf{I}_b \\ \hline \mathbf{P} \end{array}\right], \quad \mathbf{H}_1 = \left[\begin{array}{c|c} \mathbf{O} & \\ \hline \mathbf{Q} & \mathbf{I}_r \end{array}\right],$$

O: zero matrix,

P: $(r-b) \times b$ matrix with distinct b columns each of weight 2 or more,

Q: matrix with distinct $(r-b)$-bit columns each of weight 2 or more except for those in **P**.

(e) Design a $(36, 26)$ (Fb+S)EC code.

(f) Find the decoding method of the code shown in **(d)**.

10.8 Recall the fixed b-bit byte error correcting | fixed b-bit byte plus single-bit error detecting (FbEC|(Fb+S)ED) code. In Figure 10.2 and in Definitions 10.1 and 10.2, the code is a 2-level UEC code with $N_0 = b$ and $N_1 = N - b$, where F_0 is a correction of b-bit byte error in X_0 and F_{01} is a detection of b-bit byte error in X_0 plus single-bit error in X_1.

(a) Provide the necessary and sufficient conditions of the code.

(b) Prove that the minimum number of check bit is $b + 1$.

(c) Design the code with minimum number of check bit.

(d) Find the decoding method based on the code designed above.

10.9 Prove that the (Fb+S)EC code is equivalent to the (Fb+S)ED|SEC code as well as to the FbEC|DED code.

10.10 Consider now the 3-level UEC code of the fixed b-bit byte error correcting | single-bit error correcting | single-bit error detecting (FbEC|SEC|SED) code whose codeword is divided into three fixed-bytes, X_0, X_1, and X_2, with lengths (in bits) $N_0(= b), N_1$, and N_2, respectively.

(a) Provide the necessary and sufficient conditions of the code.

(b) Prove that the maximum length of N_1 is equal to $2^r - 2^b - 1$, meaning $N_{1max} = 2^r - 2^b - 1$, where r is a check-bit length.

(c) Design the code with N_{1max} in a generalized form.

(d) Design the code with $b = 3$, $r = 5$, and $N_{1max} = 23$.

10.11 The 2-level UEC codes can correct single b-bit byte errors in low-reliability areas with L bytes, meaning $N_0 = L \times b$ bits, and correct single-bit errors in high-reliability areas with N_1 bits including check bits, which is called an SbEC|SEC code. Answer the following questions:

(a) Prove that the $(N = N_0 + N_1,\ N - R)$ SbEC|SEC codes have the following relation between code parameters:

$$\frac{N_0}{b} \cdot (2^b - 1) + N_1 \leq 2^R - 1.$$

(b) Prove that the following code shows the $(72, 64)$ S4EC|SEC code with $N_0 = 16$, $N_1 = 56$, $L = 4$, and $b = 4$:

$$\mathbf{H} = \left[\begin{array}{cccc|ccccc|cc} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{P} & \mathbf{P} & \mathbf{P} & |\mathbf{P}|_3 & & \mathbf{I} & \mathbf{O} \\ \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^3 & \mathbf{T}^4 \cdot \mathbf{P} & \mathbf{T}^5 \cdot \mathbf{P} & \mathbf{T}^6 \cdot \mathbf{P} & |\mathbf{T}^7 \cdot \mathbf{P}|_3 & & \mathbf{O} & \mathbf{I} \end{array}\right],$$

where

$\mathbf{T}$: 4×4 companion matrix defined by $\mathbf{g}(x) = x^4 + x + 1$,
$\mathbf{I}$: 4×4 identity matrix,
$\mathbf{O}$: 4×4 zero matrix,
$\mathbf{P}$: parity check matrix of (15,11) SEC code,

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

$|\mathbf{P}|_3$: first 3 columns in $\mathbf{P}$,
$|\mathbf{T}^7 \cdot \mathbf{P}|_3$: first 3 columns in $\mathbf{T}^7 \cdot \mathbf{P}$.

(c) Referring to the code design shown in **(b)**, design the generalized matrix form of the $(N,\ N - R)$ SbEC|SEC code using the maximal SbEC codes indicated in Subsection 5.1.4.

10.12 Prove that the B$_l$EC|SEC codes denoted by Theorem 10.12 satisfy the following relation in addition to those indicated in Theorem 10.10:

$$\begin{gathered} r > \min(c + m), \\ n_0 \leq \mathrm{LCM}(c, 2^m - 1), \\ 2l - 1 \leq c, \\ l \leq m, \end{gathered}$$

where $\mathrm{LCM}(x, y)$ means a least common multiple of x and y.

10.13 Design the 2-level l_1-bit burst error correcting and l_2-bit burst error correcting UEP codes, denoted as $(\mathrm{B}_{l_1}\mathrm{EC})_{n_0}$-$(\mathrm{B}_{l_2}\mathrm{EC})_{n_1}$ UEP codes, where $l_1 > l_2$, which can be designed by applying the interleaving method to the $(\mathrm{B}_l\mathrm{EC})_{n_0}$-$(\mathrm{SEC})_{n_1}$ codes. In your design of the codes, also use conversion matrix $\mathbf{P}$ mentioned in Subsection 10.3.2.

10.14 Design the 2-level l_1-bit burst error correcting and l_2-bit burst error correcting UEC codes (called $\mathrm{B}_{l_1}\mathrm{EC}|\mathrm{B}_{l_2}\mathrm{EC}$ code) where $l_1 > l_2$.

10.15 Prove Theorem 10.17.

10.16 Design an (18, 15) F_2EC|SEC code over GF(4).

10.17 Design a (34, 28) (F3+S)EC code over GF(5).

10.18 Recall the two-level burst error control q-ary UEC code, denoted as B_{l/n_0}EC|SEC code, capable of correcting q-ary l-symbol burst errors in X_0 with length n_0 symbols as well as correcting q-ary single-symbol errors in X_1 with length $n_1 = n - n_0$ symbols. Answer the following questions.

(a) Find the necessary and sufficient conditions of this code.

(b) Prove that a linear $(n, n-r)$ q-ary B_{l/n_0}EC|SEC code exists only if

$$n \le \frac{q^r - ((q-1)(n_0 - l) + q) \cdot q^{l-1}}{q-1} + n_0,$$

where n indicates the code length in symbols and r the check-symbol length.

(c) Show that the null space of the following matrix $\mathbf{H}$ over $GF(q)$ is an optimal linear q-ary B_{l/n_0}EC|SEC code that satisfies the bound on code length shown in **(b)**:

$$\mathbf{H} = \left[\begin{array}{c|c} \mathbf{H}_0 & \mathbf{H}_1 \\ \hline 0\,0\cdots 0 & 1\,1\cdots 1 \end{array}\right] = \left[\begin{array}{c|c} \mathbf{H}_{0,0} & \\ \hline \mathbf{H}_{0,1} & \\ \hline \vdots & \mathbf{H}_1 \\ \hline \mathbf{H}_{0,\,l-1} & \\ \hline 0\;0\cdots 0 & 1\;1\cdots 1 \end{array}\right] \begin{array}{l} \updownarrow 2 \\ \\ \\ \\ \updownarrow 1 \end{array} \Big\} 2l \quad ,$$

$$\longleftarrow n_0 \longrightarrow \longleftarrow n_1 = n - n_0 \longrightarrow$$

where

$$\mathbf{H}_{0,0} = \left[\begin{array}{c|c|c|c|c|c} & & & & & \\ 0\cdots 0 & \alpha_0^q & \cdots & 0\cdots 0 & \alpha_0^1 & 0\cdots 0 \;\; \alpha_0^0 \\ & & & & & \\ 0\cdots 0 & & & 0\cdots 0 & & 0\cdots 0 \\ & & & & & \end{array}\right] \begin{array}{l} \uparrow \\ 2 \\ \downarrow \end{array},$$

$$\longleftarrow l \longrightarrow \quad \longleftarrow l \longrightarrow \longleftarrow l \longrightarrow$$

$$\longleftarrow\!\!\longleftarrow n_0 \longrightarrow\!\!\longrightarrow$$

$$2l \le n_0 \le (q+1) \cdot l$$

$\mathbf{H}_{0,i}$: $2 \times n_0$ matrix that shifts i symbols cyclically leftward in $\mathbf{H}_{0,0}$, $1 \le i < l-1$,

$$\mathbf{H}_1 = \left[\begin{array}{cccccc} & & & & & 0 \\ & & & & & 0 \\ \alpha_1^{n_1-2} & \alpha_1^{n_1-3} & \cdots & \alpha_1^1 & \alpha_1^0 & \vdots \\ & & & & & 0 \end{array}\right] \begin{array}{l} \uparrow \\ 2l \\ \downarrow \end{array},$$

$$\longleftarrow n_1 = n - n_0 \longrightarrow$$

$$n_1 = n - n_0 \le q^{2l}$$

$\alpha_0^{j_0}$: q-ary coefficient vector of j_0-th power of α_0, where α_0 is a root of 2nd degree q-ary primitive polynomial, $0 \le j_0 \le q$,

$\alpha_1^{j_1}$: q-ary coefficient vector of j_1-th power of α_1, where α_1 is a root of $2l$-th degree q-ary primitive polynomial, $0 \le j_1 \le n_1 - 2$.

(d) Show that the following **H** matrix is an example of a (64, 57) $B_{3/12}EC|SEC$ code over $GF(3)$ with 7 check symbols, where α_0 is a root of 2nd-degree 3-ary primitive polynomial $\mathbf{g}_0(x) = x^2 + 2x + 2$, and α_1 is a root of 6th-degree 3-ary primitive polynomial $\mathbf{g}_1(x) = x^6 + 2x + 2$:

$$\mathbf{H} = \left[\begin{array}{l|l} 001001000001 & 12201220210200221121110110100100121000110000100000 10 \\ 002001001000 & 01210122012202102002211211101101001001210001100001 00 \\ 010010000010 & 12101220122021020022112111011010010012100011000010 00 \\ 020010010000 & 21012201220210200221121110110100100121000110000100 00 \\ 100100000100 & 10122012202102002211211101101001001210001100001000 00 \\ 200100100000 & 01220122021020022112111011010010012100011000010000 00 \\ 000000000000 & 11111111111111111111111111111111111111111111111111 11 \end{array}\right].$$

↓ shows the check symbol position

REFERENCES

[ASHL00] J. Ashley, M. P. Bermal, H. Coufal, H. Guenther, et al., "Holographic Data Storage," *IBM J. Res. Dev.*, 44 (May 2000): 341–366.

[BELL90] T. C. Bell, *et al., Text Compression*, Prentice-Hall (1990).

[BETZ98] G. A. Betzos, M. S. Porter, J. F. Hutton, and P. A. Mitkas, "Optical Storage Interactive Simulator (OASIS): An Interactive Tool for the Analysis of Page-Oriented Optical Memories," *Appl. Optics*, 37 (September 1998): 6115–6126.

[BURR97] G. W. Burr and B. Marcus, "Coding Tradeoffs for High-Density Holographic Data Storage," *Proc. SPIE*, 3802 (July 1999): 18–29.

[CHEN01] H. Chen, M. Kitakami, and E. Fujiwara, "Burst Error Recovery for VF Arithmetic Coding," *IEICE Trans. Fundamentals*, E84-A (April 2001): 1050–1063.

[CHO98] S. H. Cho, R. Kohno, and J. H. Park, "Non-Proper Variable-to-Fixed Length Arithmetic Coding," *IEICE Trans. Fundamentals*, E81-A (August 1998): 1739–1747.

[CHOU98] W. Chou and M. A. Neifeld, "Interleaving and Error Correction in Volume Holographic Memory Systems," *Appl. Optics*, 37 (October 1998): 6951–6968.

[FUJI95] E. Fujiwara and M. Kitakami, "A Class of Optimal Fixed-Byte Error Protection Codes for Computer Systems," *Dig. 25th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1995): 310–319.

[FUJI98] E. Fujiwara, T. Ritthongpitak, and M. Kitakami, "Optimal Two-Level Unequal Error Control Codes for Computer Systems," *IEEE Trans. Comput.*, 47 (December 1998): 1313–1325.

[FUJI02] E. Fujiwara, K. Namba and M. Kitakami, "Parallel Decoding for Burst Error Control Codes" (in Japanese), *IEICE Trans. Fundamentals*, J85-A (November 2002): 1284–1295. (Translated into English in *Electron. Commun. Japan, P. 3*, 87 [January 2004]: 38–48.)

[FUJI03] E. Fujiwara and M. Kitakami, ''Unequal Error Protection in Ziv-Lempel Coding,'' *IEICE Trans. Info. Syst.*, E86-D (December 2003): 2595–2600.

[GUPT75] S. N. Gupta, ''On UEP Burst Codes,'' *J. Math. Sci.*, 10, (1975): 21–27.

[HAYA00] T. Hayashi and E. Fujiwara, ''Bit and Byte Error Protection Codes with Two Protection Levels'' (in Japanese), *IEICE Trans. Fundamentals*, J83-A (February 2000): 196–207.

[IWAS97] A. Iwasaki and E. Fujiwara, ''SEC-DED Codes with Byte Error Protection Capabilities,'' (in Japanese), *Proc. IEICE Fundamental Convention*, A-6-4 (1997): 122.

[KANE03] H. Kaneko and E. Fujiwara, ''Optimal Two-Level q-Ary Unequal Error Control Codes,'' *Proc. 2003 IEEE Int. Symp. on Information Theory* (June 2003): 215.

[KING00] B. M. King and M. A. Neifeld, ''Sparse modulation coding for increased capacity in volume holographic storage,'' *Appl. Optics*, 39 (December 2000): 6681–6688.

[LIN83] S. Lin and D. J. Costello Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall (1983).

[LO96] J. C. Lo, M. Kitakami, and E. Fujiwara, ''Reliable Logic Circuits with Byte Error Control Codes—A Feasibility Study,'' *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems* (November 1996): 286–295.

[MASN67] B. Masnick and J. Wolf, ''On Linear Unequal Error Protection Codes,'' *IEEE Trans. Info. Theory*, IT-3 (October 1967): 600–607.

[MORE94] R. H. Morelos-Zaragoza and S. Lin, ''On a Class of Optimal Nonbinary Linear Unequal-Error-Protection Codes for Two Sets of Messages,'' *IEEE Trans. Info. Theory*, IT-40 (January 1994): 196–200.

[NAMB02] K. Namba and E. Fujiwara, ''Two-Level Unequal Error Protection Codes with Burst and Bit Error Correcting Capabilities,'' *IEICE Trans. Fundamentals*, E-85-A (June 2002): 1426–1430.

[NAMB03] K. Namba, ''Unequal Error Control Codes with Two-Level Burst and Bit Error Correcting Capabilities'' (in Japanese), *IEICE Trans. Fundamentals*, J-86-A, (May 2003): 578–586. This is translated in English in *Electronics and Communications in Japan Part II : Fundamental Electronics Science*, 87 (April 2004) : 21–29.

[PETE72] W. W. Peterson and E. J. Weldon Jr., *Error-Correcting Codes*, 2d ed., MIT Press (1972).

[REIG60] S. H. Reiger, ''Codes for the Correction of Clustered Errors,'' *IRE Trans. Info. Theory*, IT-6 (March 1960): 16–21.

[RITT95] T. Ritthongpitak and E. Fujiwara, ''A Class of Optimal Fixed-Byte Error Protection Codes,'' *Proc. 1995 IEEE Int. Symp. on Information Theory* (September 1995): 148.

[RITT96] T. Ritthongpitak, M. Kitakami, and E. Fujiwara, ''Optimal Two-Level Unequal Error Control Codes for Computer Systems,'' *Dig. 26th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1996): 190–199.

[TILB89] H. V. Tilborg and M. Blaum, ''On Error-Correcting Balanced Codes,'' *IEEE Trans. Info. Theory*, 35 (September 1989): 1091–1095.

[WELC84] T. A. Welch, ''A Technique for High-Performance Data Compression,'' *IEEE Compu.*, 17 (June 1984): 8–16.

[ZIV77] J. Ziv and A. Lempel, ''A Universal Algorithm for Sequential Data Compression,'' *IEEE Trans. Info. Theory*, IT-23 (May 1977): 337–343.

[ZIV78] J. Ziv and A. Lempel, ''Compression of Individual Sequences via Variable-Rate Coding,'' *IEEE Trans. Info. Theory*, IT-24(5) (September 1978): 530–536.

CONTENTS

11

Codes for Mass Memories

This chapter deals with the codes for mass memories such as for magnetic tapes, magnetic disks, and optical disks. Characteristic problems with these memories include burst errors, caused by defects and dust particles on the recording surfaces, and random errors caused by noise in the read / write heads. Burst error correcting codes such as Fire codes [FIRE59] have been used in these memories, and also Reed-Solomon byte error correcting / detecting codes [REED60] combined with interleaving methods or erasure correction methods have been applied in order to extend their error correction capabilities. Today the optical disk memories of CDs and DVDs depend on the powerful interleaved byte error correcting codes. Some examples are the cross-interleaved RS code (CIRC), the long-distance code (LDC), and the RS product code (RSPC). Also high-speed decoding is implemented by use of LSI circuits, especially for modem-day LDCs in optical erasable disks.

Holographic memories are being studied and developed as forthcoming ultra-large-capacity two-dimensional memories. In these memories a new type of error control coding, including a combination of UEC coding and modulation coding, has been proposed, as we saw in Chapter 10.

11.1 TAPE MEMORY CODES

Magnetic tapes are widely used in computer and audio / video systems. The half-inch, nine-track tape system has been especially prevalent, having evolved through the extensive use of tapes over many years. The nine-track system basically depends on an 8-bit byte of information and a parity bit. The checking for these nine bits in the vertical direction is called a *vertical redundancy check (VRC)*. Another parity bit is appended horizontally to each track at the end of the record. The checking in the horizontal direction is called a *longitudinal*

Code Design for Dependable Systems: Theory and Practical Applications, by Eiji Fujiwara

redundancy check (LRC). In the low-density recording tape, the VRC and LRC were sufficient. As the bit density increased, another redundancy check, called a *cyclic redundancy check (CRC)*, was replaced for the LRC. That is, the check bits of the CRC codes were appended for the LRC. So the double-redundancy check provided a single-track error correction that can recover from a total track failure within a block of data in the nine-parallel-track system [BROW70].

Because of increased bit densities and tape speeds, the new tape systems require more sophisticated error correction codes. Two coding schemes are presented in this chapter: an *optimal rectangular code (ORC)* for the 6,250 bits-per-inch (bpi) 9-track tape units [PATE74] and an *adaptive cross parity (AXP) code* for the higher density, 18-track tape units [PATE85].

The progress made in developing correction codes for half-inch tape systems has moved to its unique application in a *mass storage system (MSS)* that attains greater data storage at extremely low cost. For example, the IBM 3850 mass storage system adopts a unique storage architecture that uses a cartridge and a rotating head [PATE80]. The cartridge concept requires a different data format than the conventional parallel-track system and thus has abandoned the track error correction scheme. Instead, the MSS uses an interleaved byte error correction code to correct single-burst error section. The MSS coding scheme is also presented in this section.

Errors in magnetic tape recording are primarily caused by *defects* on the magnetic media or variations in head-media separation in the presence of *dust particles*. These errors often affect as many as 100 bits at a time, depending on the density of recording. Furthermore long errors can occur through loss of synchronization of the read clock, which renders subsequent data unreadable. As recording density is increased, another type of error plays a more important role: the *bit-shift phenomenon* in which magnetic flux transition is shifted from its normal position because of interference from neighboring flux transitions. The bit shift usually results in a double-bit error, where 01 is read in place of 10, and vice versa. Thus the data in magnetic tape systems are organized to facilitate recovery from mixed-mode errors in magnetic recording read-write processes involving random single-bit errors caused by *random noise*, multiple double-bit errors caused by *bit shifts*, and clusters of errors caused by *defects* and *dust particles* [PATE80].

As for other tape recording systems, powerful error correcting codes, such as *product code* using two Reed-Solomon codes, *cross-interleaved codes*, and a combination of the Reed-Solomon code and the *cyclic redundancy check (CRC) code*, have been extensively applied to recent digital audio and video systems, including the multitrack PCM tape-recording system, DAT (digital audio tape) units, and 8-mm video units [INOU78, TANA80, IMAI90]. The cross parity-check code is an attractive new class of tape memory codes. It belongs to the convolutional class of codes and has been demonstrated to be a maximum distance separable (MDS) convolutional code with a geometric regularity that gives both error and erasure decoding algorithms [FUJA89]. Two-dimensional product codes with reduced redundancy are provided for burst error correction [ROTH98].

11.1.1 Optimal Rectangular Codes (ORC)

The optimal rectangular code (ORC) [PATE74] is designed to correct any single-track errors and, when given erasure pointers, to correct any double-track errors in the tape. The codewords of the ORC have a rectangular format, and the check bits are located on two orthogonal sides of the rectangle. The data format for the ORC of 9-track tapes is illustrated in Figure 11.1. In the figure B_1 through B_7 denote the seven bytes of information

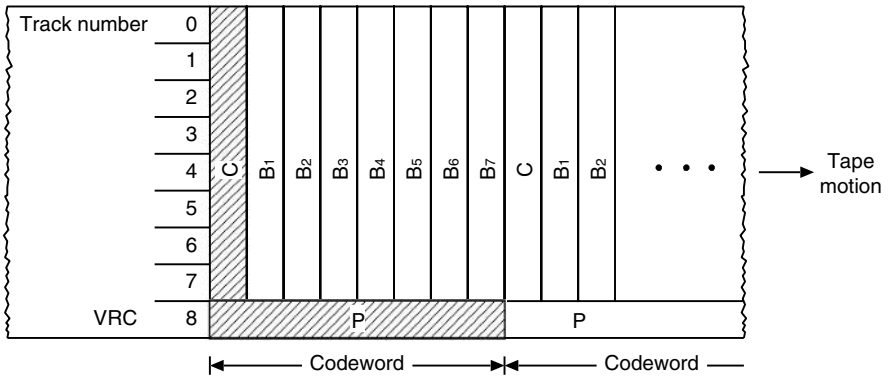


Figure 11.1 *Data format for ORC showing the information bytes as vertical columns.* Source: *[PATE74]. © 1974 by International Business Machines Corporation; republished by permission.*

in the standard 8-bit bytes. C denotes the check byte computed from the information bits. This data format clearly shows how the information bits are written as B_i.

The code corrects track errors as those in cluster of b bits along the tracks. Because we are interested in a natural description of the code, the track vectors of the codeword will be used as track bytes denoted by Z_i's as in Figure 11.2. Therefore we will first establish the error correcting capability of the code in the Z_i notation of Figure 11.2. Later, we will introduce a novel feature of the orthogonal symmetry of the ORC that gives the conversion of this Z_i notation to the B_i notation of Figure 11.1.

The ORC will be generated by using the companion matrix $\mathbf{T}$ (see Section 5.1.1) of the irreducible polynomial $\mathbf{g}(x)$ of degree 8 with binary coefficients g_i (i.e., $\mathbf{g}(x) = \sum_{i=0}^{8} g_i \cdot x^i$). Here $\mathbf{g}(x)$ is selected to obtain some specific advantages:

1. The exponent of the polynomial is small enough for high-speed error correction.
2. The polynomial $\mathbf{g}(x)$ is *self-reciprocal* (i.e., $\mathbf{g}(x) = x^8 \cdot \mathbf{g}(1/x)$) for *read backward facility* [PATE74].

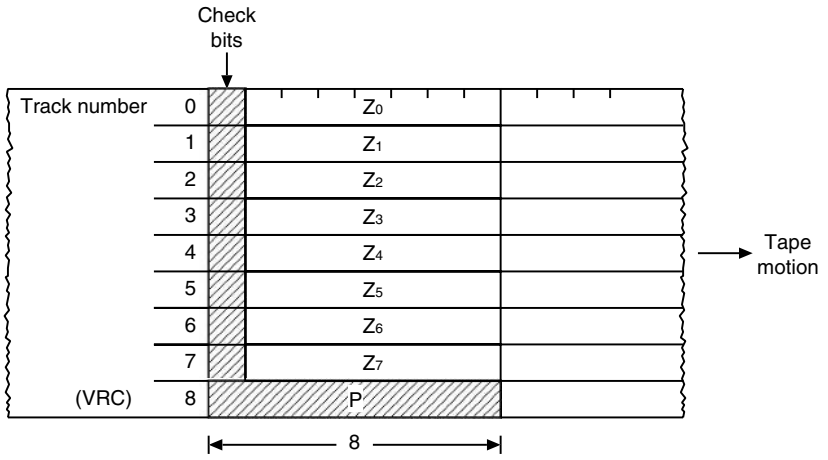


Figure 11.2 *Horizontal track bytes in ORC.* Source: *[PATE74]. © 1974 by International Business Machines Corporation; republished by permission.*

TABLE 11.1 Binary Irreducible Polynomials with Degree 8

Coefficients of Polynomial $\boldsymbol{g}(x)$									Exponent
g_0	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	e
1	0	0	0	1	1	1	0	1	255
1	0	1	1	1	0	1	1	1	85
1	1	1	1	1	0	0	1	1	51
1	0	1	1	0	1	0	0	1	255
1	1	0	1	1	1	1	0	1	85
1	1	1	1	0	0	1	1	1	255
1	0	0	1	0	1	0	1	1	255
1	1	1	0	1	0	1	1	1	17
1	0	1	1	0	0	1	0	1	255
1	1	0	0	0	1	0	1	1	85
1	0	1	1	0	0	0	1	1	255
1	0	0	0	1	1	0	1	1	51
1	0	0	1	1	1	1	1	1	85
1	0	1	0	1	1	1	1	1	255
1	1	1	0	0	0	0	1	1	255
1	0	0	1	1	1	0	0	1	17

Source: *[PATE74]. © 1974 by International Business Machines Corporation; republished by permission.*

Table 11.1 shows the list of binary irreducible polynomials with degree 8. Note among them that the self-reciprocal polynomial $\mathbf{g}(x) = 1 + x^3 + x^4 + x^5 + x^8$ having a minimum exponent $e = 17$ is chosen for the ORC application. The corresponding companion matrix $\mathbf{T}$ is

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

The parity-check matrix $\mathbf{H}$ of the ORC is written as

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \mathbf{I} \\ \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^3 & \mathbf{T}^4 & \mathbf{T}^5 & \mathbf{T}^6 & \mathbf{T}^7 & \mathbf{0} \end{bmatrix}. \tag{11.1}$$

That is, the parity-check matrix is equivalent to that of the $(72, 56)$ 2-redundant S8EC code (see Section 5.1.2). In this case the codeword is expressed as $W_Z = [Z_0\ Z_1\ Z_2\ Z_3\ Z_4\ Z_5\ Z_6\ Z_7\ P]$, and $\mathbf{I}$ is an 8×8 identity matrix and $\mathbf{0}$ is an 8×8 matrix with all zero elements. Therefore any correct codeword W_Z should satisfy the parity-check rule $W_Z \cdot \mathbf{H}^T = 0$, which means that it should satisfy the parity-check equations given by

$$\left[\sum_{i=0}^{7}{}^{\oplus} Z_i\right] \oplus P = 0 \tag{11.2}$$

and

$$\left[\sum_{i=0}^{7}{}^{\oplus} Z_i \cdot \mathbf{T}^i\right] = 0, \tag{11.3}$$

where $\sum^{\oplus}$ indicates modulo-2 sum and 0 is the null vector with all zero elements.

When the codeword is corrupted by either a single- or double-track error, the erroneous received word is denoted by $W'_Z = \left[Z'_0\ Z'_1\ Z'_2\ Z'_3\ Z'_4\ Z'_5\ Z'_6\ Z'_7\ P'\right]$. From the received word W'_Z, the syndromes S_1 and S_2 are computed as

$$S_1 = \left[\sum_{i=0}^{7}{}^{\oplus} Z'_i\right] \oplus P', \tag{11.4}$$

$$S_2 = \sum_{i=0}^{7}{}^{\oplus} Z'_i \cdot \mathbf{T}^i. \tag{11.5}$$

If there is no error, $S_1 = S_2 = 0$. Suppose that only the i-th track $(0 \le i \le 8)$ has errors and the error pattern is denoted by an eight-digit vector E (i.e., $E = Z_i \oplus Z'_i$). Then Eqs. (11.4) and (11.5) are rewritten as

$$S_1 = E, \tag{11.6}$$

$$S_2 = \begin{cases} E \cdot \mathbf{T}^i & \text{for } 0 \le i \le 7, \\ 0 & \text{for } j = 8. \end{cases} \tag{11.7}$$

If $S_2 \neq 0$, then the track position i is uniquely determined by the following fact:

$$S_2 \cdot \mathbf{T}^{-i} = S_1 = E. \tag{11.8}$$

The erroneous tracks are often identified externally by external pointers obtained upon detecting loss of signal in the read amplifiers, an excessive phase shift in clock, inadmissible recording patterns, and so on. We will see that if two erroneous tracks are identified by the external pointers, any two-track bytes in error in the ORC are correctable.

Let E_i and E_j denote the two error-pattern vectors representing errors in tracks i and j, respectively $(i < j)$. That is, the received bytes are error free except in tracks i and j, where $Z'_i = Z_i \oplus E_i$, and $Z'_j = Z_j \oplus E_j$ if $0 \le j \le 7$, or $P' = P \oplus E_j$ if $j = 8$. Then syndromes S_1 and S_2 of Eqs. (11.4) and (11.5) can be represented in the following way:

$$S_1 = E_i \oplus E_j,$$

$$S_2 = \begin{cases} E_i \cdot \mathbf{T}^i \oplus E_j \cdot \mathbf{T}^j & \text{for } 0 \le i \ne j \le 7, \\ E_i \cdot \mathbf{T}^i & \text{for } j = 8 \text{ or } j = i. \end{cases}$$

These equations uniquely determine the error patterns E_i and E_j as

$$E_i = S_1 \oplus E_j$$

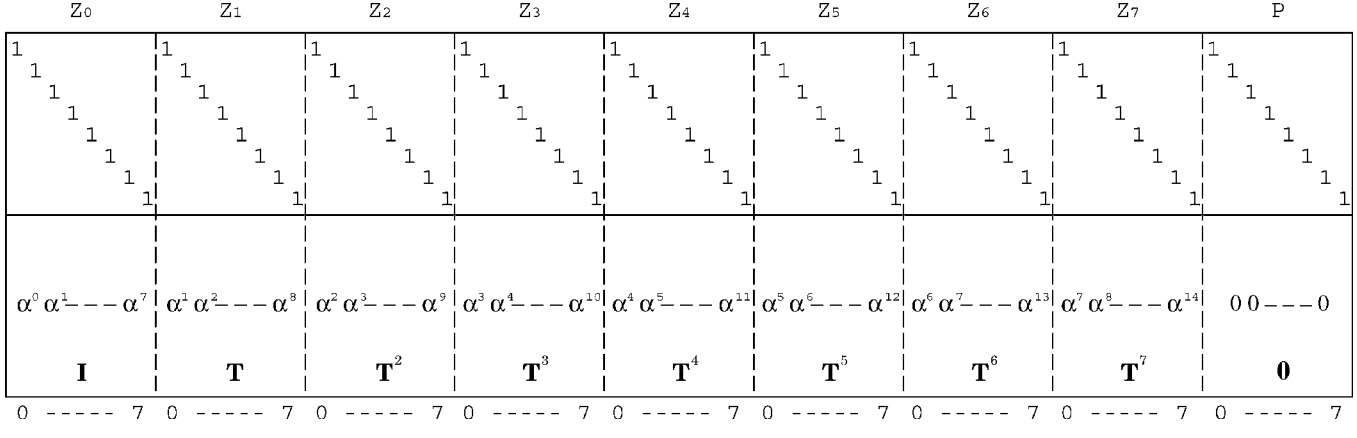


Figure 11.3 ***H*** *matrix for the ORC in Eq. (11.1).* Source: *[PATE74]. © 1974 by International Business Machines Corporation; republished by permission.*

and

$$E_j = \begin{cases} (\mathbf{I} \oplus \mathbf{T}^{j-i})^{-1} \cdot (S_1 \oplus S_2 \cdot \mathbf{T}^{-i}) & \text{for } 0 \le i \ne j \le 7, \\ S_1 \oplus S_2 \cdot \mathbf{T}^{-i} & \text{for } j = 8 \text{ or } j = i. \end{cases}$$

Here we have an interesting conversion of the parity-check matrix into the information-byte format. The alternate form allows direct computation of the check byte and syndrome. In addition the new format allows a fast implementation of the ORC without requiring a *buffer* for the encoding.

Figure 11.3 gives the binary version of the **H** matrix in Eq. (11.1). In the figure α^i is an element of $GF(2^8)$ expressed as an 8-digit column vector α^i of the binary coefficients of the polynomial x^i modulo $\mathbf{g}(x)$. First, consider the column of the **H** matrix corresponding to the bit $Z_i(j)$ of the track byte Z_i for all i and j such that $0 \le i \le 7$ and $0 \le j \le 7$. The lower half of this column is α^k, where $k = i + j$, which is the same for the column corresponding to $Z_j(i)$.

We call this property the *orthogonal symmetry* of the code. To complete the symmetry, let B_0 denote the check byte C. Then

$$Z_j(i) \equiv B_i(j) \qquad \text{for } 0 \le i \le 7 \text{ and } 0 \le j \le 7$$

Figure 11.4 shows the orthogonal symmetry and the powers of α that appear in the lower half columns of the **H** matrix. We can proceed to re-arrange the columns of the **H** matrix of Figure 11.3 to obtain another **H** matrix, $\mathbf{H}'$ in Figure 11.5, corresponding to a codeword W'_B in terms of the information bytes, written as $W'_B = [B_0\ B_1\ B_2\ B_3\ B_4\ B_5\ B_6\ B_7\ P]$.

Note that this re-arrangement does not alter the parity-checking rules. The orthogonal symmetry of the code has produced $\mathbf{T}^i$ in the lower half of the matrix $\mathbf{H}'$, corresponding to the information byte B_i, which is the same as that corresponding to the track byte Z_i. The upper half of $\mathbf{H}'$ is the conventional VRC, and it can be represented by a matrix $\mathbf{G}_i$, where $\mathbf{G}_i$ is an 8×8 all-zero matrix, except the row i, which is all ones.

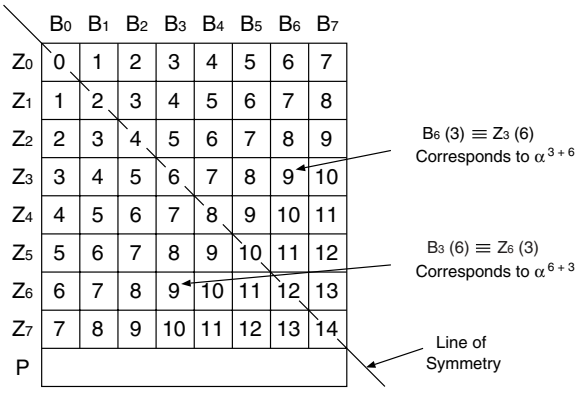


Figure 11.4 *Orthogonal symmetry and powers of α in the* ***H*** *matrix.* Source: *[PATE74]. © 1974 by International Business Machines Corporation; republished by permission.*

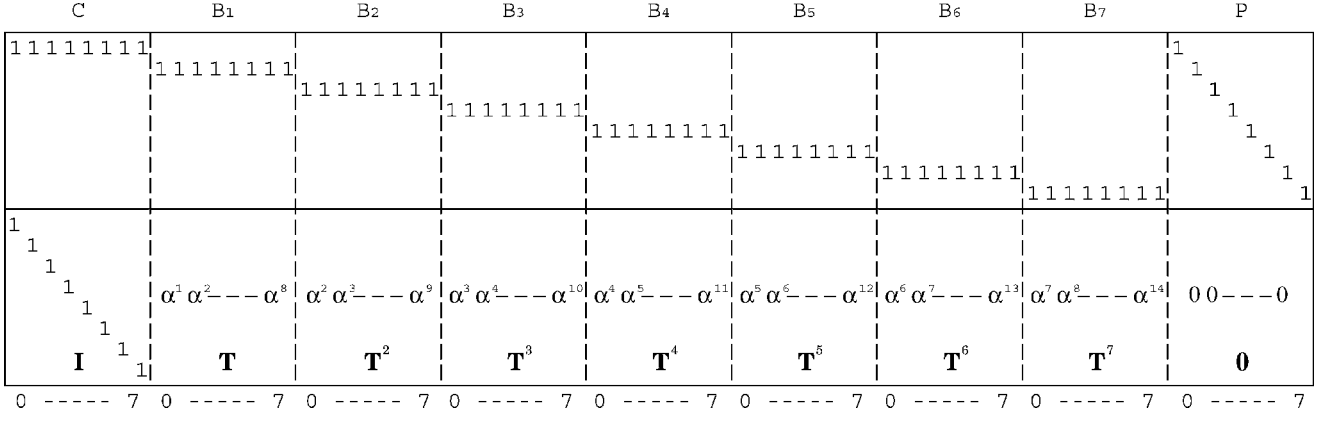


Figure 11.5 ***H*** *matrix for the ORC in Eq. (11.9).* Source: *[PATE74]. © 1974 by International Business Machines Corporation; republished by permission.*

Thus the new parity-check matrix $\mathbf{H}'$ is expressed as follows:

$$\mathbf{H}' = \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \mathbf{G}_2 & \mathbf{G}_3 & \mathbf{G}_4 & \mathbf{G}_5 & \mathbf{G}_6 & \mathbf{G}_7 & \mathbf{I} \\ \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & \mathbf{T}^3 & \mathbf{T}^4 & \mathbf{T}^5 & \mathbf{T}^6 & \mathbf{T}^7 & \mathbf{0} \end{bmatrix} \tag{11.9}$$

for the codeword $W'_{\mathrm{B}} = [B_0\ B_1\ B_2\ B_3\ B_4\ B_5\ B_6\ B_7\ P]$. The parity-checking equation is

$$W'_{\mathrm{B}} \cdot \mathbf{H}'^{T} = 0$$

Alternatively, the parity check can be computed from the information bytes as

$$C \equiv B_0 = \sum_{i=1}^{7} {}^{\oplus} B_i \cdot \mathbf{T}^i \tag{11.10}$$

and

$$P(i) = \sum_{j=0}^{7} {}^{\oplus} B_i(j), \qquad 0 \le i \le 7. \tag{11.11}$$

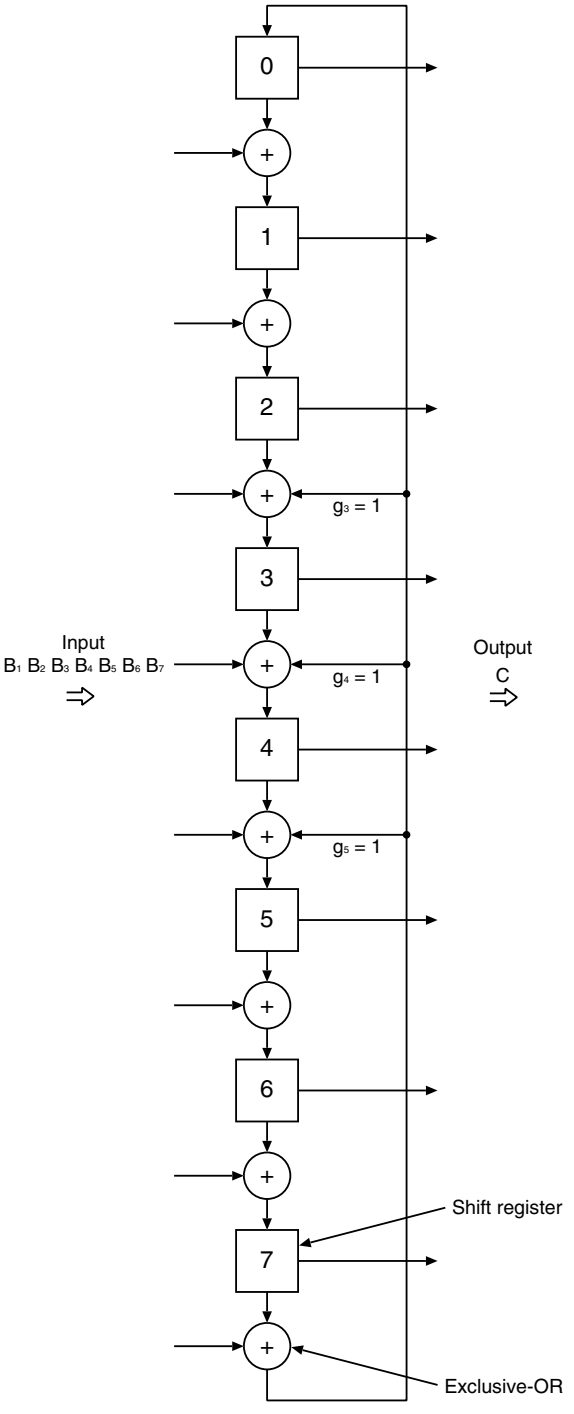
Equation (11.11) gives the conventional parity computation, which can be implemented by the usual exclusive-OR network. Equation (11.10) can be implemented by means of a linear feedback shift register (LFSR) connected for modulo $\mathbf{g}(x)$ operation (shown in Figure 11.6).

The shift operation in this LFSR corresponds to multiplying the content polynomial by x modulo $\mathbf{g}(x)$, which is equivalent to multiplying the content vector by the companion matrix $\mathbf{T}$. Input connections are such that the entering information bytes are premultiplied by $\mathbf{T}$. Initially this LFSR contains all zeros. The information byte $B_7,\ B_6,\ B_5,\ \ldots,\ B_2,\ B_1$ are successively shifted in parallel into the LFSR, in this order. Thus at the end of seven shifts the LFSR contains the vector $B_1 \cdot \mathbf{T} \oplus B_2 \cdot \mathbf{T}^2 \oplus \cdots \oplus B_7 \cdot \mathbf{T}^7$, which equals the check byte C.

Syndrome generation, especially generation of S_2, can be implemented by an LFSR similar to the one used in encoding. This can be accomplished by satisfying the equation

$$S_2 = C' \oplus \sum_{i=1}^{7} {}^{\oplus} B'_i \cdot \mathbf{T}^i,$$

where B'_i and C' denote the received vectors that are corrupted by error. The erroneous track can be uniquely pointed out only when Eq. (11.8) is satisfied. If we use a *backward-shifting register* [PATE74], its contents after i shifts should match $S_1 = E$. Thus, when a match occurs, the number of shifts determines the track position. Alternatively, a forward-shifting register, such as the LFSR shown in Figure 11.6, can be used for determination of the error track position i. Because $\mathbf{T}^{-i} = \mathbf{T}^{e-j}$, this requires a maximum of e shifts to determine the index i. In this case the polynomial $\mathbf{g}(x)$ with the lowest exponent e saves correction time.



$$g(x) = x^8 + x^5 + x^4 + x^3 + 1$$

Figure 11.6 *LFSR encoder (information bytes arrive from B_7, $B_6, \ldots, B_1$, in parallel to the shift (register).*
Source: *[PATE74]. © 1974 by International Business Machines Corporation; republished by permission.*

When tracks have erasures (instead of errors), double-track erasures can be corrected. (See [PATE74] or [LIN83].) The ORC is extended to become a code that can correct a track error together with a track erasure, or three track erasures. This can also be generalized to a $B(n, m)$ code of an $(n + 1) \times n$ array code in an $(n + 1)$-track tape, where there are check columns $B_0,\ B_1,\ \ldots,\ B_m,\ 0 \leq m \leq n - 1$, that can correct s track errors and t track erasures whenever $2s + t \leq m + 2$ [BLAU85]. So the ORC is a particular case of this family, $B(8, 0)$.

11.1.2 Adaptive Cross-Parity (AXP) Codes

The AXP code has been developed for a new high-density, 18-track tape storage subsystem [PATE85]. In this coding scheme the 18 tracks are divided into two sets of 9 tracks, with each set consisting of 7 data tracks and 2 check tracks. The proportion of check tracks is thus the same as that of the 9-track scheme. However, through adaptive use of the checks in the two sets, the new scheme corrects up to 3 erased tracks in any one set of 9 tracks and up to 4 erased tracks in two sets together. The coding structure, however, is simple, for it avoids the complex computations of Galois fields. It is based on vertical and cross-parity checks.

Figure 11.7 shows the data format for 18 tracks grouped into two sets. The set A consists of 9 parallel tracks, and the set B consists of the remaining 9 parallel tracks. In Figure 11.7 the two sets are shown side by side with a symmetrically ordered arrangement of the tracks.

Let $A_m(t)$ and $B_m(t)$ denote the m-th bit in the track t of sets A and B, respectively. The track number t takes on values from 0 to 8 in each set. The bit position m takes on values from 0 to M. Tracks labeled 0 and 8 in each set are check tracks.

Each check bit in track 0 of set A provides a cross-parity check along the diagonal with positive slope, involving bits from both sets, as seen in Figure 11.7. The m-th cross-parity check of set A is given by the encoding equation

$$A_m(0) = \sum_{t=1}^{7}{}^{\oplus} A_{m-t}(t) \oplus \sum_{t=0}^{7}{}^{\oplus} B_{m+t-15}(t), \tag{11.12}$$

where $\sum^{\oplus}$ indicates module-2 sum.

Each check bit in track 0 of set B provides a cross-parity check along the diagonal with the negative slope, involving bits from both sets, as seen in Figure 11.7. The m-th cross-parity check of set B is given by the encoding equation

$$B_m(0) = \sum_{t=1}^{7}{}^{\oplus} B_{m-t}(t) \oplus \sum_{t=0}^{7}{}^{\oplus} A_{m+t-15}(t). \tag{11.13}$$

Equations (11.12) and (11.13) can be rewritten as follows:

$$\sum_{t=0}^{7}{}^{\oplus} [A_{m-t}(t) \oplus B_{m+t-15}(t)] = 0, \tag{11.14}$$

$$\sum_{t=0}^{7}{}^{\oplus} [B_{m-t}(t) \oplus A_{m+t-15}(t)] = 0. \tag{11.15}$$

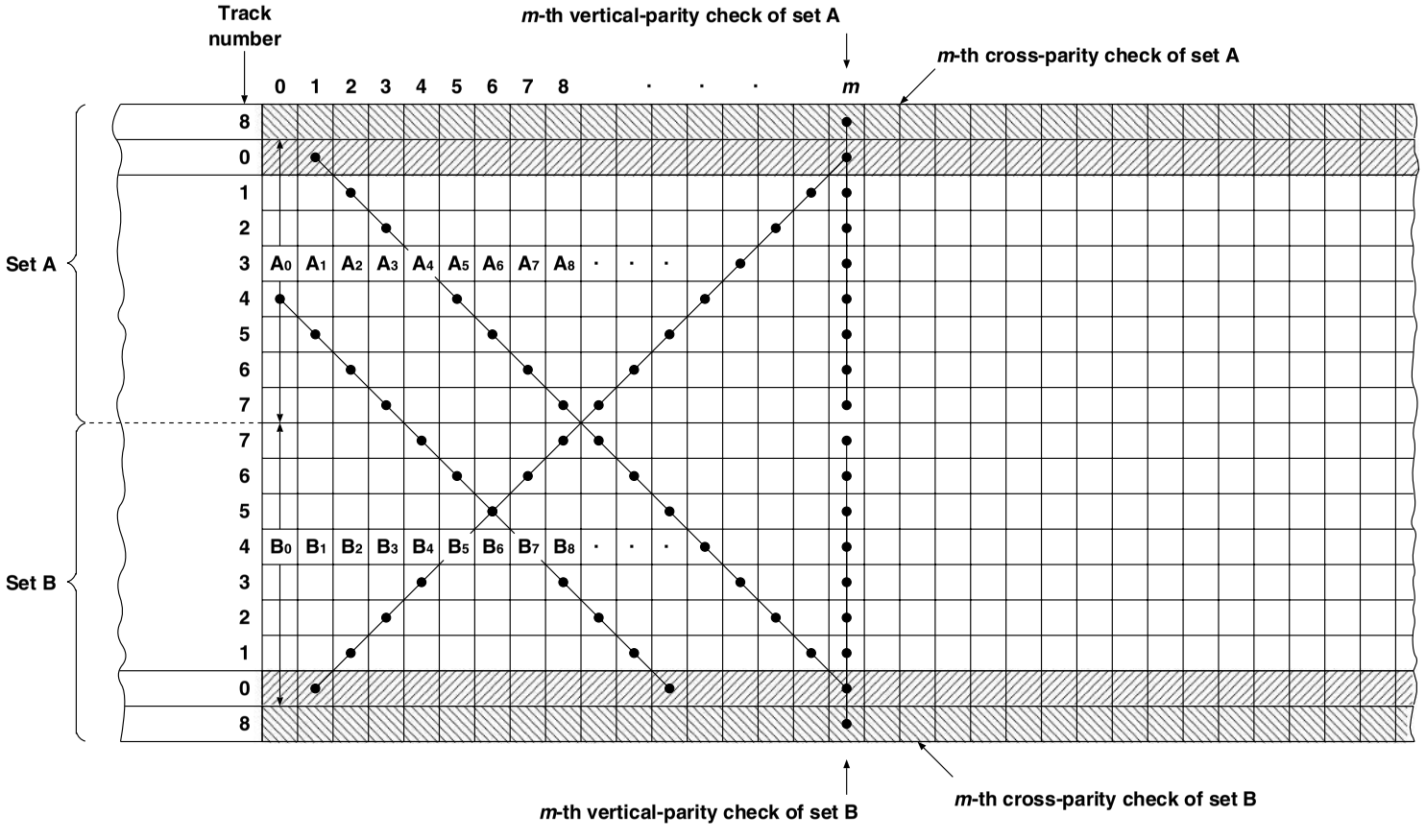


Figure 11.7 *Data format of 18 tracks grouped into two sets.* Source: *[PATE85]. © 1985 by International Business Machines Corporation; republished by permission.*

Each check bit in the track 8 of set A is a vertical parity over the bits of same position number m in set A. The m-th vertical-parity bit of set A is given by the equation

$$A_m(8) = \sum_{t=0}^{7} {}^{\oplus} A_m(t). \tag{11.16}$$

Similarly the m-th vertical-parity bit of set B is given by the equation

$$B_m(8) = \sum_{t=0}^{7} {}^{\oplus} B_m(t). \tag{11.17}$$

The encoding equations (11.12), (11.13), (11.16), and (11.17) are simple parity equations. These can be implemented by means of exclusive-OR circuits receiving inputs of appropriate data bit values.

Let $A'_m(t)$ and $B'_m(t)$ denote the bit values corresponding to $A_m(t)$ and $B_m(t)$, respectively, as they are read from the tape. These bits may be corrupted by errors. A nonzero syndrome indicates the presence of an error.

From Eqs. (11.14) through (11.17) the syndrome can be computed as follows:

$$Sd^a_m = \sum_{t=0}^{7} {}^{\oplus} \left[A'_{m-t}(t) \oplus B'_{m+t-15}(t)\right],$$

$$Sd^b_m = \sum_{t=0}^{7} {}^{\oplus} \left[B'_{m-t}(t) \oplus A'_{m+t-15}(t)\right],$$

$$Sv^a_m = \sum_{t=0}^{8} {}^{\oplus} A'_m(t),$$

$$Sv^b_m = \sum_{t=0}^{8} {}^{\oplus} B'_m(t).$$

Here Sd^a_m is the m-th cross-parity syndrome of set A, Sd^b_m is the m-th cross-parity syndrome of set B; Sv^a_m is the m-th vertical syndrome for set A, and Sv^b_m is the m-th vertical syndrome for set B.

The modulo-2 difference between the read $A'_m(t)$ and the written $A_m(t)$ is called the error pattern $E^a_m(t)$ in the m-th position of track t in set A. So for set A we have

$$E^a_m(t) = A'_m(t) \oplus A_m(t).$$

Similarly for set B,

$$E^b_m(t) = B'_m(t) \oplus B_m(t).$$

From the relations above we have the following syndromes expressed by the error patterns:

$$Sd_m^a = \sum_{t=0}^{7} \oplus \left[E_{m-t}^a(t) \oplus E_{m+t-15}^b(t)\right],$$

$$Sd_m^b = \sum_{t=0}^{7} \oplus \left[E_{m-t}^b(t) \oplus E_{m+t-15}^a(t)\right],$$

$$Sv_m^a = \sum_{t=0}^{8} \oplus E_m^a(t),$$

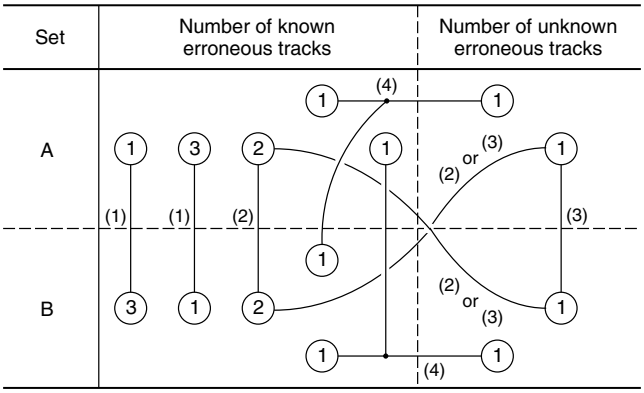
$$Sv_m^b = \sum_{t=0}^{8} \oplus E_m^b(t).$$

The erroneous track can be identified by the external pointers mentioned before. In the absence of the pointers, the erroneous track is identified by processing the syndromes. According to [PATE85], any one of the following combinations of track errors can be corrected by the AXP code:

(1) Up to three known erroneous tracks in one set and up to one known erroneous track in the other set.
(2) Up to two known erroneous tracks in one set and up to one unknown or two known erroneous tracks in the other set.
(3) Up to one unknown erroneous track in one set and up to one unknown or two known erroneous tracks in the other set.
(4) Up to one known and one unknown erroneous tracks in one set and up to one known erroneous track in the other set.

Figure 11.8 shows the combinations of track errors correctable by the AXP code. Using the error correction for the basic case described next, the reader should attempt to demonstrate the correction procedures for other cases.

We consider a case with three known erroneous tracks in set A. The three tracks are correctable if set B is error free or has only one known erroneous track. The erroneous



Number in each circle shows the maximum number of correctable tracks.

Figure 11.8 *Combinations of track errors correctable by the AXP code.*

tracks are indicated by track-error pointers i, j, and k in set A and l in set B. For convenience, assume that $i < j < k$. Since set B has only one known erroneous track, the vertical parity-check syndrome Sv^b_m yields the error patterns for this track:

$$Sv^b_m = E^b_m(l). \tag{11.18}$$

Let us assume that all errors have been corrected up to byte $m-1$ and that the syndrome equations have been adjusted for all corrected error patterns. Then, as shown in Figure 11.9, the error patterns for the m-th position of track i, j, and k of set A can be determined from the syndromes Sd^a_{m+i}, Sd^b_{m+15-k} and Sv^a_m as follows:

$$Sd^a_{m+i} = E^a_m(i), \tag{11.19}$$

$$Sd^b_{m+15-k} = \begin{cases} E^a_m(k) \oplus E^b_{m+15-l-k}(l) & \text{if } l < 8, \\ E^a_m(k) & \text{if } l = 8 \text{ or set B is error free,} \end{cases} \tag{11.20}$$

$$Sv^a_m = E^a_m(i) \oplus E^a_m(j) \oplus E^a_m(k). \tag{11.21}$$

From Eq. (11.18) we have

$$Sv^b_{m+15-l-k} = E^b_{m+15-l-k}(l). \tag{11.22}$$

Equations (11.19) through (11.22) yield the following error patterns:

$$\begin{aligned} E^a_m(i) &= Sd^a_{m+i}, \\ E^a_m(k) &= \begin{cases} Sd^b_{m+15-k} \oplus Sv^b_{m+15-l-k} & \text{if } l < 8, \\ Sd^b_{m+15-k} & \text{if } l = 8 \text{ or set B is error free,} \end{cases} \\ E^a_m(j) &= Sv^a_m \oplus E^a_m(i) \oplus E^a_m(k). \end{aligned}$$

The m-th bits in tracks i, j, and k are corrected by these error patterns as follows:

$$\begin{aligned} A_m(i) &= A'_m(i) \oplus E^a_m(i), \\ A_m(j) &= A'_m(j) \oplus E^a_m(j), \\ A_m(k) &= A'_m(k) \oplus E^a_m(k). \end{aligned}$$

Before proceeding to the correction of the next position, we must modify the syndromes affected by these corrections. The modification is shown by an arrow pointing from the previous value of a syndrome to its new value:

$$\begin{aligned} Sd^a_{m+i} &\leftarrow Sd^a_{m+i} \oplus E^a_m(i), \\ Sd^a_{m+j} &\leftarrow Sd^a_{m+j} \oplus E^a_m(j) \quad \text{if } j < 8, \\ Sd^a_{m+k} &\leftarrow Sd^a_{m+k} \oplus E^a_m(k), \\ Sd^b_{m+15-i} &\leftarrow Sd^b_{m+15-i} \oplus E^a_m(i), \\ Sd^b_{m+15-j} &\leftarrow Sd^b_{m+15-j} \oplus E^a_m(j) \quad \text{if } j < 8, \\ Sd^b_{m+15-k} &\leftarrow Sd^b_{m+15-k} \oplus E^a_m(k). \end{aligned}$$

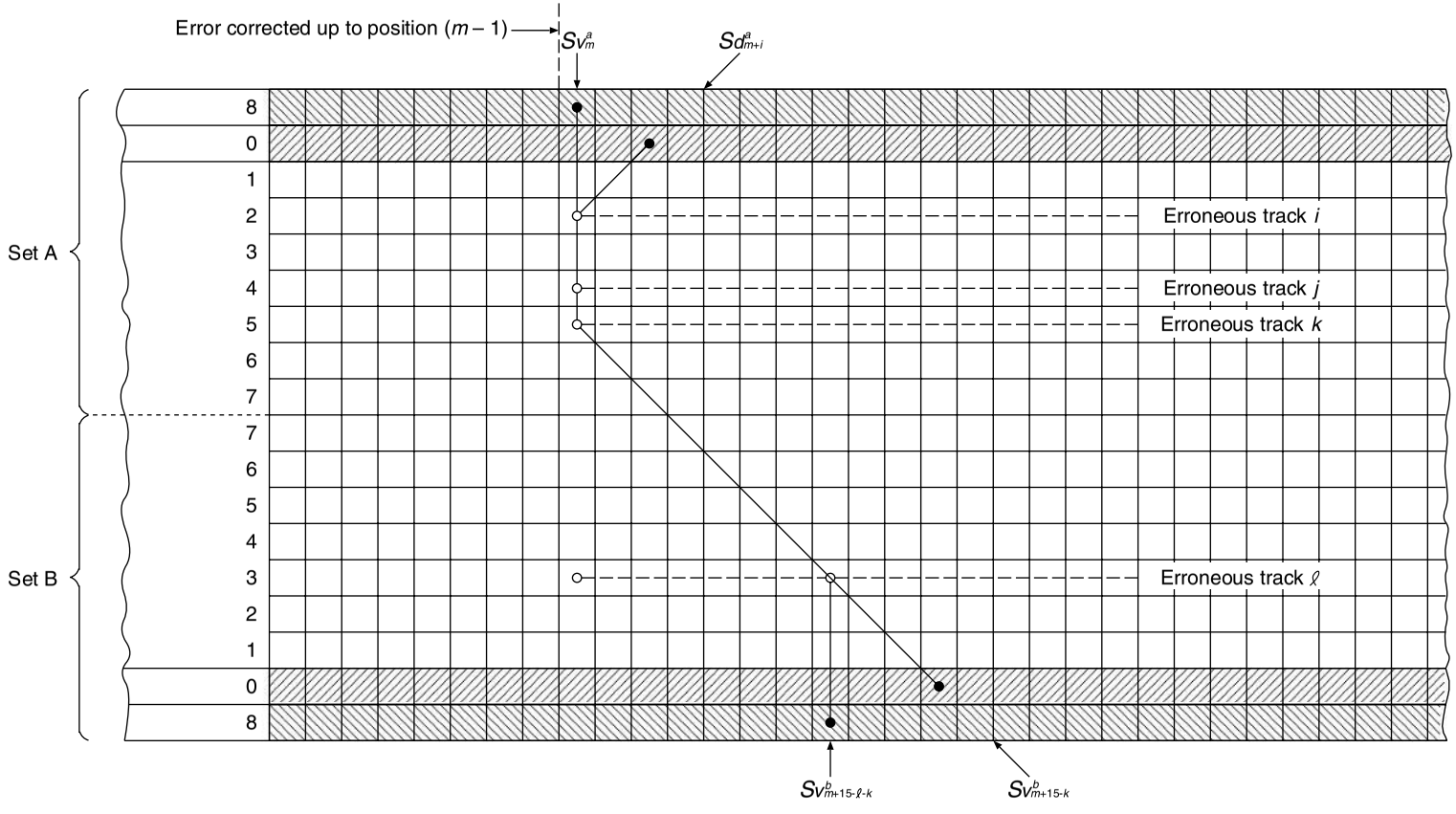


Figure 11.9 *Example of three-track error correction in set A.* Source: *[PATE85]. © 1985 by International Business Machines Corporation; republished by permission.*

Now the decoding procedure can be applied to the next bit position by changing the value of m by 1.

Next we consider determination of the track error pointers. Errors confined to only one unknown track in set A can be detected and corrected if set B has at most one unknown or two known erroneous tracks. It is assumed that errors in all tracks of set B have been corrected up to bit position $m-1$ and that the syndrome values have been adjusted for all corrected error patterns. When all tracks in set A are error free, the parity-check syndromes Sv_m^a and Sd_m^a are equal to zero for $0 < i < 7$. When any of these syndromes are found to be nonzero, it is indicated that an error is present in at least one of the tracks in the neighborhood. Assuming that only one erroneous track is affecting the syndromes, the index of the erroneous track can be determined by examining syndromes Sd_{m+7}^a and Sv_m^a as the bit position value m progresses.

Let m_1 and m_2 denote the lowest values of bit positions such that

$$\begin{aligned} Sd_{m+i}^a &\neq 0 \qquad \text{for } m = m_1, i = 7, \\ Sv_m^a &\neq 0 \qquad \text{for } m = m_2. \end{aligned}$$

Then track q is in error at bit position m_2 and the track index q is given by

$$q = \begin{cases} 7 - (m_2 - m_1) & \text{if } m_2 \geq m_1, \\ 8 & \text{otherwise.} \end{cases}$$

Figure 11.10 shows the case where m_2 is greater than or equal to m_1. Note that if the resulting value q in this case is smaller than zero, then the syndromes are affected by two or more unknown erroneous tracks and the errors are uncorrectable. The error can be easily corrected by using the syndrome Sv_m^a as follows:

$$\begin{aligned} E_{m_2}(q) &= Sv_{m_2}^a, \\ A_{m_2}(q) &= A'_{m_2}(q) \oplus E_{m_2}(q). \end{aligned}$$

Also syndromes should be adjusted as

$$\begin{aligned} Sv_{m_2}^a &\leftarrow Sv_{m_2}^a \oplus E_{m_2}(q), \\ Sd_{m_1+7}^a &\leftarrow Sd_{m_1+7}^a \oplus E_{m_2}(q). \end{aligned}$$

Since the coding rules possess a built-in mirror-image symmetry around set A and set B, the encoding and decoding equations for set B obviously can be obtained from those of set A by substitution of the corresponding variables.

11.1.3 Interleaved RS S*b*EC Codes for Mass Storage System (MSS)

The mass storage system (MSS) of the IBM 3850 system [PATE80], for example, stores digital data on flexible magnetic tape media; however, it is different in many respects from the conventional multitrack tape machines. The IBM system consists of an array of data cartridges about 1.9 in (4.8 cm) in diameter and 3.5 in (8.9 cm) long, each with a capacity of 50.4 million bytes (byte = 8-bit) of data. Each cartridge contains a spool of magnetic tape of 2.7 in (6.9 cm) wide and 64 ft (195 m) long. Up to 4,720 cartridges are stored in

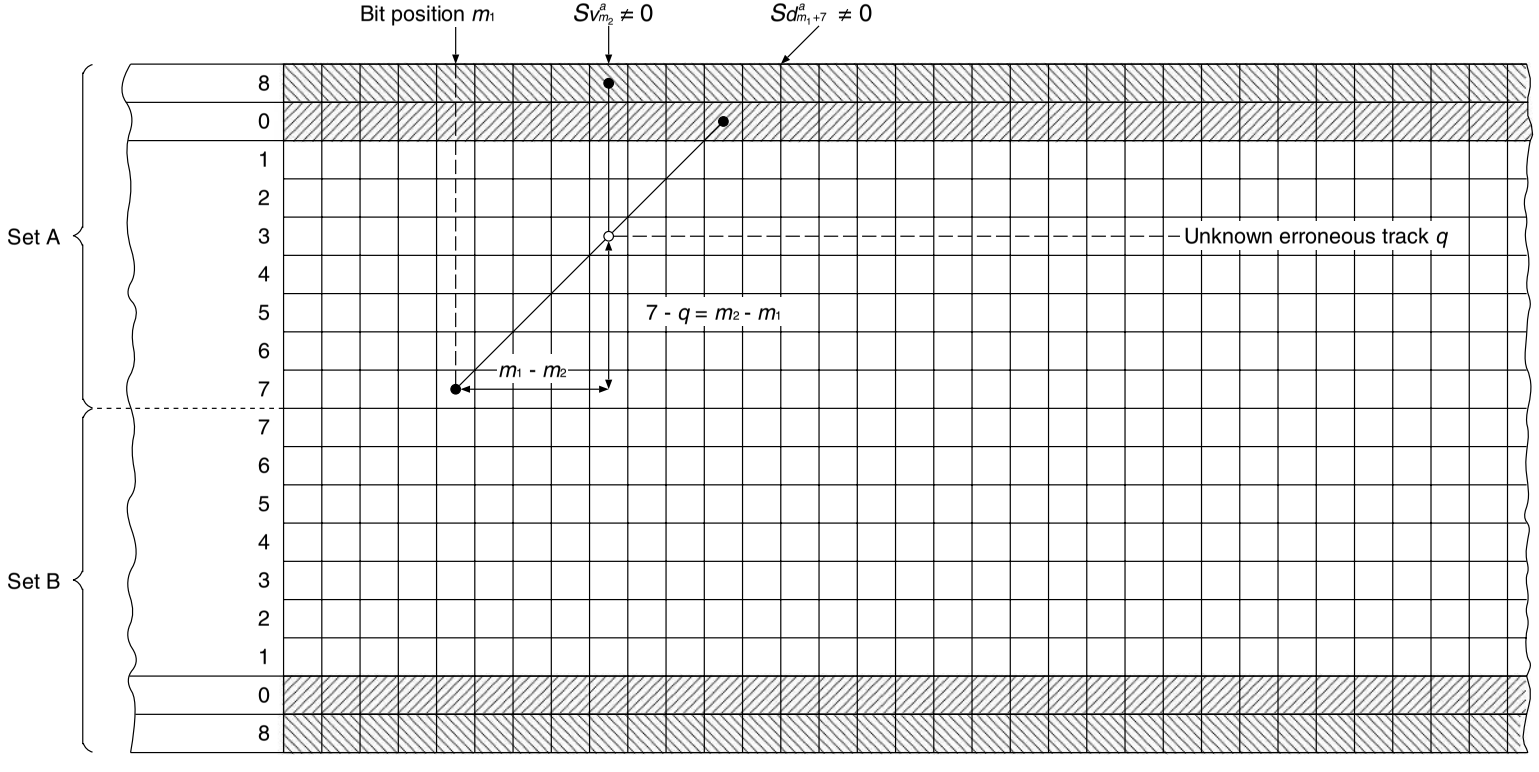


Figure 11.10 *Generation of pointer to the first erroneous track when $q \neq 8$.* Source: *[PATE85]. © 1985 by International Business Machines Corporation; republished by permission.*

hexagonal compartments in a honeycomblike apparatus that includes mechanisms for fetching cartridges from the compartments, for reading and writing of data on them, and for the replacement of cartridges in the compartments. Unlike the conventional fixed head of the multitrack tape machines, this system uses a rotary read-write head. Recent mass storage systems with peta-byte order capacity can accept commercial magnetic tapes, commercial video cassette tapes [ITAO85], or optical disks [YAMA91] as data cartridges, and reduce the system cost.

In the IBM3850 MSS systems, data are recorded in short slanted stripes across the tape. The tape follows a helical path around a read-write mandrel and is stepped in position from one slanted stripe to the next over a circular slit. The rotary read-write heads revolve with the mandrel. The stripe is divided into 20 segments, each of which consists of 13 data sections followed by 2 check sections. Each section consists of 16 bytes (a total of 128 bits). This arrangement is shown in Figure 11.11. Figure 11.12 shows the rectangular array of each segment, in which 15 bytes in each column form a codeword from a $(15, 13)$ single-symbol error correcting code with symbols from the Galois field $GF(2^8)$. The codewords are interleaved in the data format of 15 sections in a segment.

All detected errors, such as errors in a ZM (zero modulation) decoding algorithm, which checks for errors through stringent runlength and dc charge constraints, and odd-parity checked errors at the end of each section, are reported to the decoder of the error correction code for error recovery. When a defect or dust particle affects up to two full sections (i.e., 32 data bytes), the resultant error is recoverable by correcting the corresponding two bytes in each of the 16 codewords.

The basic structure of the codeword W is designated as

$$W = [B_0\ B_1\ B_2\ \cdots\ B_{14}].$$

In this codeword, B_0 and B_1 are the check bytes, and the remaining 13 bytes are the data bytes. Consider the Galois field $GF(2^8)$, which is constructed based on the primitive polynomial

$$\mathbf{p}(x) = x^8 + x^5 + x^3 + x + 1. \tag{11.23}$$

Let α be a primitive element in $GF(2^8)$ such that

$$\mathbf{p}(\alpha) = \alpha^8 + \alpha^5 + \alpha^3 + \alpha + 1 = 0.$$

Let $\beta = \alpha^{\lambda}$ represent a primitive element of the 16-element subfield of $GF(2^8)$, where λ is a multiple of 17 and prime to 15. The choice of $\lambda = 68$ is made simple because it provides a minimum number of hardware connections in the implementation of multiplication by β [PATE80]. Therefore we can easily check that

$$0,\ 1,\ \beta,\ \beta^2, \ldots, \beta^{14}$$

form the field $GF(2^4)$, which is a subfield of $GF(2^8)$. Then the code is generated by

$$\begin{aligned} \mathbf{g}(x) &= (x + 1)(x + \beta) \\ &= x^2 + (1 + \beta)x + \beta. \end{aligned} \tag{11.24}$$

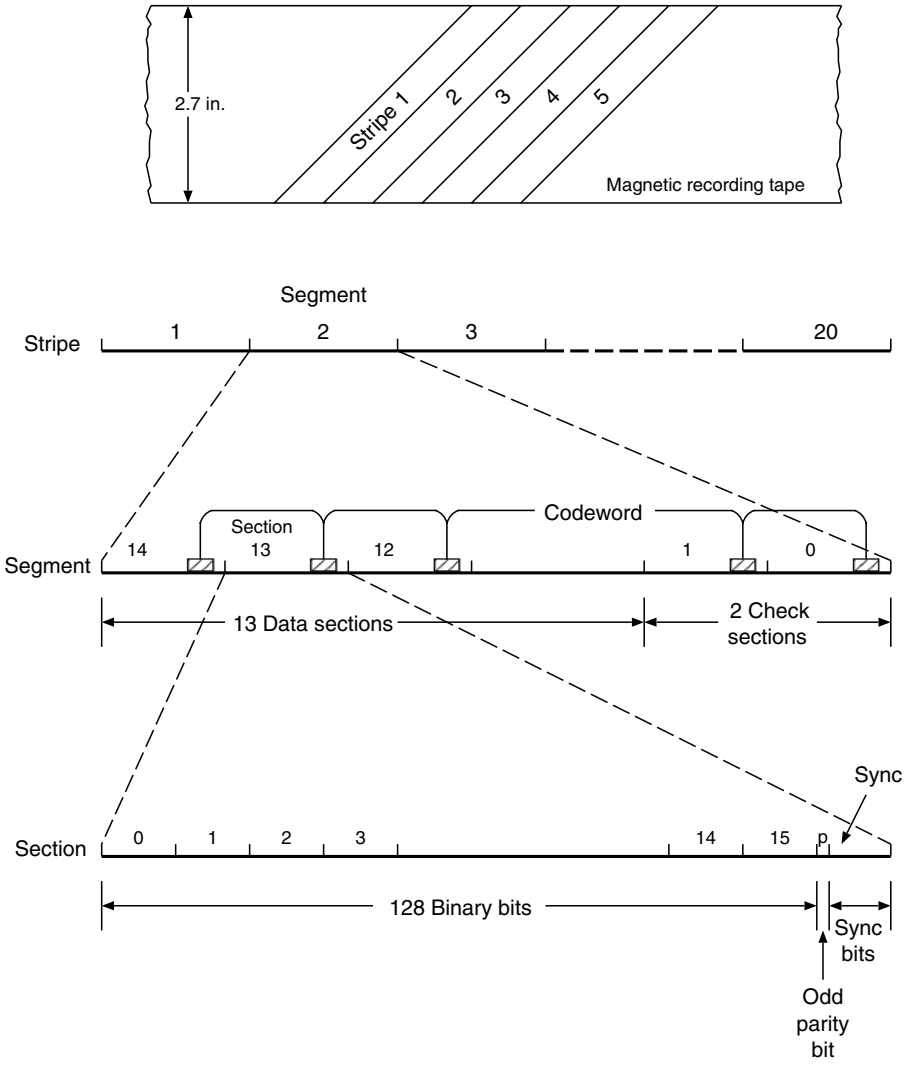


Figure 11.11 *Stripe data format.* Source: *[PATE80]. © 1980 by International Business Machines Corporation; republished by permission.*

This code can also be expressed in binary form by way of the companion matrix $\mathbf{T}$, an 8×8 matrix, obtained from the primitive polynomial of Eq. (11.23):

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

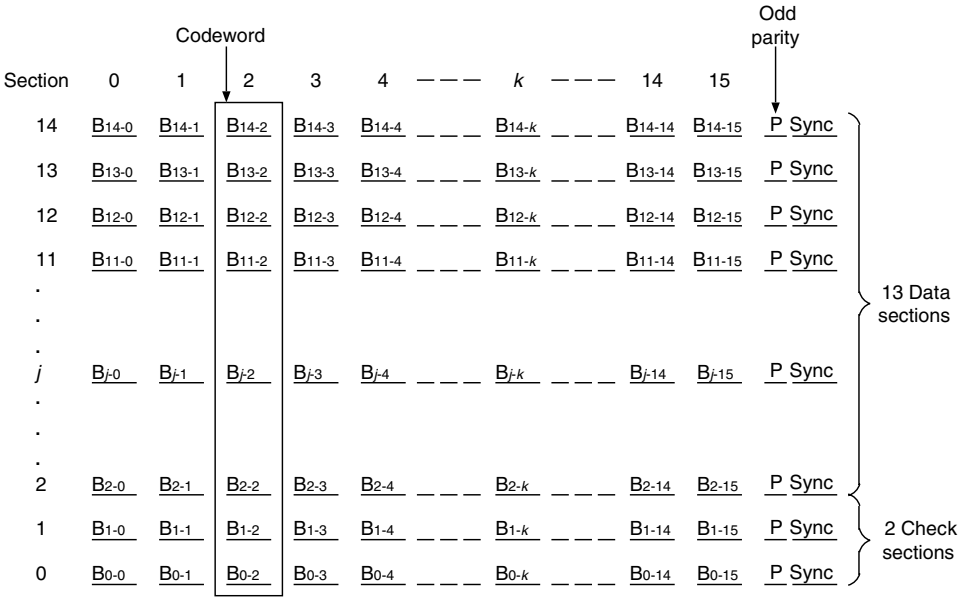


Figure 11.12 *A segment of 15 sections formed with 16 interleaved codewords.* Source: *[PATE80]. © 1980 by International Business Machines Corporation; republished by permission.*

The $\mathbf{H}$ matrix of this code is expressed as

$$\mathbf{H} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & \mathbf{I} & \dots & \mathbf{I} \\ \mathbf{I} & \mathbf{T}^{\lambda} & \mathbf{T}^{2\lambda} & \dots & \mathbf{T}^{14\lambda} \end{bmatrix}, \tag{11.25}$$

where $\mathbf{I}$ is an 8×8 identity matrix, and λ is 68. This code has minimum distance 3, and therefore it can correct any single-byte error. The coding rules are given in the form of the following (modulo-2) equations:

$$B_0 \oplus B_1 \oplus B_2 \oplus \ldots \oplus B_{14} = 0,$$
$$B_0 \oplus B_1 \cdot \mathbf{T}^{\lambda} \oplus B_2 \cdot \mathbf{T}^{2\lambda} \oplus \ldots \oplus B_{14} \cdot \mathbf{T}^{14\lambda} = 0.$$

The encoding circuit for this MSS code is presented in Figure 11.13. The block diagram of the shift register can be derived from Eq. (11.24). As soon as the 13 data symbols have been shifted into the register, the parity-check symbols B_0 and B_1 are in the low- and high-order stages.

Suppose that the i-th byte B_i is in error. Let E_i denote the error pattern. Then

$$B_i' = B_i \oplus E_i.$$

The syndrome can be obtained such that

$$\begin{aligned} S_0 &= E_i, \\ S_1 &= E_i \cdot \mathbf{T}^{i\lambda}. \end{aligned} \tag{11.26}$$

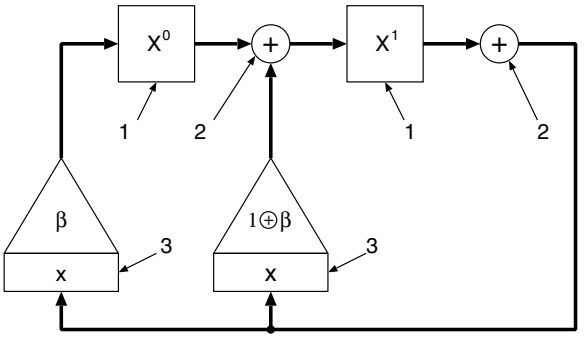


Figure 11.13 *Block diagram of encoding network.* Source: *[PATE80]. © 1980 by International Business Machines Corporation; reprinted by permission.*

Thus the error pattern E_i is determined by the syndrome S_0. From these equations we further have

$$S_1 \cdot \mathbf{T}^{-i\lambda} = S_0 = E_i, \tag{11.27}$$

that is,

$$S_1 \cdot \mathbf{T}^{(15-i)\lambda} = S_0 = E_i \tag{11.28}$$

because $\mathbf{T}^{15\lambda} = \mathbf{I}$. Therefore the error byte position i is uniquely determined from the fact that either Eq. (11.27) or Eq. (11.28) is satisfied. Error correction is accomplished by adding $S_0 = E_i$ to B_i.

When the erroneous sections are indicated by external pointers, this information is passed on to the decoder in the form of error pointers. Let i and j denote the position values of two erroneous bytes in a codeword, where $i < j$. The error patterns E_i and E_j are errors in bytes B_i and B_j, respectively, so that

$$B_i' = B_i \oplus E_i \quad \text{and} \quad B_j' = B_j \oplus E_j.$$

The syndrome can be obtained such that

$$\begin{aligned} S_0 &= E_i \oplus E_j, \\ S_1 &= E_i \cdot \mathbf{T}^{i\lambda} \oplus E_j \cdot \mathbf{T}^{j\lambda}. \end{aligned}$$

Since i and j are known, the two simultaneous equations can be solved for the two unknown variables E_i and E_j, to obtain

$$E_j = \left[\mathbf{I} \oplus \mathbf{T}^{(j-i)\lambda}\right]^{-1} \cdot \left[S_0 \oplus S_1 \cdot \mathbf{T}^{-i\lambda}\right]$$

and

$$E_i = S_0 \oplus E_j.$$

TABLE 11.2 Parameter *p* As a Function of *i*

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
p	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Source: *[PATE80]. © 1980 by International Business Machines Corporation; republished by permission.*

TABLE 11.3 Parameter *q* As a Function of (*j* − *i*)

j − *i*	1	2	3	4	5	6	7	8	9	10	11	12	13	14
q	3	6	11	12	5	7	2	9	13	10	1	14	8	4

Source: *[PATE80]. © 1980 by International Business Machines Corporation; republished by permission.*

Let $[\mathbf{I} \oplus \mathbf{T}^{(j-i)\lambda}]^{-1} = \mathbf{T}^{q\lambda}$ and $\mathbf{T}^{-i\lambda} = \mathbf{T}^{p\lambda}$. Then we obtain the following:

$$E_i = S_0 \oplus E_j$$
$$E_j = \mathbf{T}^{q\lambda} \cdot [S_0 \oplus S_1 \cdot \mathbf{T}^{p\lambda}].$$

The parameters p and q for all possible values of i and j are listed in Table 11.2 and Table 11.3, respectively.

The decoder then consists of the following five steps.

Step 1. *Multiply S_1 by the matrix $\mathbf{T}^{p\lambda}$.*
Step 2. *Add S_0 to the result of step 1.*
Step 3. *Multiply the result of step 2 by $\mathbf{T}^{q\lambda}$. This gives the error pattern E_j.*
Step 4. *Add S_0 to the result of step 3. This results in the error pattern E_i.*
Step 5. *Add E_i to byte B_i' and E_j to byte B_j'.*

Note that the designed code corrects single-byte errors and also detects random double-bit errors, mentioned in Section 6.2.

11.2 MAGNETIC DISK MEMORY CODES

Magnetic disk memory has played an important role in high-speed, large-capacity file memory of computer systems over many years. Since the 1960s disk technology has greatly improved in terms of density, storage capacity, data rate, cost, and reliability.

A disk has a higher data rate than a tape because of its higher rotation speed. Hence its sensing circuit design has to allow for greater tolerance. In the first generation of magnetic disk memories, only *simple parity checks* were used for checking data integrity. In the next generation, the *burst error detecting 16-bit polynomial code* was used for improved error detection capability. In the 1970s a high-density and high-data-rate disk (e.g., IBM 3330) established a new basic disk technology. This disk memory used a *Fire code* that had burst error correction and detection capabilities. In the latest generation of disk design, *interleaved Reed-Solomon codes* and computer-generated polynomial codes have displaced the Fire code for error correction. Besides error correction / detection codes, other important recovery techniques used to enhance reliability and data integrity are *defect skip, alternate data block*, and *reread* [HSIA81].

As with the magnetic tape systems, burst errors predominate in magnetic disk systems. Most errors are related to imperfections on disk surfaces, or surface irregularities, such as defects. The remaining errors are mostly due to heads, which are susceptible to random noise-induced errors [HOWE84].

In this section Fire codes, interleaved Reed-Solomon codes, and computer-generated polynomial codes are introduced for magnetic disk memories, and a recovering technique from single-disk failure using parity-check codes is presented for high-performance disk array systems, namely RAID systems. Codes for RAID systems will be more fully discussed in Chapter 14.

11.2.1 Fire Codes

As shown in Subsection 2.3.7, the Fire code [FIRE59], which is capable of correcting any burst of length l or less, and of simultaneously detecting any burst of length $d \geq l$, is generated by the following generator polynomial:

$$\mathbf{g}(x) = (x^c + 1)\mathbf{p}(x),$$

where $\mathbf{p}(x)$ is an irreducible polynomial of degree m with exponent e and $l \leq m$, $c \geq l + d - 1$. The length n of this code is equal to the least common multiple (LCM) of e and c:

$$n = \text{LCM}(e,\ c)$$

The number of check digits of this code is $c + m$.

Figure 11.14 shows the decoding circuit for this code. In the decoding scheme the received word $\mathbf{r}(x)$ is stored in the buffer register and, at the same time, is entered into

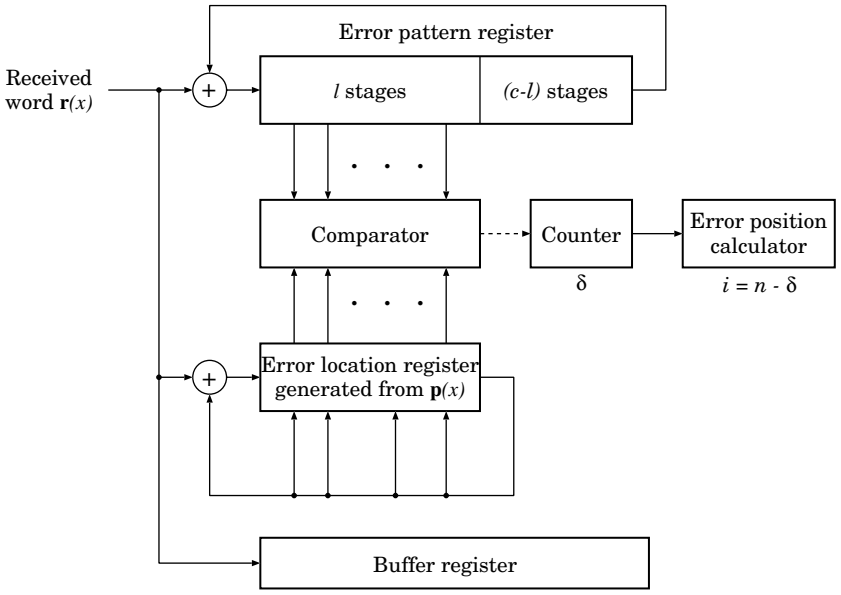


Figure 11.14 *Fire code decoder.*

the two shift registers, called the error pattern register and the error location register, to store residue polynomials divided by $x^c + 1$ and $\mathbf{p}(x)$, respectively. For this code, suppose that the transmitted codeword is $\mathbf{v}(x)$ and that the burst error polynomial is $\mathbf{b}(x)$ whose degree is less than or equal to $l - 1$. Then $\mathbf{r}(x)$ can be expressed as

$$\mathbf{r}(x) = \mathbf{v}(x) + x^i \cdot \mathbf{b}(x),$$

where i shows the position that the burst error starts. If $\mathbf{b}(x) = 0$, then $\mathbf{r}(x)$ can be divided by both $x^c + 1$ and $\mathbf{p}(x)$, and hence the residue is equal to zero. If $\mathbf{b}(x) \neq 0$, the shift registers have the following residues:

$$x^i \cdot \mathbf{b}(x) = \mathbf{s}_1(x) \mod x^c + 1, \tag{11.29}$$

$$x^i \cdot \mathbf{b}(x) = \mathbf{s}_2(x) \mod \mathbf{p}(x). \tag{11.30}$$

We use these residues, $\mathbf{s}_1(x)$ and $\mathbf{s}_2(x)$, to obtain $\mathbf{b}(x)$ and the value i. After $n - i$ shifts of these shift registers, we have the following residues:

$$\begin{aligned} \mathbf{s}_1(x) \cdot x^{n-i} &= x^n \cdot \mathbf{b}(x) \\ &= \mathbf{b}(x) \mod x^c + 1, \\ \mathbf{s}_2(x) \cdot x^{n-i} &= x^n \cdot \mathbf{b}(x) \\ &= \mathbf{b}(x) \mod \mathbf{p}(x). \end{aligned}$$

So, when the contents of the lower part of the l registers of the error pattern register are equal to those of the error location register after δ shifts of these registers, the burst error pattern, expressed as polynomial $\mathbf{b}(x)$, can be set in these registers. In addition to $\mathbf{b}(x)$, the error position i can be computed from the number of shifts $\delta(= n - i)$, and hence $i = n - \delta$.

The foregoing decoding method takes a lot more decoding time when i is small compared to n. In that case the high-speed decoding method is desirable [CHIE69]. To see this, assume that c and e are relatively prime. Figure 11.15 shows a high-speed decoding circuit.

In this decoding scheme the received word $\mathbf{r}(x)$ is simultaneously entered into the buffer register, the error pattern register, and the error location register. Then the residues expressed by Eqs. (11.29) and (11.30) are obtained in the error pattern register and the error location register, respectively. At this stage we can decode from the obtained residues the received word as follows:

$$\begin{aligned} &\mathbf{s}_1(x) = \mathbf{s}_2(x) = 0 \rightarrow \mathbf{r}(x): \text{ error free,} \\ &\left.\begin{aligned} &\mathbf{s}_1(x) \neq 0 \text{ and } \mathbf{s}_2(x) = 0 \\ &\mathbf{s}_1(x) = 0 \text{ and } \mathbf{s}_2(x) \neq 0 \end{aligned}\right\} \rightarrow \mathbf{r}(x): \text{ uncorrectable.} \end{aligned}$$

If $\mathbf{s}_2 \neq 0$ and $\mathbf{s}_2 \neq 0$, the decoding can be processed by the following algorithm:

Step 1. *Shift the error pattern register until the contents of the higher $c - l$ registers are all 0's, and store the required number of shift δ_0. If the contents of the error pattern register are not equal to all 0's after $c - l$ shifts, an uncorrectable error is assumed to have occurred; therefore stop the decoding process.*

Step 2. *Shift the error location register until its contents are equal to those of the lower l bits of the error pattern register, and store the number of shift δ_1. If the contents of the lower part of the error pattern register are not equal to those of the error location register after e−1 shifts, an uncorrectable error is assumed to have occurred; therefore stop the decoding process.*

Step 3. *Calculate the error position i from the value δ_0 and δ_1. Hence the errors can be corrected by using the error pattern $\boldsymbol{b}(x)$. That is, the contents of the lower l bits of the error pattern register denote $\boldsymbol{b}(x)$.*

In the algorithm the error position can be calculated by the *Chinese remainder theorem* [BLAH83] such that

$$\begin{aligned}\delta_0 &= c - i \\ &= -i \mod c, \\ \delta_1 &= e - i \\ &= -i \mod e.\end{aligned}$$

Then $-i = \delta_0 \cdot A_0 \cdot e + \delta_1 \cdot A_1 \cdot c \bmod n$, where A_0 and A_1 are such integers that satisfy

$$A_0 \cdot e + A_1 \cdot c = 1 \mod n.$$

The computation of A_0 and A_1 can be easily done off line with the numbers $A_0 \cdot e$ and $A_1 \cdot c$ stored in the error position calculator shown in Figure 11.15. In this decoding the

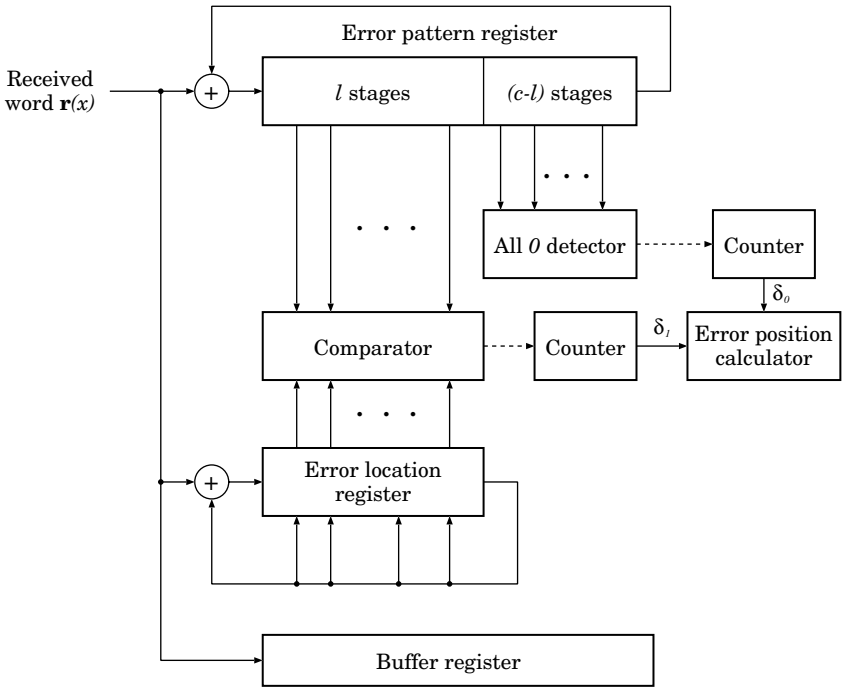


Figure 11.15 *High-speed decoder for the Fire code.*

maximum number of shifts is $e + c - 2$; recall that in the former decoding, it was $e \cdot c - 1$. Hence this decoding has proceeded at a much faster rate.

Next we consider a class of codes that is a *generalization of the Fire codes* [CHIE69]. They are generated by a polynomial of the form

$$\mathbf{g}(x) = (x^c + 1) \prod_{i=1}^{h} \mathbf{p}_i(x), \tag{11.31}$$

where $\mathbf{p}_i(x)$ is a distinct irreducible polynomial having exponent e_i and degree r_i. It is further assumed that the e_i's ($i = 1,\ 2,\ \ldots,\ h$) do not divide c. The code length is

$$n = \text{LCM}(c,\ e_1,\ e_2,\ \ldots,\ e_h).$$

Theorem 11.1 *[CHIE69]. The code generated by $\mathbf{g}(x)$ given by Eq. (11.31) detects all error bursts of length $\leq d$; it corrects all bursts $\mathbf{b}(x)$ of length $\leq l$ that are relatively prime to $\prod_{i=1}^{h} \mathbf{p}_i(x)$, provided that $c \geq l + d - 1$ and $l \leq \sum_{i=1}^{h} r_i$.*

Equations that determine the error position i can be written as

$$-i = \delta_0 \cdot A_0 \cdot \prod_{j=1}^{h} e_j + c \cdot \sum_{j=1}^{h} \delta_j \cdot A_j \cdot \prod_{\substack{k=1 \\ k \neq j}}^{h} e_k, \tag{11.32}$$

$$A_0 \cdot \prod_{j=1}^{h} e_j + c \cdot \sum_{j=1}^{h} A_j \cdot \prod_{\substack{k=1 \\ k \neq j}}^{h} e_k = 1. \tag{11.33}$$

The IBM 3330-compatible disk systems use the *generalized Fire code* generated by the following polynomial:

$$\mathbf{g}(x) = (x^{22} + 1) \prod_{i=1}^{3} \mathbf{p}_i(x),$$

where

$$\mathbf{p}_1(x) = x^{11} + x^7 + x^6 + x + 1$$
$$\mathbf{p}_2(x) = x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$$
$$\mathbf{p}_3(x) = x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1$$

are distinct irreducible polynomials. The exponents of $\mathbf{p}_1(x)$, $\mathbf{p}_2(x)$, and $\mathbf{p}_3(x)$ are 89, 13, and 23, respectively. Therefore the code length is

$$n = \text{LCM}(22,\ 89,\ 13,\ 23) = 585,442.$$

It has $22 + 11 + 12 + 11 = 56$ check bits and is capable of correcting any single error burst of length 11 bits or less and detecting any single error burst of length 22 bits or less [GLOV91].

From Eqs. (11.32) and (11.33), the error position can be determined as follows:

$$i = \delta_0 A_0 e_1 e_2 e_3 + \delta_1 A_1 c e_2 e_3 + \delta_2 A_2 c e_1 e_3 + \delta_3 A_3 c e_1 e_2 \mod n, \tag{11.34}$$

where A_0, A_1, and A_3 satisfy the following.

$$A_0 e_1 e_2 e_3 + A_1 c e_2 e_3 + A_2 c e_1 e_3 + A_3 c e_1 e_2 = -1 \mod n. \tag{11.35}$$

For this code we can find that

$$\begin{aligned} A_0 e_1 e_2 e_3 &= -1 \mod c, \\ A_1 c e_2 e_3 &= -1 \mod e_1, \\ A_2 c e_1 e_3 &= -1 \mod e_2, \\ A_3 c e_1 e_2 &= -1 \mod e_3. \end{aligned} \tag{11.36}$$

The minimum integers that satisfy Eqs. (11.35) and (11.36) are $A_0 = 5$, $A_1 = 78$, $A_2 = 6$, and $A_3 = 10$. These integers are substituted into Eq. (11.34), and then we have the following equation that can determine the error position:

$$i = 133,055\delta_0 + 513,084\delta_1 + 270,204\delta_2 + 254,540\delta_3 \mod 585,442.$$

The decoding circuit for this code is shown in Figure 11.16.

Shift operations of the error pattern register are performed until the contents of the upper 11 stages of the register are all 0's. Shift operations of the error location register are also performed until all contents of the three error location registers are equal to those of the lower 11 stages of the error pattern register. Given that these numbers of shifts are δ_0, δ_1, δ_2, and δ_3, then $K_0 = 133,055\delta_0$, $K_1 = 513,084\delta_1$, $K_2 = 270,204\delta_2$, and $K_3 = 254,540\delta_3$ are calculated, and hence i can be obtained by $i = K_0 + K_1 + K_2 + K_3 \mod n$.

Another faster error correction scheme using the ordinary Fire code is proposed in [ADI84]. This is a hybrid technique combining sequential error-trapping and table lookup techniques. The decoder needs at most $c + m - 1$ shift cycles to find both the burst error pattern and its location. The decoder can be implemented by making use of programmable read-only memories (PROMs) and programmable logic arrays suitable for LSI implementation.

11.2.2 Interleaved RS S*b*EC-D*b*ED Codes

Distance-4 RS Code Interleaved to Degree 3 The magnetic disk system (e.g., found in the IBM 3370 and 3380 systems) uses an interleaved Reed-Solomon (RS) distance-4 code, as discussed in [HODG80], in a fixed-block data format. Each data block consists of 512 bytes. For error control, nine check bytes are appended to it. The code of this system is a shortened Reed-Solomon distance-4 code with symbols from the field $GF(2^8)$. The generator polynomial of the code is

$$\mathbf{g}(x) = (x + 1)(x + \alpha)(x + \alpha^{-1}),$$

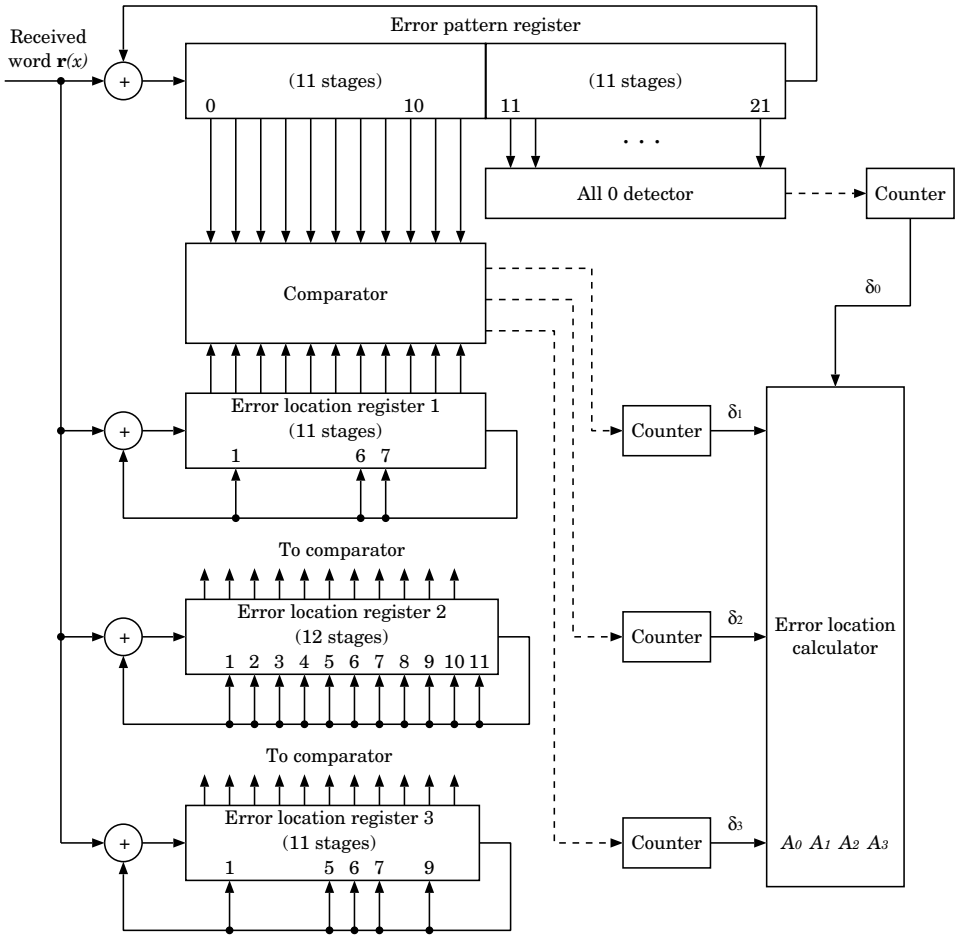


Figure 11.16 *Decoding circuit for the generalized Fire code used in IBM 3330-compatible disk system.*

where α is the primitive element of $GF(2^8)$ and is a root of $\mathbf{p}(x) = x^8 + x^4 + x^3 + x^2 + 1$. The RS distance-4 code is capable of correcting any single-symbol (one-byte) error and simultaneously detecting any combination of double-symbol (two-byte) errors. The code is a $(174,\ 171)$ code over $GF(2^8)$.

For encoding, a data block is divided into three subblocks, $\mathbf{D}_{-1}$, $\mathbf{D}_0$, and $\mathbf{D}_1$, each consisting of 171 bytes. We represent these three subblocks in polynomial form as follows:

$$\mathbf{D}_i(x) = D_{i,170}x^{170} + D_{i,169}x^{169} + \cdots + D_{i,1}x + D_{i,0}, \qquad i = -1, 0, 1,$$

where each byte $D_{i,k}$, $(i = -1,\ 0,\ 1,\ k = 0,\ 1,\ \ldots,\ 170)$, is regarded as a symbol in $GF(2^8)$. The three check bytes for $\mathbf{D}_i$, denoted as $C_{i,-1}$, $C_{i,0}$, and $C_{i,1}$, $i = -1, 0, 1$, are obtained as follows:

$$C_{i,j} = x \cdot \mathbf{D}_i(x) \mod (x + \alpha^j),$$
$$i = -1,\ 0,\ 1, \quad j = -1,\ 0,\ 1.$$

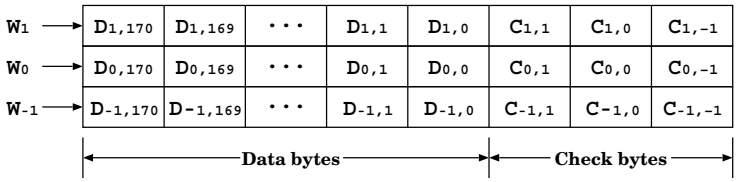


Figure 11.17 *Codewords, $\mathbf{W}_1$, $\mathbf{W}_0$, and $\mathbf{W}_{-1}$.*

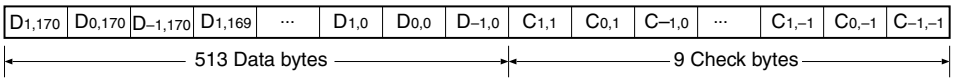


Figure 11.18 *Overall interleaved codeword.*

Note the difference between this encoding method and the method used for encoding cyclic codes, where all three check bytes are computed by dividing $x^3 \cdot \mathbf{D}_i(x)$ by $\mathbf{g}(x)$. After the three check bytes for $\mathbf{D}_i$ have been formed, they are appended to $\mathbf{D}_i$ to form a codeword $\mathbf{W}_i$, as shown in Figure 11.17.

The three codewords $\mathbf{W}_1$, $\mathbf{W}_0$, and $\mathbf{W}_{-1}$ are interleaved to form a coded block as shown in Figure 11.18. Therefore the overall code is the $(174,\ 171)$ RS code interleaved to degree 3. This code is capable of correcting any burst error confined to three consecutive bytes and detecting any burst error confined to six consecutive bytes.

When a coded block is read from the disk, it is decomposed into three subwords, $\mathbf{W}'_1$, $\mathbf{W}'_0$, and $\mathbf{W}'_{-1}$, where

$$\begin{aligned} \mathbf{W}'_i &= (\mathbf{D}'_i, C'_{i,1}, C'_{i,0}, C'_{i,-1}), \\ i &= 1, 0, -1. \end{aligned}$$

Syndrome $S_{i,j}$ $(i = 1, 0, -1, j = 1, 0, -1)$ is obtained as follows:

$$\begin{aligned} S_{i,j} &= x \cdot \mathbf{D}'_i(x) + C'_{i,j} \mod (x + \alpha^j) \\ &= \alpha^j \cdot \mathbf{D}'_i(\alpha^j) + C'_{i,j}, \end{aligned}$$

where $\mathbf{D}'(x) = D'_{i,170}x^{170} + \ldots + D'_{i,1}x + D'_{i,0}$. If $S_{i,1} = S_{i,0} = S_{i,-1} = 0$, the word is error free. If all syndromes $S_{i,1}$, $S_{i,0}$, and $S_{i,-1}$ are nonzeros, there are single-byte errors in the readout word. This is because if error $\alpha^m (0 \le m \le 254)$ is added to the data symbol $D_{i.l}$ $(l = 0,\ 1,\ \ldots,\ 170)$, then the readout word is expressed as

$$\begin{aligned} D'_{i,l} &= D_{i,l} + \alpha^m, \\ D'_{i,k} &= D_{i,k}, \qquad k \ne l, k = 0, 1, \ldots, 170, \\ C'_{i,j} &= C_{i,j}. \end{aligned}$$

Hence not all zero syndromes can be obtained as

$$\begin{aligned} S_{i,1} &= \alpha^{m+l+1}, \\ S_{i,0} &= \alpha^m, \\ S_{i,-1} &= \alpha^{m-(l+1)}. \end{aligned}$$

TABLE 11.4 Error Information Based on Syndromes

$S_{i,1}$	$S_{i,0}$	$S_{i,-1}$	Error information
0	0	0	Error free
NZ	NZ	NZ	Single-byte errors
NZ	0	0	Errors in $C'_{i,1}$
0	NZ	0	Errors in $C'_{i,0}$
0	0	NZ	Errors in $C'_{i,-1}$
NZ	NZ	0	Errors in $C'_{i,1}$ and $C'_{i,0}$
0	NZ	NZ	Errors in $C'_{i,0}$ and $C'_{i,-1}$
NZ	0	NZ	Double-byte errors

Note: NZ: nonzero syndrome.

In this case the error pattern can be expressed as $S_{i,0}$, and error position can be determined by the following relation:

$$\frac{S_{i,1}}{S_{i,0}} = \frac{\alpha^{m+l+1}}{\alpha^m} = \alpha^{l+1},$$
$$\frac{S_{i,0}}{S_{i,-1}} = \frac{\alpha^m}{\alpha^{m-(l+1)}} = \alpha^{l+1}.$$

Other information regarding errors is given in Table 11.4.

Two-Level Coding for Multiple-Burst Errors Another coding architecture for the correction of multiple-burst errors that has been applied to IBM 3380J and 3380K disk files [PATE89] is a two-level coding scheme. This coding scheme offers high coding efficiency along with a fast decoding strategy that closely matches the requirements of online correction of multiple bursts of errors. The first level, on a smaller block size, provides very fast correction of most errors commonly encountered in disks. The second level, on a larger block size, provides a reserved capability for correcting additional errors that may be encountered in a device with symptoms of a weaker component or oncoming failure.

The basic error event is a *byte-in-error.* A burst error may cause correlated errors in adjacent bytes; however, sufficient interleaving is assumed to effectively randomize these errors. With appropriate interleaving, all bytes are assumed to be seen by the coding scheme as equally likely to be in error. In disk files, major defects in the media are avoided by means of *surface analysis test* and *defect-skipping strategy.* The error correction code is expected to provide coverage for errors caused by noise and small defects that cannot be identified easily in the surface analysis test. These errors are usually two to four bits long [PATE89]. Therefore two-way or three-way byte interleaving of the codewords is adequate in magnetic disks. The data format of the IBM 3380J and 3380K disks is designed with a two-level architecture consisting of subblocks within a block, combined with two-way interleaved codewords. This is shown in Figure 11.19. Each subblock (except the last) consists of 96 data bytes and six first-level check bytes in the form of two interleaved codewords. At the end of the block, six additional check bytes are appended, two of which are used for second-level error correction and the remaining four for an overall data-integrity check after correction of the errors at both levels. The two-way interleaved two-level code of Figure 11.19 provides correction of at least one byte error in each subblock and detection of up to two byte errors in any one of the many subblocks of a block.

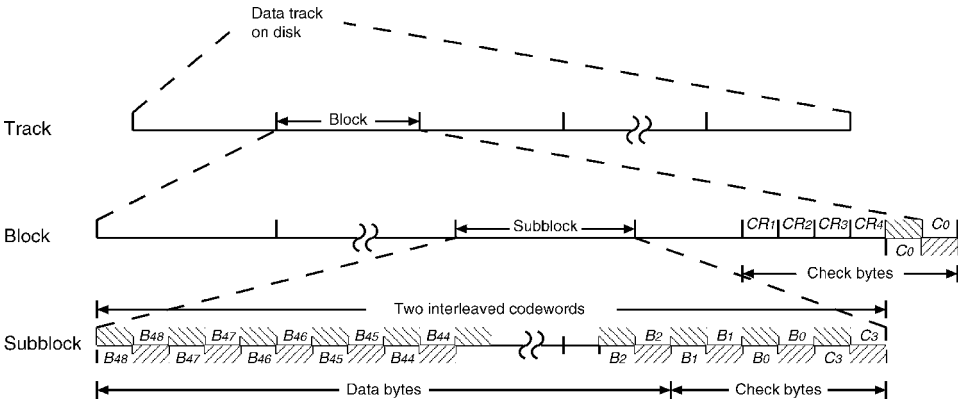


Figure 11.19 *Data format of the IBM 3380J and 3380K disk files.* Source: *[PATE89]. © 1989 by International Business Machines Corporation; republished by permission.*

The first-level codeword consists of three check bytes denoted by C_3, B_0, and B_1, and of m data bytes denoted by $B_2, B_3, \ldots, B_{m+1}$. The $\mathbf{H}$ matrix for a block is expressed as

$$\mathbf{H} = \begin{array}{c|cccccccccccc|c} & \multicolumn{12}{c}{\longleftarrow\ \text{block}\ \longrightarrow} & \\ & \multicolumn{12}{c}{\longleftarrow\ \text{subblock}\ \longrightarrow} & \\ \hline \ldots & B_{m+1} & C_3 & B_0 & B_1 & B_2 & \ldots & B_{m+1} & C_3 & B_0 & \ldots & \ldots & C_0 \\ & & \mathbf{0} & \mathbf{I} & \mathbf{T} & \mathbf{T}^2 & & \mathbf{T}^{m+1} & & & & & \mathbf{0} \\ & \ldots & \mathbf{0} & \mathbf{I} & \mathbf{T}^2 & \mathbf{T}^4 & \ldots & \mathbf{T}^{2(m+1)} & \ldots & & \ldots & \ldots & \mathbf{0} \\ & & \mathbf{I} & \mathbf{I} & \mathbf{T}^3 & \mathbf{T}^6 & & \mathbf{T}^{3(m+1)} & & & & & \mathbf{0} \\ \hline \ldots & \mathbf{I} & \mathbf{0} & \mathbf{I} & \mathbf{I} & \mathbf{I} & \ldots & \mathbf{I} & \mathbf{0} & \mathbf{I} & \ldots & \ldots & \mathbf{I} \end{array},$$

where B_i is an 8-bit, $i = 0, 1, \ldots, m+1 \leq 2^8 - 2$, $\mathbf{T}$ is a companion matrix of the primitive polynomial of degree 8, meaning $\mathbf{g}(x) = x^8 + x^7 + x^5 + x^3 + 1$. We use $m = 48$. The code in each subblock is an extended RS code with a Hamming distance-4. The second-level codeword consists of n subblocks with one additional check byte denoted by C_0 at the end. This check byte is the modulo-2 sum of all subblock bytes excluding C_3 and accumulated over all subblocks. It is readily seen that for $n = 1$, the $\mathbf{H}$ matrix shown above represents a code that is an extended RS code for correction of double-byte errors. In the case of n greater than 1, the second-level codeword (i.e., the block codeword) can be viewed as modulo-2 superposition of n first-level codewords (i.e., the n subblock codewords). Double-byte errors in this superpositioned codeword are correctable. Suppose that a block consisting of n subblocks encounters multiple bytes in error. If these errors are located in separate subblocks, each error will be corrected as a single-byte error in the corresponding subblock. If one of the subblocks has double bytes in error, the first-level code will detect these errors. Then, at the second-level, the first-level syndromes, together with the second-level syndromes, will be reprocessed for the correction of the subblock with the double byte in error. If any subblock has more than double bytes in error, or if two or more subblocks have multiple bytes in error, these errors cannot be corrected. The two-level code includes four additional check bytes at the second level shown in Figure 11.19. They are denoted as CRC checks. These four CRC check bytes provide an overall data-integrity confirmation against miscorrections in the presence of an excessive number of errors.

In general, the data format and the error control capabilities of the two-level coding scheme can be described as follows [PATE89]. Let each subblock be a codeword from a code with a minimum Hamming distance of d_1 consisting of m data bytes and r_1 check bytes. Also let the block consist of n subblocks and r_2 check bytes that are shared by its subblocks. The data part of the block-level code is viewed as modulo-2 superposition of n subblock codewords. The r_2 check bytes (either independently or along with the superpositioned r_1 check bytes of all subblocks) provide a minimum Hamming distance of d_2 (over one subblock) at the block-level where $d_2 > d_1$. The codewords of both levels may be interleaved in order to provide correction for burst errors or clustered multi-byte errors. The decoding process provides correction of up to t_1 errors and detection of up to $t_1 + c$ errors in each subblocks, where $d_1 = 2t_1 + c + 1$. If the number of errors in a subblock exceeds the error correcting capability at the first-level, such errors are either left uncorrected or are miscorrected. If all errors are confined to one subblock and exceed the error correcting capability at the first-level, the block-level code will provide correction of up to t_2 errors, where $d_2 \geq 2t_2 + 1$. However, many combinations of errors in multiple subblocks, including t_2 errors not confined to one subblock, are also correctable.

11.2.3 Computer-Generated Polynomial Codes

A *computer-generated polynomial code* has been applied to recent small-sized 2.5 in or 3.5 in *HDD (hard disk drive)* units. An example is the code generated by the polynomial $\mathbf{g}(x) = x^{32} + x^{28} + x^{26} + x^{19} + x^{17} + x^{10} + x^6 + x^2 + 1$, which has a maximum code length of 526 bytes including 32 check bits, and has an error control capability of detecting any single burst of 14 bits or less as well as correcting any single burst of 8 bits or less with a miscorrection probability of 1.25×10^{-4} [WEST, GLOV91]. An extended polynomial code with degree 48, 56, or 64, has been generated by computer search and employed in the recent HDD units. A disk controller is mounted on each recent HDD unit and it includes encoding / decoding functions.

11.2.4 Introduction to Disk Array Codes

In the 1980s parallel disk arrays using multiple disks were discussed as a way to ensure total I/O performance. The driving force behind the parallel arrays concept was the rapid improvement in semiconductor technology that made possible faster microprocessors and larger primary memory systems. The faster microprocessors required larger capacity, higher performance secondary storage systems.

Large disk arrays are highly vulnerable to disk failure. The obvious solution is to employ redundancy in the form of error correcting codes to tolerate disk failures. However, redundancy has negative consequences. Since all write operations must update the check information, the performance of writes in redundant disk arrays can be significantly worse than that of writes in nonredundant disk arrays.

In recent years the interest in *redundant array of independent disks*, called RAID, has grown explosively [GIBS92]. Successful parity-check coding techniques for RAID levels 3, 4 and 5 will be discussed here and in Section 14.1. These RAID architectures are shown in Figure 11.20.

Before discussing the architectures, we should keep in mind that each disk has its own strong error correction and detection capability due to the Fire codes, interleaved

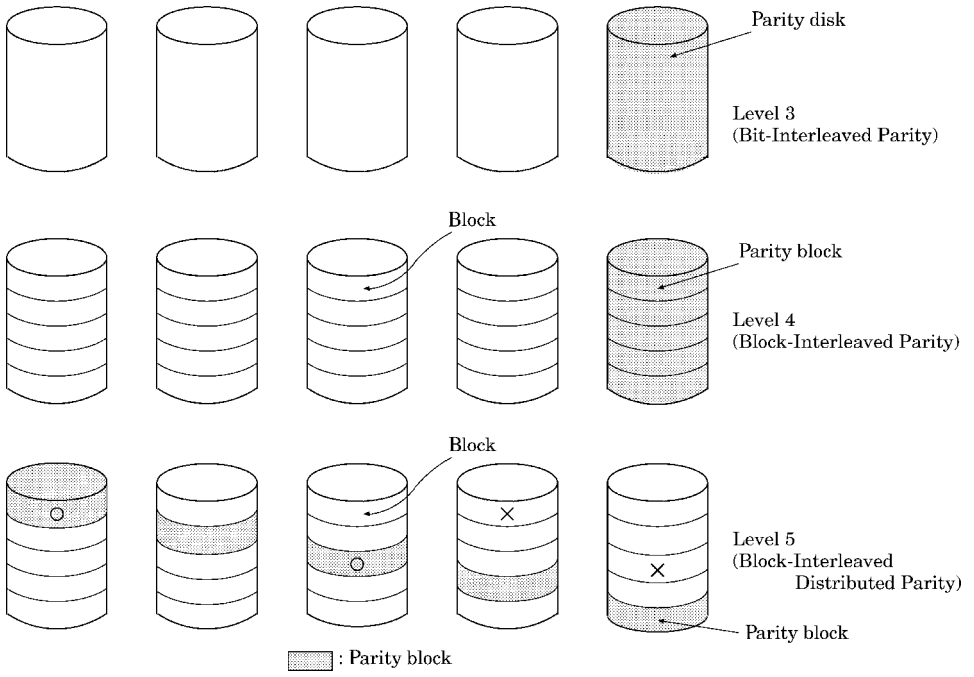


Figure 11.20 *RAID architecture.*

Read-Solomon (RS) codes, or computer-generated polynomial codes mentioned in the previous subsections. As a disk detects unrecoverable errors by these codes, the information is sent to the disk controller, which is located upward among the disk arrays pictured in Figure 11.20. In RAID system the disk controller gathers every data bit readout from all disks together with the information above, and therefore can identify which disk is in error or in an unrecoverable state. By this information the simple parity-check code can recover the lost information in a single disk; that is, single erasure correction can be performed by using distance-2 parity-check codes.

We first consider the RAID system with n data disks and one parity disk. The simple parity-check bit p of the parity disk is determined as

$$p = d_0 \oplus d_1 \oplus \ldots \oplus d_{i-1} \oplus d_i \oplus d_{i+1} \oplus \ldots \oplus d_{n-1},$$

where, d_i, $i = 0, 1, \ldots, n-1$, is the data bit of the i-th disk. If the i-th disk has failed, then the data d_i can be recovered by using the readout data of the other disks and the readout parity data of the parity disk as

$$\hat{d}_i = d'_0 \oplus d'_1 \oplus \ldots \oplus d'_{i-1} \oplus d'_{i+1} \oplus \ldots d'_{n-1} \oplus p',$$

where d'_i and p' are the readout i-th data and the readout parity data, respectively.

In a RAID level 3 (i.e., a *bit-interleaved parity disk array*), the data are conceptually interleaved bitwise over the data disks, and a single parity disk is added to tolerate any single-disk failure. Each read request accesses all data disks, and each write request accesses all data disks and the parity disk.

In a RAID level 4 (i.e., a *block-interleaved parity disk array*), the data are interleaved as in the level 3 except that the interleaving is across disks in blocks of arbitrary size rather than in bits. Each block size is called a *striping unit*. Read requests smaller than the striping unit access only a single data disk. Write requests must update the requested data blocks, and must compute and update the parity block. For large-writes that touch the blocks on all disks, the parity is easily computed by exclusive-ORing the new data in each disk. For small-write requests that update only one data and apply the differences to the parity block, four disk operations are required: one to write the new data, two to read the old data and old parity for computing the new parity, and one to write the new parity. This is referred to as a *read-modify-write procedure*. The new parity can be calculated from the old data, the old parity, and the new data as follows:

$$\text{New parity} = (\text{Old data}) \oplus (\text{New data}) \oplus (\text{Old parity}).$$

Since this type of disk array has only one parity disk that must be updated on all write operations, the parity disk can easily become a bottleneck.

In a RAID level 5 (i.e., a *block-interleaved distributed-parity disk array*), the *parity disk bottleneck* is eliminated by distributing the parity blocks uniformly over all of the disks. That is, several reads and writes can be serviced concurrently. For example, the small-writes on the blocks of the fourth and the fifth disks marked by an $\times$ in Figure 11.20 can be operated simultaneously because the corresponding parity blocks marked by a $\bigcirc$ are in different disks, that is, in the first and the third disks. In addition to this, small-read on the second disk can also be performed simultaneously. Block-interleaved distributed-parity disk arrays have the best small-read, large-read, and large-write performance among any redundant disk arrays. Small-write requests, however, need to perform read-modify-write operations to update parity. In the RAID level 5, a performance evaluation is done for the different parity placements [LEE93].

Some extended practical schemes for tolerating double-disk failures in RAID architectures are proposed in [HELL94], [BLAU95], [XU99a, 99b]. Theoretically, in order to retrieve the information lost in two failed (erased) disks, we need at least two redundant disks. In coding theory this is known as the *Singleton bound*. These schemes give an efficient encoding procedure that is based on exclusive-OR (or XOR) operations and *independent parities*, and also give a simple decoding procedure for two erasures and for a single error with small number of XOR operations. This will be discussed in detail in Section 14.1.

The MDS (maximum distance separable) array codes with small number of 1's in the parity-check matrix, namely the low-density parity-check matrix, require a small number of additions (XORs) and hence enable high-speed decoding upon disk failure [BLAU93, 96, 99], [HELL94]. As was discussed in Subsection 3.1.2, Blaum and Roth [BLAU99] demonstrated how to design low-density MDS array codes, and also the lower bounds on the number of nonzero elements in the systematic parity-check matrix, along with the upper bound on the length of any MDS array code that attains those lower bounds.

To prepare for the detailed discussion on practical erasure-correcting codes in Section 14.1, we need to give attention to the following four important metrics for redundancy in disk arrays: *mean time to data loss*, *check disk overhead*, *update penalty*, and *group size* [HELL94].

The *mean time to data loss* is the expected number of repair periods until an unrecoverable set of failures occurs. Using independent exponential disk lifetime with

mean M, we can calculate the probability of y failures (erasures) in a repair period T as

$$\binom{N}{y}(1-e^{-T/M})^{y}(e^{-T/M})^{N-y},$$

where N is the total number of disks. The *check disk overhead* for a coding scheme is the ratio of the number of check disks to that of information disks or to the total number of disks. The *update penalty* of a coding scheme is the number of check disks whose contents should be changed when a change is made in the contents of a given information disk. The number of disks accessed to effect a small data update has to be minimized. The set of disks that must be accessed during the reconstruction of a single failed disk form a group. The *group size* is an important metric because the duration of reconstruction is likely to scale linearly with the number of disks to be read. Until reconstruction is completed, the group size indicates the number of disks that must be accessed.

The latter three metrics are easily expressed in terms of parity-check matrix $\mathbf{H}$. The check disk overhead is the ratio of the number of rows in $\mathbf{H}$ (i.e., the number of check columns r) to the number of columns in $\mathbf{H}$ (i.e., the code length n). The size of a group is the weight of the row for that group, meaning the number of 1's in that row. The update penalty for any information disk, which is the number of groups including that disk, is the weight of its column.

11.3 OPTICAL DISK MEMORY CODES

Digital optical disks are a relatively new technology for storing data. Each disk is coated with special reflective materials. Reading and writing are performed by a laser. More specifically, reading involves determining the reflectivity of a given position on the disk, while writing usually consists of melting holes into the coating (at a high-power setting of the laser). In erasable disks, however, magneto-optical recording has a different reading / writing principle, one based on the Faraday-effect and Curie-point reorientation. The bit density on digital optical disks can be much higher, since optical recording permits a much finer ''grain'' than magnetic recording. Therefore this medium can be expected to play an increasingly important role in memory systems—document file, image file, digital data file for large computer systems, and so forth [LEIS84].

Errors are primarily due to imperfections in the optical disk medium, and are also due to a focusing shift in recording and random noise in reading. Therefore both burst error and random error correcting facilities are needed for optical disk memories [SAIT86, ITAO87]. Presently, especially for erasable disks, powerful error correction techniques, mainly a *doubly encoded Reed-Solomon (RS) code* (a combination of a cyclic code and an RS code) and an RS code with a large minimum Hamming distance, are being applied to attain the same level of reliability as that of the magnetic media. As for the recent DVDs (*digital versatile discs*), the powerful error correcting RS product codes have been applied for correcting two-dimensional clustered errors.

11.3.1 Cross-interleaved RS Code (CIRC)

In the optical disk systems, burst errors as well as random errors have to be considered. Thus a code is required to correct both types of errors.The *C*ross-*I*nterleaved *R*eed-Solomon

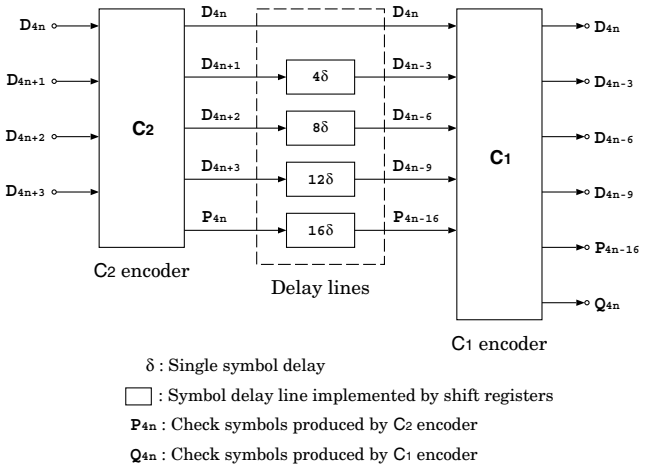


Figure 11.21 *CIRC encoder.*

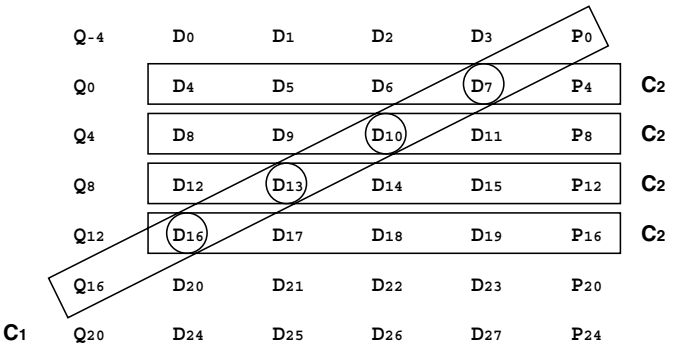
*C*ode (CIRC) is a new class of doubly encoded codes in which the second RS code encodes the delayed and dispersed (i.e., *cross-interleaved*) output of the first encoded RS code [DOI79]. To illustrate the concept behind this class of codes, a simple example is provided.

Example 11.1

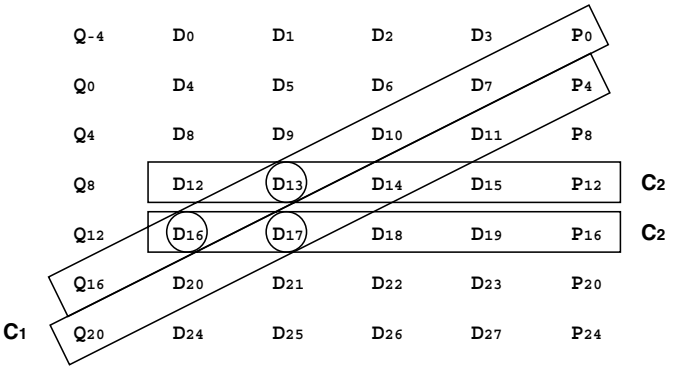
Let the two codes that can individually correct single-symbol errors be $\mathbf{C}_1$ and $\mathbf{C}_2$. Figure 11.21 shows the encoding circuit of the CIRC. Assume that the check symbols P and Q are represented as two symbols for the code $\mathbf{C}_2$ and $\mathbf{C}_1$, respectively.

Check symbols P_{4n} are produced from the consecutive four symbols, D_{4n}, D_{4n+1}, D_{4n+2}, and D_{4n+3} by using the $\mathbf{C}_2$ encoder. The encoded symbols, D_{4n}, D_{4n+1}, D_{4n+2}, D_{4n+3}, and P_{4n}, are delayed by unequal length and then yield to D_{4n}, D_{4n-3}, D_{4n-6}, D_{4n-9}, and P_{4n-16}. These delays are different for each symbol. Check symbols Q_{4n} are produced from the delayed symbols by using the $\mathbf{C}_1$ encoder. The delay lines, which characterize the cross-interleaved concept, yield an encoding scheme such that check symbols Q_{4n} can be produced from the symbols included in the positive slope shown in Figure 11.22.

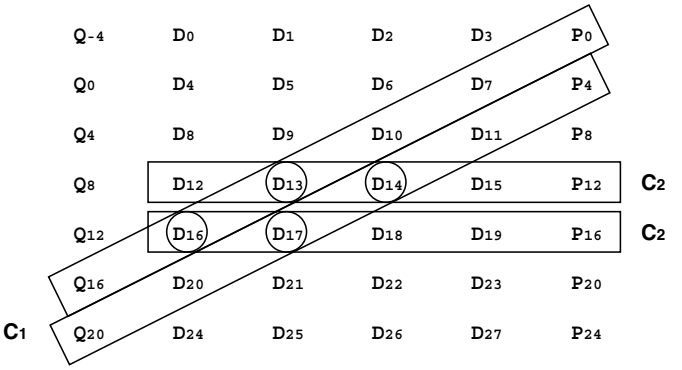
Figure 11.22 also shows the decoding concept of this code. As a result of cross-interleave, even if there are consecutive four symbols errors, for example, errors in D_7, D_{10}, D_{13}, and D_{16} (circled in Figure 11.22(a), that cannot be corrected in $\mathbf{C}_1$ decoding. These four symbols errors, however, are dispersed into single-symbol errors in each $\mathbf{C}_2$ code, and hence these can be corrected in $\mathbf{C}_2$ decoding. In this case, $\mathbf{C}_i$ decoding, $i = 1,\ 2$, means the decoding of the code $\mathbf{C}_i$. If there exist errors in three symbols, for example, in D_{13}, D_{16}, and D_{17} (shown in Figure 11.22(b), these errors can also be corrected in sequential steps such that a single-symbol error in D_{13} can be corrected by $\mathbf{C}_2$ decoding, and then a single-symbol error in D_{16} can be corrected by $\mathbf{C}_1$ decoding; finally, a single-symbol error in D_{17} can be corrected by $\mathbf{C}_2$ decoding. In Figure 11.22(c), however, errors in four symbols, for example, in D_{13}, D_{14}, D_{16}, and D_{17}, cannot be corrected, since there are double-symbol errors in both $\mathbf{C}_1$ decoding and $\mathbf{C}_2$ decoding at the first stage. In order to correct these four symbols errors, we apply the



(a)



(b)



(c)

(Di) : **Symbol D_i error**

Figure 11.22 *Decoding method.*

erasure correction method to the distance-3 single-symbol error correcting codes, which allows correction of up to double-symbol errors. Therefore, at the first stage, when double-symbol errors are detected in $\mathbf{C}_1$ decoding (i.e., where code $\mathbf{C}_1$ acts as a double-symbol error detecting code), error pointers are set to all symbols included in the positive slopes. At the next stage, $\mathbf{C}_2$ decoding can correct these double-symbol errors indicated by the error pointers. This is because the codes $\mathbf{C}_1$ and $\mathbf{C}_2$ are distance-3 codes, and therefore can correct up to double-symbol erasures.

Codes for Compact Disc (CD) Digital Audio Systems The following two distance-5 RS codes over $GF(2^8)$ are applied to the compact disc (CD) digital audio system [VRIE80]. The codes $\mathbf{C}_1$ and $\mathbf{C}_2$ are as follows:

$\mathbf{C}_1$: (32,28) double-symbol error correcting RS code.
$\mathbf{C}_2$: (28,24) double-symbol error correcting RS code.

In this coding scheme many decoding strategies can be considered. Among these, this system adopts the following decoding method called *super strategy:*

C_1 Decoding

1. All-zero syndrome → error free.
2. Single-symbol errors → single-symbol error correction.
3. Double-symbol errors → double-symbol error correction. Error location pointers are given to the $\mathbf{C}_2$ decoding.
4. More than triple-symbol errors → only detection. Error location pointers are given to the $\mathbf{C}_2$ decoding.

C_2 Decoding

1. All-zero syndrome → error free.
2. Single-symbol errors → single-symbol error correction.
3. Double-symbol errors →
   (a) Number of error pointers from the $\mathbf{C}_1$ decoding is less than or equal to four, and at the same time the error locations of the two error pointers are equal to those calculated from the syndrome → double-symbol error correction.
   (b) Others → only error detection.
4. More than triple-symbol errors → only error detection.

Note that when double-symbol errors are corrected in the $\mathbf{C}_1$ decoding, these two error location pointers are given to the $\mathbf{C}_2$ decoding in order to avoid miscorrection of triple-symbol errors as double-symbol errors in the $\mathbf{C}_1$ decoding.

In case where the $\mathbf{C}_2$ decoder cannot correct, it lets pass through 24 data symbols uncorrected but marked only with error pointers originally given by the $\mathbf{C}_1$ decoder. This

way, even if the $\mathbf{C}_2$ decoder cannot decode, most of the symbols are nevertheless probably error free, and the uncorrected marked sample values can be reconstructed via *linear interpolation* [VRIE80] in digital audio systems. For the marked sample values, this can estimate and interpolate their correct values from both sides of values that have not been marked with error pointers. The CIRC scheme having super strategy is said to have a maximum fully correctable burst length up to 4,000 bits.

Codes for Digital Data Storage (CD-ROM) For digital data storage systems, called *compact disc ROM* (CD-ROM), linear interpolation cannot be applied, and therefore another method for increasing data quality should be added. One system applies both doubly encoded RS SbEC codes and cyclic redundancy check (CRC) code to the data already encoded by CIRC. That is, the CD-ROM employs such a powerful error coding scheme that the original data are encoded by CIRC, and then the resulting data are further encoded by two RS SbEC codes. Thus the data are quadruply encoded as follows. (If CRC is included, they are quintuply encoded.) One disc has about 540-megabyte data capacity.

1. ***CRC code.*** This code is appended in order to check the miscorrection of the RS codes, or to detect uncorrectable errors. The generator polynomial of the CRC code is as follows:

$$\mathbf{g}(x) = (x^{16} + x^{15} + x^2 + 1)(x^{16} + x^2 + x + 1).$$

2. ***Doubly encoded RS codes.*** The codes are determined by the following primitive polynomial $\mathbf{g}'(x)$, and α is a primitive element in $GF(2^8)$ (i.e., a root of $\mathbf{g}'(x)$):

$$\mathbf{g}'(x) = x^8 + x^4 + x^3 + x^2 + 1.$$

   The code includes two RS codes (i.e., (26, 24) RS code $\mathbf{C}_1$, and (45, 43) RS code $\mathbf{C}_2$). Both codes are distance-3 RS codes.

Two check bytes of the code $\mathbf{C}_1$ are generated from the data included in the vertical direction. Also two check bytes of the code $\mathbf{C}_2$ are generated from the user data included in the negative slope that has 24×43 bytes ($= 1,032$ bytes) in a two-dimensional data format. This can be seen in Figure 11.23. The $\mathbf{H}$ matrices of the code $\mathbf{C}_1$ and the code $\mathbf{C}_2$ are expressed as follows:

$$\mathbf{H}_1 = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 & 1 \\ \alpha^{25} & \alpha^{24} & \alpha^{23} & \dots & \alpha & 1 \end{bmatrix},$$

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 & 1 \\ \alpha^{44} & \alpha^{43} & \alpha^{42} & \dots & \alpha & 1 \end{bmatrix}.$$

In Figure 11.23 the code $\mathbf{C}_1$ is generated first, and then the code $\mathbf{C}_2$ is generated. Note that the decoding is permitted for either of the sequences, that is, from $\mathbf{C}_1$ decoding to $\mathbf{C}_2$ decoding, or from $\mathbf{C}_2$ decoding to $\mathbf{C}_1$ decoding. These sequences can be repetitive: for example, $\mathbf{C}_1$ decoding $\rightarrow$ $\mathbf{C}_2$ decoding $\rightarrow$ $\mathbf{C}_1$ decoding $\rightarrow$ $\mathbf{C}_2$ decoding $\rightarrow$ .... Furthermore we can

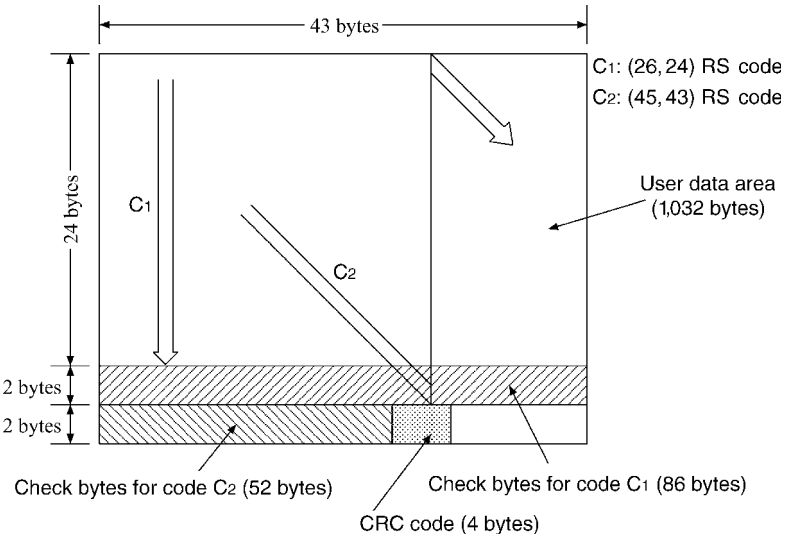


Figure 11.23 *Code design format for CD-ROM.*

make use of the results of CIRC decoding. Each code has the following error correction capability:

1. Single-byte error correction
2. Double-byte erasure correction

Therefore there are many possible decoding strategies. For example, by using the erasure correction capability of the codes shown as item **2** above, the following decoding is applicable:

$$\mathbf{C}_1(\mathbf{2}) \rightarrow \mathbf{C}_2(\mathbf{2}) \rightarrow \mathbf{C}_1(\mathbf{2}) \rightarrow \mathbf{C}_2(\mathbf{2}) \rightarrow \cdots,$$

where $\mathbf{C}_i(\mathbf{2})$ $i = 1, 2$, shows the $\mathbf{C}_i$ decoding with error correction shown as item **2** above. To guard against miscorrection in item 2, the CRC code checks these miscorrections and detects uncorrectable errors. This decoding strategy can attain the same bit-error rate as the commercial magnetic disk or tape units [SAKO85, IMAI90].

11.3.2 Long-Distance Code (LDC)

Another type of optical disk system, the *Write Once Read Many optical disk* (CD-WORM or CD-R), has been popularly applied to the computer mass memories. With these disks, there are two types of error distribution: one is caused by a short burst error (less than 10 bits) with a high rate and the other by a long burst error (10 to 100 bits) with a low rate [SAIT86]. The following strategies have been proposed for correcting these types of errors: one is the large distance ($d \approx 17$) RS code with 120 to 140 bytes code length, interleaved to degree 4 to 10, and another is the product code using two RS codes (with distance 3 to 5), which result in a distance 15 to 25 code, with 30 to 50 bytes code length [YAMA86, KURT87].

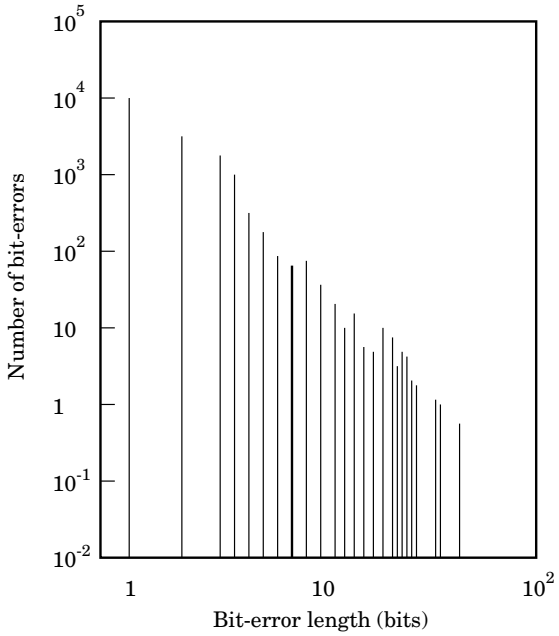


Figure 11.24 *Example of a bit-error length distribution of the magneto-optical disk.* Source: *[SAIT90a]. © 1990 IEICE Japan.*

The recent *erasable optical disks* or *rewritable optical disks*, called CD-RW, which are a magneto-optical recording type, have similar error characteristic, and therefore RS codes with large Hamming distances (i.e., long-distance RS codes) are effectively applied to them as well [FUNK87, ITAO87, KATO87]. An example of the error length histogram for magneto-optical disk is shown in Figure 11.24 [SAIT90a].

A powerful error correcting code has been applied to this type of optical disk, that is, a shortened RS code with a minimum Hamming distance-17 and 10-way byte interleaving, called *long-distance code (LDC)*. The LDC codeword has 104 data bytes (byte = 8 bits) and 16 check bytes, that is, 120 code length in bytes. Figure 11.25 shows the data format of

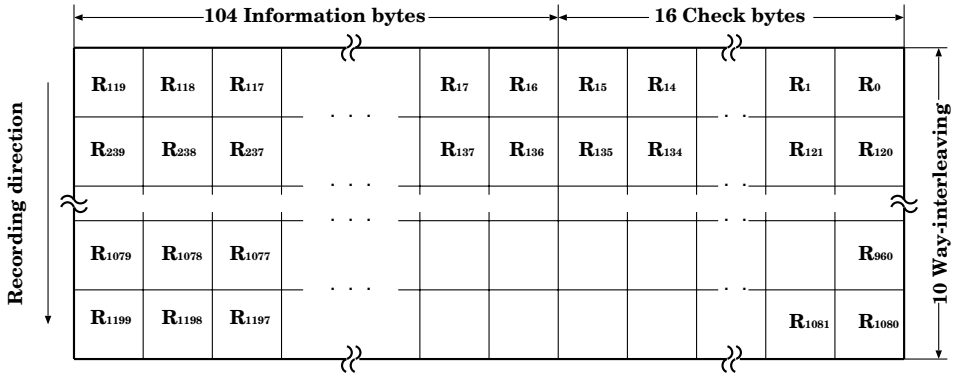


Figure 11.25 *Data format of LDC.*

this code. This code can correct up to consecutive 80 erroneous bytes. The generator polynomial of the code is expressed as

$$\mathbf{g}(x) = \prod_{i=120}^{135} (x + \beta^{88i}),$$

where β is a primitive element in $GF(2^8)$, generated by $\mathbf{p}(x) = x^8 + x^5 + x^3 + x^2 + 1$. This powerful code requires a very complex decoding circuit. It also is required to finish the decoding process within a disk rotation time, that is, to perform real-time error correction. One-chip decoder LSI that satisfies this requirement has been fabricated using a 1.3 μm CMOS process and mounted in 160-pin plastic QFP [YOSH90]. A Euclidean algorithm [SUGI75] is applied to this decoding for computing the error byte locations and error values.

As an effective defect-tolerant technique, alternate data tracks or sectors are used in order to switch the defective tracks or sectors to the spare ones [SAIT88]. An error control strategy mixed with this defect management and powerful error correcting code can achieve a highly reliable optical disk system [SAIT90a].

11.3.3 RS Product Codes for DVDs

Large-capacity optical DVDs (digital versatile discs) with 4.7 giga byte (GB) capacity, such as DVD-ROM (read-only memory), DVD-R (recordable, write-once memory), DVD-RW (rewritable), DVD-RAM, and DVD+RW, are now in wide use for audio-video systems. In this type of optical disk more powerful error control scheme has proved to reduce the error correction redundancy rate to approximately half of that of the CDs.

A DVD error correction scheme using the *Reed-Solomon (RS) product code* with much more powerful error correction than the CIRC in CD has been applied across a large amount of disc data. Figure 11.26 shows the information recorded on a DVD, formatted

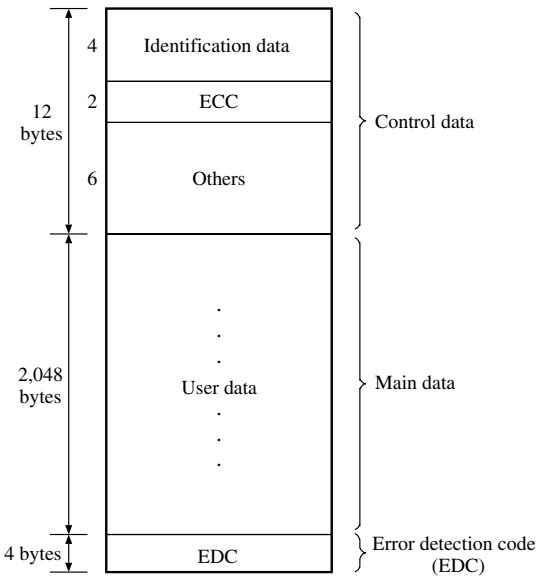


Figure 11.26 *DVD Data Sector with total 2,064 bytes.*

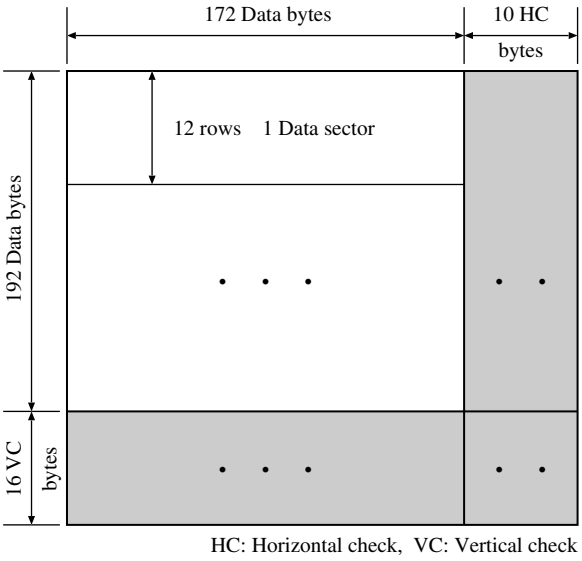


Figure 11.27 *ECC Block with a product code of a (208,192) RS code × a (182,172) RS code.*

into sectors. A sector is the smallest addressable part of the information track. A data sector has 2,064 bytes, consists of main data with 2,048 bytes, control data with 12 bytes, and the error detection code (EDC) with 4 bytes. In order to certify the identification data (ID) in control data, two check bytes (ECC) are appended to correct single byte errors in ID.

Like CD-ROMs, the CRC with 32 bits redundancy is given for the EDC in order to check the miscorrection of the corrected data, or to detect uncorrectable errors. After the EDC calculation over the data sectors, *scrambling* is performed to the 2,048 bytes of main data in the sector in order to randomize the data.

After scrambling the main data in the data sectors, the check information generated by the RS product code is added to each group of 16 data sectors to form an ECC block. This ECC block forms a two-dimensional array of 208 rows by 182 columns in which 16 check bytes for vertical check (VC) are generated by the (208, 192) RS code and then added to each column, and similarly 10 check bytes for horizontal check (HC) are generated by the (182, 172) RS code and then added to each of the 208 rows. These result in a Reed-Solomon (RS) product code with 208 rows (192 data-rows + 16 rows formed by VC) and 182 columns (172 data columns + 10 columns formed by HC). This is shown in Figure 11.27. This RS product code can correct at least 5 bytes in each row and at least 8 bytes errors in each column. That is, at least 5×8 bytes clustered errors can be corrected. By several alternating calculations applied over row and column, much larger erroneous bytes can be corrected or detected.

The rows for VC are interleaved with the data rows in a regular order, which forms a recording sector with (12 data rows + 1 VC row). Hence the interleaved ECC block is divided into 16 recording sectors. This way each recording sector contains the original data with 12 rows × 172 bytes/row + 12 × 10 HC-bytes + 1 row of 182 bytes, that is, a total of 2,366 bytes. This is shown in Figure 11.28. The DVD-ROM, DVD-R, and DVD-RW have the same sector format and ECC block format.

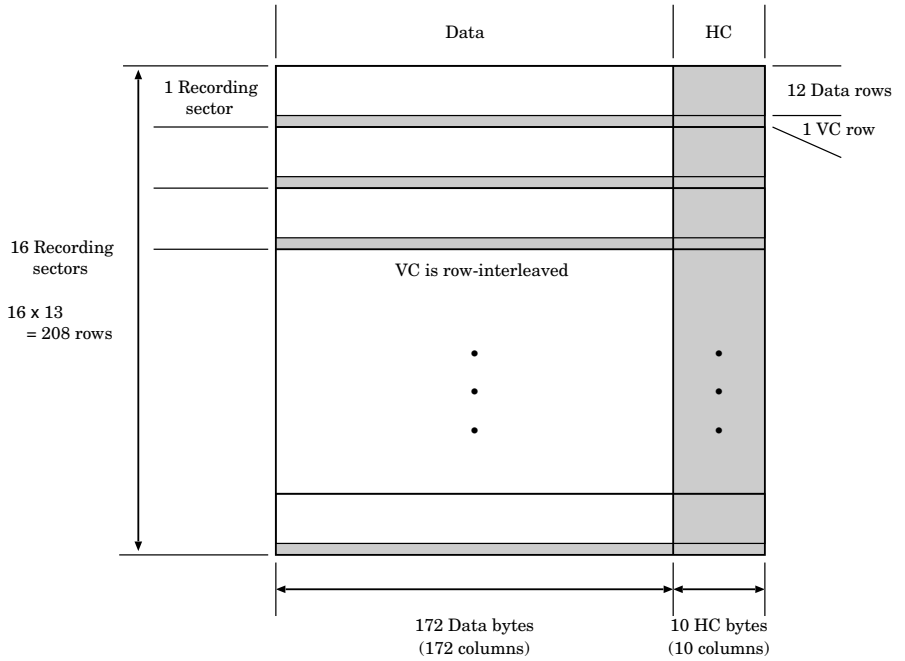


Figure 11.28 *Recording sectors, each with (12 data rows + 1 interleaved VC row).*

A high-speed VLSI architecture for the RS product code decoder has been proposed, and implemented using a Euclidean algorithm or a Berlekamp-Massey algorithm to solve the key equations of the RS code with a large Hamming distance [YOSH90, WILH99, CHAN01, LEE03]. These equations were shown in Subsections 2.3.5 and 2.3.6.

EXERCISES

11.1 For the ORC in the 7-track tape memory unit, do the following:

(a) Design an ORC that consists of five information bytes, B_1, B_2, B_3, B_4 and B_5, and two check bytes, C and P, where a byte has 6 bits. The binary irreducible polynomial $\mathbf{g}(x)$ with degree 6 is shown below.

Coefficients of polynomial $\mathbf{g}(x)$							Exponent
g_0	g_1	g_2	g_3	g_4	g_5	g_6	e
1	1	0	0	0	0	1	63
1	1	1	0	1	0	1	21
1	1	1	0	0	1	1	63
1	0	0	1	0	0	1	9
1	0	1	1	0	1	1	63

(b) Design the LFSR encoder that produces check bytes P and C. The state of the LFSR is changed by a clock pulse. Next, for the following five information bytes

$$B_1 = (\ 0\ 1\ 1\ 0\ 1\ 0\ ),$$
$$B_2 = (\ 1\ 0\ 1\ 1\ 0\ 1\ ),$$
$$B_3 = (\ 1\ 1\ 1\ 0\ 0\ 0\ ),$$
$$B_4 = (\ 0\ 0\ 1\ 1\ 1\ 0\ ),$$
$$B_5 = (\ 1\ 1\ 0\ 1\ 1\ 1\ ),$$

give the state sequence of the LFSR encoder clock by clock in which information bytes $B_5,\ B_4,\ B_3,\ B_2,\ B_1$ are entered in this order to the LFSR encoder, and finally check byte C is produced.

(c) In the five information bytes above, assume that the fourth track data Z_3 has an error $E = (\ 0\ 1\ 1\ 0\ 1\ 1\ )$. Give the state sequence of the LFSR decoder clock by clock in which the LFSR decoder corrects the error E with forward shift of the LFSR as well as with backward shift of the LFSR. Show that, in general, the backward shift of the LFSR decoder has high possibility to decode faster than the forward shift.

11.2 In an AXP code, consider the case where set A is being corrected for errors in a known erroneous track, and another unknown track in set A begins to be affected by errors. Show that these erroneous tracks of set A can be corrected, provided that set B has at most one known erroneous track.

11.3 Show that the AXP code can correct track errors of up to one unknown erroneous track in one set and up to one unknown or two known erroneous tracks in the other set.

11.4 Show that the AXP code can correct track errors of up to two known erroneous tracks in one set and up to one unknown or two known erroneous tracks in the other set.

11.5 Design the encoding circuit of the MSS code shown in Eq. (11.25) with

$$\mathbf{T}^{\lambda} = \mathbf{T}^{68} = \begin{bmatrix} 00001000 \\ 10001100 \\ 01000110 \\ 00101011 \\ 10010101 \\ 01000010 \\ 00100001 \\ 00010000 \end{bmatrix}.$$

11.6 Prove that the code adopted in MSS, shown in Eq. (11.25), has the following characteristics:

(a) If the code is used for correction of single-byte errors, then it does not miscorrect any combination of double-bit errors.

(b) If the code is used for correction of single-bit errors, then it does not miscorrect any combination of a single-byte error with single-bit error in another byte.

11.7 For the Fire code generated by $\mathbf{g}(x) = (x^9 + 1)(x^5 + x^2 + 1)$, do the following:

(a) Find the code length of this code.

(b) Find the burst error correction length of this code.

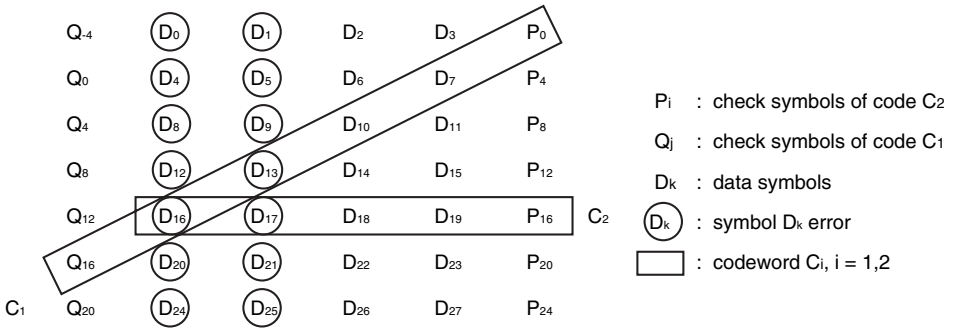
(c) Suppose that an error burst $\mathbf{b}(x) = x^6 + x^5 + x^4 + x^3 + x^2$ has occurred. Show the decoding procedure of this case using a high-speed decoding method.

(d) Design the high-speed decoder of this code.

11.8 Design the high-speed decoder for the code generated by $\mathbf{g}(x) = (x^{11} + 1)(x^3 + x + 1)(x^4 + x + 1)$. Determine the error detection and correction capability of this code.

11.9 Discuss the decoding procedure of the two-level coding in IBM 3390J and 3380K disk systems, shown in Figure 11.19.

11.10 In the CIRC systems, assume that half of the data information shown below is in errors. Show how to correct these errors using the codes $\mathbf{C}_1$ and $\mathbf{C}_2$, each having a minimum Hamming distance $d_{min} = 3$.



11.11 Determine the error correction capability of the CIRC in which the codes $\mathbf{C}_1$ and $\mathbf{C}_2$ both have minimum Hamming distance $d_{min} = 2$.

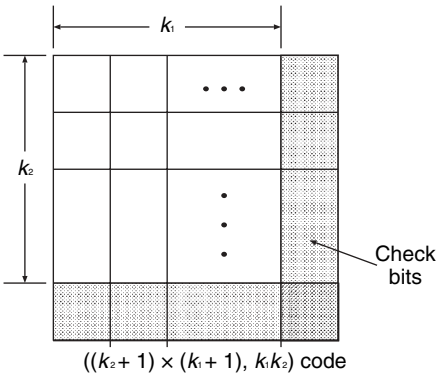
11.12 For the two-dimensional code, do the following:

(a) Show that the following $(20, 12)$ two-dimensional code with parity-check bits 16, 13, 10, 11, 15, 19, 3, 7 can correct any burst error of length up to 3 bits in readout data if the data bits are read diagonally instead of horizontally (i.e., readout order: $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 16$). The horizontal syndrome is represented by a vector $(h_0\ h_1\ h_2\ h_3\ h_4)$, where h_i is modulo-2 sum of all the received bits in row *i*. Similarly the vertical syndrome is represented by a vector $(v_0\ v_1\ v_2\ v_3)$, where v_j is modulo-2 sum of the received bits in column *j*.

0	17	14	11	h_0
4	1	18	15	h_1
8	5	2	19	h_2
12	9	6	3	h_3
16	13	10	7	h_4
v_0	v_1	v_2	v_3	

(5 × 4, 4 × 3) code

(b) Prove that, in general, the $(k_2 + 1) \times (k_1 + 1)$ two-dimensional code, in which the last row and the last column contain parity bits, can correct any burst error of length up to k_1 if and only if the data bits are read diagonally and $k_2 \geq 2(k_1 - 1)$.



11.13 Discuss error correction capability and redundancy of the RS product codes in DVDs as compared to the CIRC and the LDC in CDs.

REFERENCES

[ADI84] W. Adi, "Fast Burst Error-Correction with Fire Code," *IEEE Trans. Comput.*, C-33 (July 1984): 613–618.

[AVIZ83] A. Avizienis, "Two-Dimensional Low-Cost Arithmetic Error Codes," *Dig., 6th IEEE Int. Symp. on Comput. Arithmetic* (June 1983): 169–172.

[BLAH83] R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley (1983).

[BLAU85] M. Blaum and R. L. McEliece, "Coding Protecting for Magnetic Tapes: A Generalization of the Patel-Hong Code," *IEEE Trans. Info. Theory*, IT-31 (September 1985): 690–693.

[BLAU86] M. Blaum, P. G. Farrell, and H. C. A. van Tilbory, "A Class of Burst Error-Correcting Array Codes," *IEEE Trans. Info. Theory*, IT-32 (November 1986): 836–839.

[BLAU95] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures," *IEEE Trans. Comput.*, 44 (February 1995): 192–202.

[BLAU96] M. Blaum, J. Bruck, and A. Vardy, "MDS Array Codes with Independent Parity Symbols," *IEEE Trans. Info. Theory*, 42 (March 1996): 529–542.

[BLAU99] M. Blaum and R.M. Roth, ''On Lowest Density MDS Codes,'' *IEEE Trans. Info. Theory*, 45 (January 1999): 46–59.

[BOSE84b] B. Bose and T. R. N. Rao, ''Unidirectional Error Codes for Shift-Register Memories,'' *IEEE Trans. Comput.*, C-33 (June 1984): 575–578.

[BOSE84c] B. Bose, ''Two Dimensional ARC Codes,'' *Dig., 14th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1984): 324–329.

[BOSS72] D. C. Bossen, R. A. Henle, M. Y. Hsiao, G. A. Maley, and W. D. Pricer, ''System for Expanded Detection and Correction of Errors in Parallel Binary Data Produced by Data Tracks,'' US Patent 3675200 (July 4, 1972).

[BROW70] D. T. Brown and F. F. Sellers, ''Error Correction for IBM 800-Bit-per-Inch Magnetic Tape,'' *IBM J. Res. Dev.*, 14 (July 1970): 384–389.

[CHAN01] H. -C. Chang, C. B. Shung, and C. -Y. Lee, ''A Reed-Solomon Product-Code (RS-PC) Decoder Chip for DVD Applications,'' *IEEE J. Solid-State Circ.*, 36 (February 2001): 229–238.

[CHEN94] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, ''RAID: High-Performance, Reliable Secondary Storage,'' *ACM Comput. Surveys*, 26 (June 1994): 145–185.

[CHIE69] R. T. Chien, ''Burst-Correcting Codes with High-Speed Decoding,'' *IEEE Trans. Info. Theory*, IT-15 (January 1969): 109–113.

[DOI79] T. Doi, K. Odaka, G. Fukuda, and S. Furukawa, ''Cross Interleave Code for Error Correction of Digital Audio Systems,'' *Proc. 64th AES Convention* (November 1979).

[FIRE59] P. Fire, ''A Class of Multiple-Error-Correcting Binary Codes for Non-independent Errors,'' *Sylvania Report*, RSL-E-2, Sylvania Electronic Defense Laboratory, Reconnaissance, Systems Division (1959).

[FUJA89] T. Fuja, C. Heegard, and M. Blaum, ''Cross Parity Check Convolutional Codes,'' *IEEE Trans. Info. Theory*, 35 (November 1989): 1264–1276.

[FUNK87] A. W. Funkenbusch, T. Rinehart, D. W. Siitrari, Y. S. Hwang, and R. N. Gardner, ''Magneto-Optics Technology for Mass Storage Systems,'' *Dig., 8th IEEE Symp. on Mass Storage Systems* (May 1987): 101–106.

[GIBS92] G. A. Gibson, *Redundant Disk Arrays*, MIT Press (1992).

[GOTO80] M. Goto, ''Rates of Unidirectional 2-Column Errors Detectable by Arithmetic Codes,'' *Dig., 10th IEEE Int. Symp. on Fault-Tolerant Computing* (October 1980): 21–25.

[GLOV91] N. Glover and T. Dudley, *Practical Error Correction Design for Engineers*, rev. 2d ed., Cirrus Logic (1991).

[HELL94] L. Hellerstein, G. A. Gibson, R. M. Karp, R. H. Katz, and D. A. Patterson, ''Coding Techniques for Handling Failures in Large Disk Arrays,'' *Algorithmica*, 12 (1994): 182–208.

[HODG80] P. Hodges, W. J. Schaeuble, and P. L. Shaffer, ''Error Correcting System for Serial by Byte Data,'' US Patent 4185269 (January 22, 1980).

[HOWE84] T. D. Howell, ''Analysis of Correctable Errors in the IBM3380 Disk File,'' *IBM J. Res. Dev.*, 28 (March 1984): 206–211.

[HSIA81] M. Y. Hsiao, W. C. Carter, J. W. Thomas, and W. R. Stringfellow, ''Reliability, Availability and Serviceability of IBM Computer System: A Quarter Century of Progress,'' *IBM J. Res. Dev.*, 25 (September 1981): 453–465.

[IMAI90] H. Imai (ed.), *Essentials of Error-Correcting Coding Techniques*, Academic Press (1990), chs. 7, 8.

[INOU78] T. Inoue, Y. Sugiyama, K. Ohnishi, T. Kanai, and K. Tanaka, ''A New Class of Burst-Error-Correcting Codes and Its Application to PCM Tape Recording Systems,'' *Proc. Nat. Telecommun. Conf.* (1978): 20.6.1–20.6.5.

[ITAO85] K. Itao and S. Hosakawa, ''An Automated Mass Storage System with Magnetic Tape Cartridge,'' *Dig., 7th IEEE Symp. on Mass Storage Systems* (November 1985): 87–90.

[ITAO87] K. Itao, A. Yamaji, S. Hara, and N. Izawa, "Magneto-Optical Mass Storage System with 130 mm Write-Once Disk Compatibility," *Dig., 8th IEEE Symp. on Mass Storage Systems* (May 1987): 92–97.

[KATO87] O. Kato, K. Kurosawa, K. Iwakuni, and M. Shimbo, "Development of Error Correction Method and LSI for Optical Disk Data Storage System," *Paper of Technical Group on Computer Architecture*, IPS Japan (September 1987): 67–73.

[KURT87] C. Kurtz, "Development of a High-Capacity Performance Optical Storage System," *Dig., 8th IEEE Symp. on Mass Storage Systems* (May 1987): 107–111.

[LEE93] L. -W. Lee, J. -F. Wang, J. -Y. Lee, and J. -D. Shie, "Dynamic Search-Window Adjustment and Interlaced Search for Block-Matching Algorithm," *IEEE Trans. Circ. Syst. Video Technol.*, 3 (February 1993): 85–87.

[LEE03] H. Lee, "High-Speed VLSI Architecture for Parallel Reed-Solomon Decoder," *IEEE Trans. VLSI Syst.*, 11, (April 2003): 288–294.

[LEIS84] E. L. Leis, "Data Integrity in Digital Optical disks," *IEEE Trans. Comput.*, C-33 (September 1984): 818–827.

[LIN83] S. Lin and D. J. Costello, Jr., Chap. 16.2 in *Error Control Coding: Fundamentals and Applications*, Prentice Hall (1983).

[PATE74] A. M. Patel and S. J. Hong, "Optimal Rectangular Code for High Density Magnetic Tapes," *IBM J. Res. Dev.*, 18 (November 1974): 579–588.

[PATE80] A. M. Patel, "Error Recovery Scheme for the IBM3850 Mass Storage System," *IBM J. Res. Dev.*, 24 (January 1980): 32–42.

[PATE85] A. M. Patel, "Adaptive Cross-Parity (AXP) Code for a High-Density Magnetic Tape Subsystem," *IBM J. Res. Dev.*, 29 (November 1985): 546–562.

[PATE86] A. M. Patel, "On-the-Fly Decoder for Multiple Byte Errors," *IBM J. Res. Dev.*, 30 (May 1986): 259–269.

[PATE89] A. M. Patel, "Two-Level Coding for Error Control in Magnetic Disk Storage Products," *IBM J. Res. Dev.*, 33 (July 1989): 470–484,

[PATT88] D. A. Patterson, G. A. Gibson, and R. H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proc. Int. Conf. on Management of Data (SIGMOD)*, ACM (1998): 109–116.

[REED60] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *SIAM J. Appl. Math.*, 8 (June 1960): 300–304.

[ROTH98] R. Roth and G. Seraussi, "Reduced-Redundancy Product Codes for Burst Error Correction," *IEEE Trans. Info. Theory*, 44 (July 1998): 1395–1406.

[SAIT86] M. Saito, T. Takeda, and M. Nunotani, "An Evaluation of Error Correcting Codes for Optical Disks" (in Japanese), *Paper of Technical Group*, *IECE Japan* IT86–48 (1986).

[SAIT88] M. Saito and T. Takeda, "Optical Disk Redundancy Design Considering Bit-Error Characteristics" (in Japanese), *Trans. IEICE Japan*, J71-C (February 1988): 287–295.

[SAIT90a] M. Saito, T. Takeda, and K. Itao, "Optimum Error Control Strategy for Optical Micro-Disk Subsystems," *Trans. IEICE Japan*, E73 (May 1990): 712–717.

[SAIT90b] M. Saito, "Optical Disk Reliability Estimation Considering Error Control Strategy" (in Japanese), *Trans. IEICE Japan*, J73-C-II (February 1990): 112–118.

[SAKO85] Y. Sako, T. Suzuki, T. Furuya, and S. Furukawa, "Data Quality of CD-ROM" (in Japanese), *Paper of Technical Group*, *IECE Japan*, IT85-31 (1985).

[SUGI75] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A Method for Solving Key Equation for Decoding Goppa Codes," *Info. Contr.*, 27 (1975): 87–95.

[TANA80] K. Tanaka, M. Ozaki, T. Inoue, and T. Yamaguchi, "On a Tape Format for Reliable PCM Multi-Channel Tape Recorders," *Proc. 66th AES Convention* (May 1980): 1669.

[TSUN83] Y. Tsunoda, M. Miyazaki, and S. Abe, "Large Capacity Optical Disk File Unit" (in Japanese), *Nikkei Electronics* (November 21, 1983): 189–213.

[VRIE80] L. B. Vores, K. A. Imink, J. G. Nibor, H. Hoeve, T. Doi, K. Okada, and H. Ogawa, "The Compact Disc Digital Audio System-Modulation and Error Correction," *Proc. 67th AES Convention* (October 1980): 1674 (H-8).

[YAMA86] A. Yamagishi, "Encoder and Decoder LSIs for BCH Code and RS Code" (in Japanese), *Proc. 1986 IEEE Workshop on Coding Theory and Its Applications*, WCTA86-2 (August 1986).

[WEST] Western Digital, *WD1100-06 ECC/CRC Logic, Data Catalog*, Western Digital Corp.

[WILH99] W. Wilhelm, "A New Scalable VLSI Architecture for Reed-Solomon Decoders," *IEEE J. Solid-State Circ.*, 34 (March 1999): 388–396.

[XU99] L. Xu and J. Bruck, "X-Code: MDS Array Codes with Optimal Encoding," *IEEE Trans. Info. Theory*, 45 (January 1999): 272–276.

[YAMA91] I. Yamada, M. Saito, A. Watabe, and K. Itao, "Automated Optical Mass Storage Systems with 3-Bean Magneto-Optical Disk Drives," *Dig., 11th IEEE Symp. on Mass Storage Systems* (October 1991): 149–154.

[YOSH90] H. Yoshida, A. Yamagishi, T. Inoue, and K. Tanaka, "Erasures and Error Correction LSI for Optical Disk Systems" (in Japanese), *Trans. IEICE Japan*, J73-A (February 1990): 261–268.

[ZEMO91] G. Zmor and G. D. Cohen, "Error-Correcting WOM-Codes," *IEEE Trans. Info. Theory*, 37 (May 1991): 730–734.

CONTENTS

12

Coding for Logic and System Design

A number of techniques exist nowadays for improving computer reliability and availability [ANDE81, SIEW82]. The general technique of *standby sparing* seems to have found much acceptance in logic and system design. Standby sparing requires a modular design in which several identical modules of each type are present, some being used actively to perform the computing function and the others waiting to be switched in when one of the active modules fails. An ultrareliable computer, the *JPL self-testing and repairing (STAR) computer* [AVIZ71], for example, makes extensive use of the techniques of modularity and standby sparing.

The fundamental task of standby sparing is to detect and localize a malfunction so that restructuring of the computer can take place by switching the faulty module out of service. Thus an important part of the design of a highly reliable computer utilizing the standby sparing technique is an efficient and complete method of error detection and fault location [CART71a]. Errors caused by faults may be detected by hardware check circuits (i.e., *checkers*). The recently devised diagnostic approach for fault isolation in logic systems also uses checkers extensively to capture errors, interpret their syndromes, and locate the faulty positions [BOSS82]. A circuit whose faults or malfunctions are always checked by itself is said to be *self-checking* [WAKE78].

Self-checking circuits offer a number of advantages, the most obvious of which is the *immediate detection of errors* during online operation. Another is the capability of detecting errors caused by *transient faults*. Further self-checking circuits are becoming more attractive with the advances in VLSI.

Self-checking circuits rely on redundancy to detect errors. That is, during error-free operation the circuit outputs cannot assume all possible states. This redundancy by a check circuit is to determine if they form a proper codeword; if they do not, an error is presumed. Hence the theory of error detecting codes is important in the design of these circuits.

Code Design for Dependable Systems: Theory and Practical Applications, by Eiji Fujiwara

Some error detecting codes, such as *simple parity-check codes, checksum codes, low-cost residue codes*, and some unidirectional error detecting codes (e.g., *m-out-of-n* codes and *Berger codes*), have been studied for application. These applications are for logic circuits, such as the *arithmetic logic unit (ALU)*, the *field programmable gate array (FPGA)*, and some kinds of decoding circuits. Considerable literature has appeared on the implementation of these checking circuits. Checkers can no longer be assumed to be error free in computer systems because check circuits are constructed from the same components as the circuits that perform the computer operations and hence are subject to the same types of faults. Clearly, a good design is important to have an effective self-checking checker.

This chapter discusses the self-checking concept and how the error detecting or correcting codes are applied to some logic circuits and systems. An implementation method for a self-checking checker is also discussed. That is, this chapter demonstrates how these checkers are made self-testing—how they can be designed so that if there are faults in the checker, the checker itself detects them. In such self-testing checkers we have to tend to the input space of the checker because the checker is generally an embedded circuit and not all input codewords are given to the checker. This chapter also provides some examples of self-checking ALUs and self-checking computers in which self-testing checkers are extensively used. Again, it should be noted that in today's microprocessors, some types of error detecting / correcting codes are embedded on chips, as is the case extensively with on-chip ECCs. This chapter also briefly covers these innovations.

12.1 SELF-CHECKING CONCEPT

In order to detect transient faults, as well as stuck faults in logic circuits, many kinds of checkers, such as *parity code checkers, duplication checkers, parity-prediction checkers*, and *residue code checkers*, have already been applied in an adaptive manner to some functional circuits (e.g., adders, multipliers, decoding circuits, and data path circuits [SELL68]). Further the theory of self-checking circuits (e.g., *totally self-checking (TSC) circuits, strongly fault-secure (SFS)* circuits, and *error-secure (ES)* circuit) has been studied extensively [CART68, CART71b, ANDE71, WAKE74, SMIT78, NICO84, NANY88]. Some practical design methods for self-checking circuits appear in [CART77, SMIT83, NANY88].

Here we study both the general concept of the self-checking circuits and the checker concept for the self-checking checkers.

12.1.1 General Concept

Basically the logic circuits should be designed such that *they indicate any malfunction during normal operation and not produce an erroneous result without an error indication*. In some actual circuits any fault from a specified set of faults can cause a detectable erroneous output without also producing an error signal [CART68, ANDE71]. The general structure of the self-checking circuits is shown in Figure 12.1; the *error indicator* output Z must be designed to produce an error signal for some normal circuit input whenever a fault from a specified set of faults occurs within the circuit.

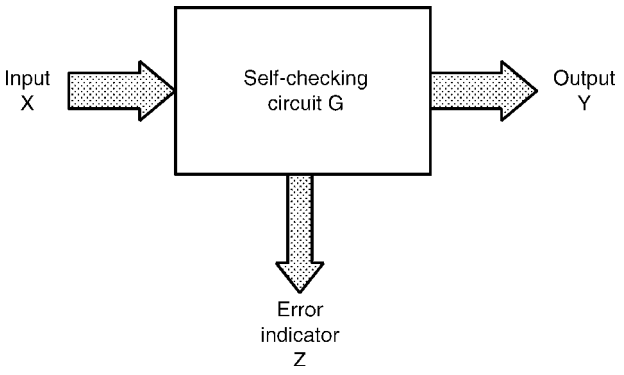


Figure 12.1 *Self-checking circuit.*

In the design of the circuits in Figure 12.1, the circuit **G** is considered to be divided into two connected circuit blocks, the functional circuit block **L** and the check circuit block **CK**. These circuit blocks are shown in Figure 12.2.

First we discuss self-checking functional circuits and then extend that to self-checking functional networks. Check circuits will be covered in a subsequent section.

We begin with a combinational circuit **L** that produces an output vector $Y(X, f)$ that is a function of the input vector X and a fault $f \in \mathbf{F}$, which is a specified set of faults in the circuit. The absence of a fault is called a *null fault* and is denoted by λ. If the circuit **L** has n inputs, then the *input space* Ω_x of **L** is the set of all 2^n input vectors. Similarly, if **L** has r outputs, then the set of all 2^r output vectors is the *output space* Ω_y of **L**. In general, logic circuits will receive only a subset of their input space during "normal" (i.e., fault-free) operation; this subset is called the *input codespace* **N**. Then $\Omega_x - \mathbf{N}$ is the *input noncodewords*. Here we consider only circuits whose outputs under normal operation are codewords, and therefore we also define for each circuit its *output codespace* **S**. Then

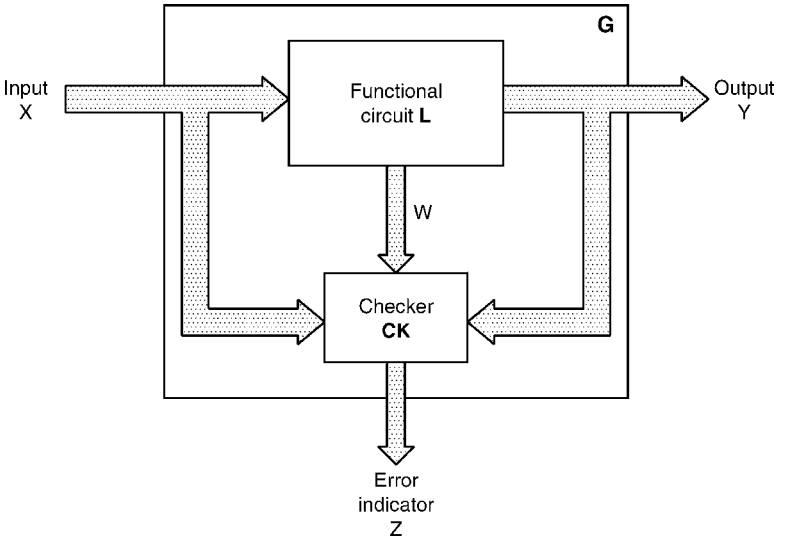


Figure 12.2 *Self-checking circuit model.*

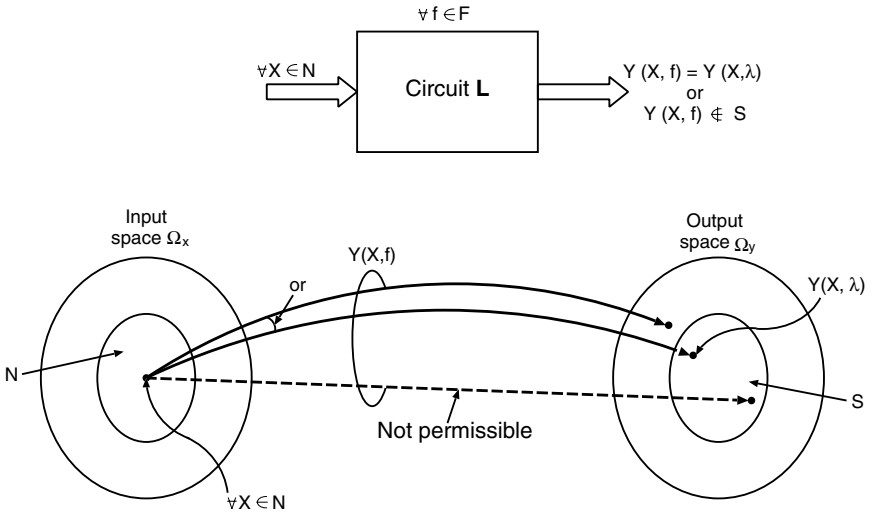


Figure 12.3 *Fault-secure circuit.*

$\Omega_y - \mathbf{S}$ is the *output noncodespace*. For a given input $X \in \mathbf{N}$, a fault may or may not cause an error in the output. If the fault does cause an error, it may change the output, either to a noncodeword (i.e., a *detectable error*), or to a different codeword (i.e., an *undetectable error*).

1. Fault Secure and Self-testing

Definition 12.1 A circuit **L** is *fault secure (FS)* for an input set **N** and a fault set **F** if for input X in **N** and for any fault f in **F**, $Y(X, f) = Y(X, \lambda)$, or $Y(X, f) \notin \mathbf{S}$. □

That is, the output of a fault-secure circuit is always either the correct codeword or a noncodeword but never a wrong codeword for any fault in **F**. This is illustrated in Figure 12.3.

Definition 12.2 A circuit **L** is *self-testing (ST)* for an input set **N** and a fault set **F** if for every fault f in **F** there is some input X in **N** such that $Y(X, f)$ is not in **S**. □

Figure 12.4 illustrates this definition. By this definition, an input X for which $Y(X, f)$ is not in **S**, or $Y(X, f) \neq Y(X, \lambda)$, is called *a test pattern* for f. If each input in **N** occurs during the normal operations of the circuit, then self-testing guarantees that all faults in **F** produce detectable errors during normal operation.

The properties of self-testing and fault secureness are further illustrated in Figure 12.5. Shown in the figure are also the set of all faults and its subset **F**, the input space Ωy and its subset **S**. For example, **F** shows the set of single-stuck faults, or the set of unidirectional faults. The outside of **F** shows all possible faults excluding the faults in **F**. For input vectors, **N** shows the set of normal *input codewords*, and the outside of **N** shows the set of *input noncodewords*. For output vectors, **S** shows the set of *output codewords*, and outside of **S** shows the set of *output noncodewords*.

In the absence of faults, inputs from **N** produce outputs in **S**, as indicated by the translation $Y(X, \lambda)$ for various X. Self-testing is demonstrated by the existence of a test

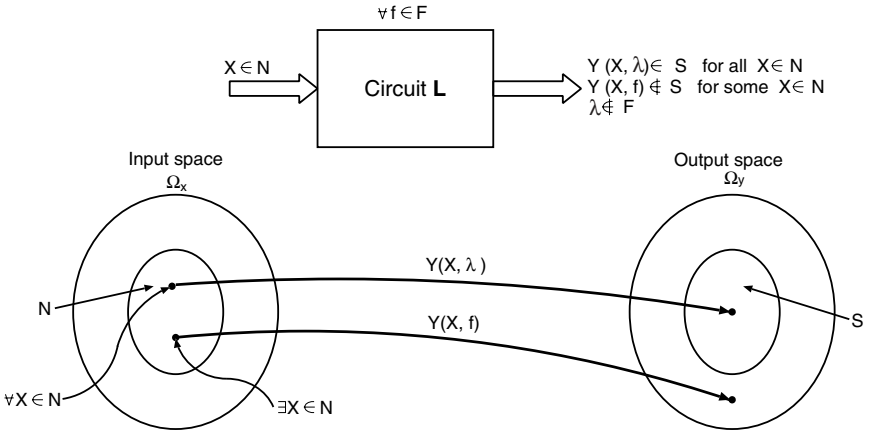


Figure 12.4 *Self-testing circuit.*

pattern in **N** for each of the faults f_1 and f_2 in **F** (X_2 for f_1, and X_1 and X_2 for f_2 are test patterns). The fault-secure property is illustrated by the transition of $Y(X_1, f)$ for the various faults f. In the presence of a fault from **F**, the output is either correct (e.g., $Y(X_1, f_1)$), or it is a noncodeword (e.g., $Y(X_1, f_2)$). However, faults outside of **F** may produce erroneous codeword outputs (e.g., $Y(X_2, f_3)$) that are undetectable.

Definition 12.3 A circuit **L** is *totally self-checking (TSC)* if it is both self-testing and fault secure. □

Self-testing is necessary in addition to fault secureness because even if the circuit output is always a correct codeword for a single fault in the fault-secure circuit, the second fault may appear after some time has elapsed, and then this circuit may not detect these two faults.

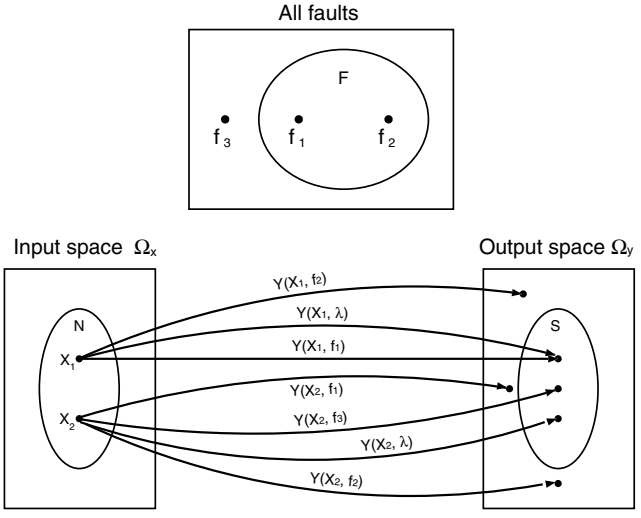


Figure 12.5 *Examples of self-testing and fault secure properties [WAKE78].*

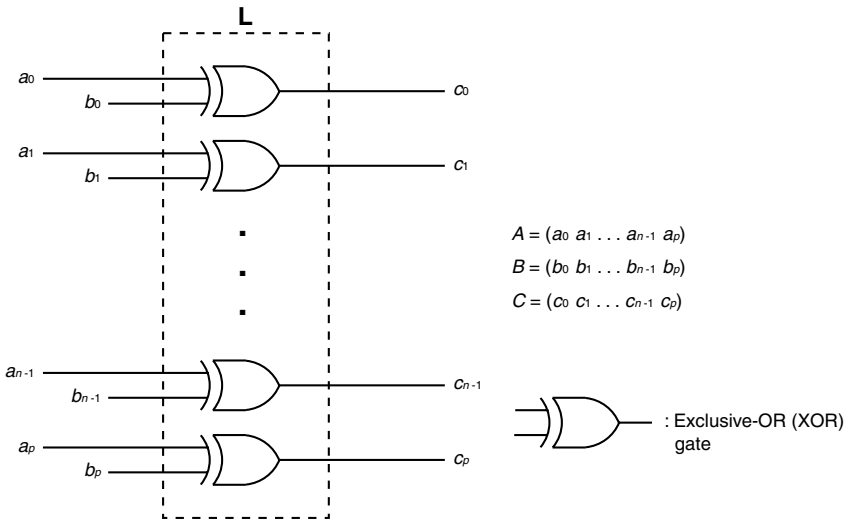


Figure 12.6 *Self-checking exclusive-OR circuit.*

Example 12.1

Figure 12.6 shows a circuit that computes in parallel the exclusive-OR (XOR) of two input vectors, $A = (a_0\ a_1\ \ldots\ a_{n-1}\ a_p)$ and $B = (b_0\ b_1\ \ldots\ b_{n-1}\ b_p)$, where a_p and b_p are even-parity bits. It is apparent that XOR of any two even-parity codewords is also an even-parity codeword. That is, an output vector $C = (c_0\ c_1\ \ldots\ c_{n-1}\ c_p)$, where c_p is a parity bit, is also an even-parity codeword. This property is called *code preserving*, and the code is said to be *preserved* under this operation. We now consider F to be the set of single faults. Input codespace **N** is defined as the set in which both A and B are even-parity codewords. Output codespace **S** is defined as the set of even-parity codewords (shown in Figure 12.7).

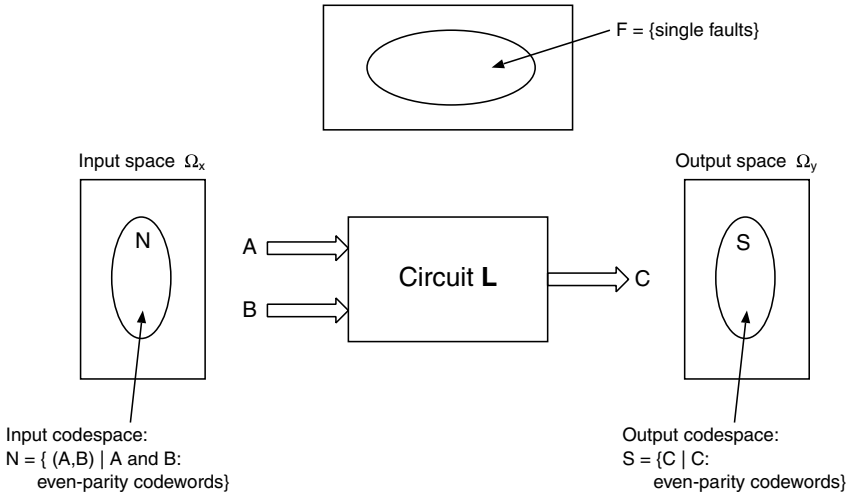


Figure 12.7 *Input and output codespaces for the example circuit of Figure 12.6.*

TABLE 12.1 Test Patterns (a_i, b_i) for Single Faults

Input single fault		Output single fault	
Stuck-at '0'	Stuck-at '1'	Stuck-at '0'	Stuck-at '1'
(1,1)	(0,0)	(1,0) or (0,1)	(0,0) or (1,1)

The even-parity code is a distance-2 linear code, and the circuit of Figure 12.6 is code preserving. This circuit is also *fault secure* for all single faults, since a single fault produces at most a distance-1 change in the output. That is, single faults produce single errors in the output, or produce masked output (i.e., correct output). Therefore the output always belongs to noncodespace, or correct output. This satisfies the definition of fault secureness. As for the self-testing property, we can easily find out test patterns for each single fault (see Table 12.1). For example, for a single-stuck fault input in an XOR gate (e.g., stuck-at '0' fault in a_i or b_i input), the codeword inputs A and B that include $(a_i, b_i) = (1, 1)$ are the test pattern. Clearly, the circuit shown in Figure 12.6 is self-testing and fault secure, and hence is TSC for all codeword inputs under single faults.

From the standpoint of reliability, TSC is ideal. However, in general, self-testing is a rather difficult condition to satisfy perfectly, as compared to fault secureness. This is because there are sometimes not enough inputs **N** to detect every fault in **F**. With this difficulty in mind, another self-checking concept is proposed in [SMIT78]. It has to do with a *fault sequence* $\langle f_1, f_2, \ldots, f_n \rangle$, where $f_i \in \mathbf{F}$, $1 \leq i \leq n$, is defined to represent the event where fault f_1 occurs, followed later by f_2 (at which point both f_1 and f_2 are present in the circuit), and so forth, until f_n occurs. It is assumed that once a line becomes stuck-at '0' or '1', it remains stuck-at that value; also assumed is that faults occur one at a time and that between any two fault occurrences there is a time interval sufficient for all input code combinations to be applied to the circuit.

The following definitions are based on [SMIT78].

Definition 12.4 Assume that circuit **L** always gives correct codewords for a sequence of fewer than m faults (accumulated $m-1$ faults), $2 \leq m \leq n$, in **F**, and for all $X \in \mathbf{N}$. That is,

$$Y(X, \langle f_1, f_2, \ldots, f_{m-1} \rangle) = Y(X, \lambda).$$

Also assume that for the m-fault sequence $\langle f_1, f_2, \ldots, f_{m-1}, f_m \rangle$ there is some $X \in \mathbf{N}$ such that

$$Y(X, \langle f_1, f_2, \ldots, f_m \rangle) \notin \mathbf{S}.$$

Then **L** is said to be *strong fault secure (SFS)* for $\langle f_1, f_2, \ldots, f_m \rangle$. □

Definition 12.5 A circuit **L** is *strongly fault secure (SFS)* for $\mathbf{F} = \{f_1, f_2, \ldots, f_m\}$ if it is SFS for all sequences of faults in **F**. □

Figure 12.8 illustrates the concept of SFS.

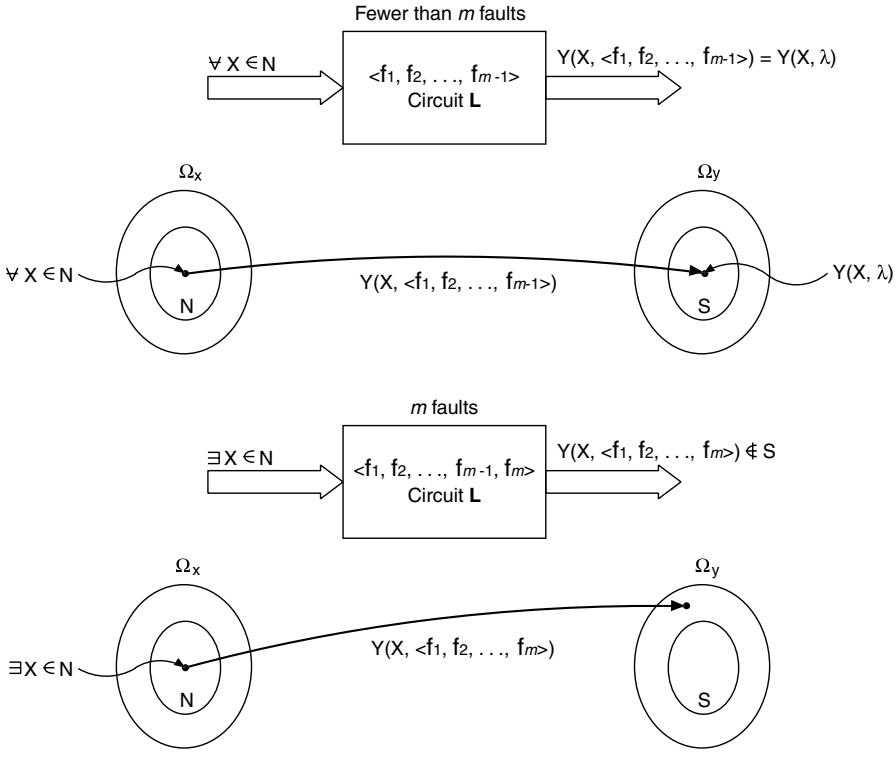


Figure 12.8 *SFS circuit for* $\langle f_1, f_2, \ldots, f_{m-1}, f_m \rangle$.

Example 12.2

For the circuit **L** given in Figure 12.9, let faults in **F** occur in the following sequence:

f_1: stuck-at '1' fault at one input of the AND_1 gate.
f_2: stuck-at '0' fault at the input of the NOT gate.
f_3: stuck-at '1' fault at one input of the AND_2 gate.
f_4: stuck-at '1' fault at the output of the AND_1 gate.

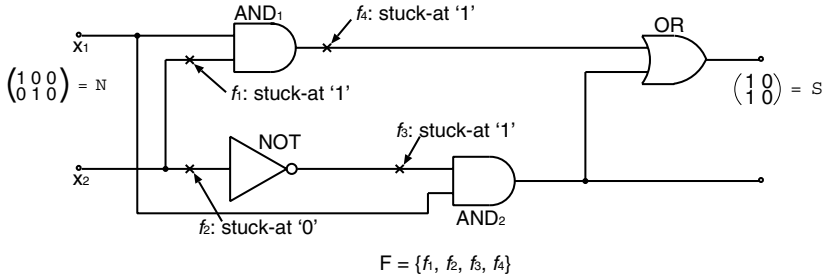


Figure 12.9 *Example SFS circuit for* $\langle f_1, f_2, f_3, f_4 \rangle$.

In this case, circuit **L** is fault secure for input X, $X \in N$,

$$\mathbf{N} = \left\{ \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right\} = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\},$$

and for the fault sequence $\langle f_1, f_2, f_3 \rangle$, since

$$Y(X, \langle f_1, f_2, f_3 \rangle) = Y(X, \lambda) \qquad \text{for all } X \in \mathbf{N}.$$

That is, circuit **L** always gives correct output regardless of the fault sequences: $\langle f_1 \rangle$, $\langle f_1, f_2 \rangle$, and $\langle f_1, f_2, f_3 \rangle$.

Next, let the stuck-at '1' fault f_4 occur at the output of the AND_1 gate. For $X_1 \in \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$, the circuit manifests itself as

$$Y(X_1, \langle f_1, f_2, f_3, f_4 \rangle) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \notin \mathbf{S}.$$

Therefore the circuit **L** is SFS for the fault sequence $\langle f_1, f_2, f_3, f_4 \rangle$.

It can be shown that any SFS circuit satisfies TSC conditions. Furthermore, if a circuit is not SFS, it is always possible to produce an erroneous code output prior to a noncode output. Hence SFS circuits form the largest class of circuits to satisfy TSC conditions.

2. Error Secure and Error Preserving

Note that both fault secureness and self-testing specify only the behavior of the circuit for codeword inputs. That is, inputs are always assumed to be error free. For some circuits, such as self-checking checkers, we are also interested in the behavior of the circuit for noncodeword inputs. When the interconnection of self-checking circuit blocks is considered, it is necessary to examine the mapping of the input noncodeword, since under failures in previous circuit blocks a circuit block might receive noncodeword inputs.

Now we define a pair (X_p, X_q) where X_p is a member of the input codespace of the circuit **L** and X_q is member of the input noncodespace. The significance of the ordered pair is that X_q represents an erroneous input that may be received by **L** instead of X_p. That is, $X_q = X_p + E$, where $X_p \in \mathbf{N}$, $X_q \in \Omega_x - \mathbf{N}$, and E expresses an input error. The circuit **L** we consider here is assumed to be fault free, and X_p and X_q are entirely dependent on circuit blocks that are previous to **L**.

Definition 12.6 [SMIT76] A circuit **L** with output codespace **S** is *error secure (ES)* for noncodespace (input) $\Omega_x - \mathbf{N}$ if for any input X_q in $\Omega_x - \mathbf{N}$, where $X_q = X_p + E$, $X_p \in \mathbf{N}$, $E \neq 0$.

$$Y(X_q, \lambda) \notin S \quad \text{or} \quad Y(X_q, \lambda) = Y(X_p, \lambda). \qquad \square$$

Figure 12.10 illustrates this definition.

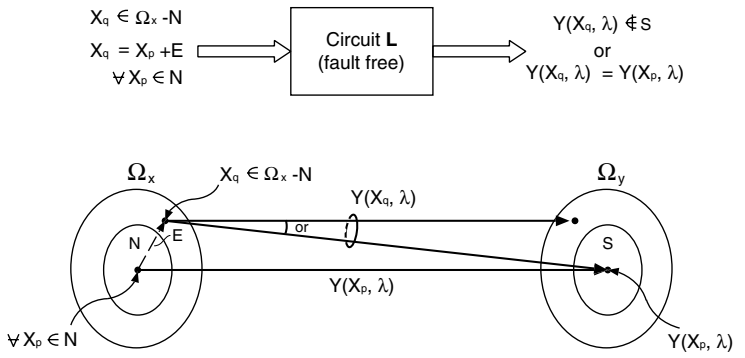


Figure 12.10 *Error secure circuit.*

If the circuit is error secure and receives an erroneous input, then it either passes a noncodeword on to subsequent circuit blocks, or masks (i.e., corrects) the error in the input word.

Definition 12.7 [SMIT76] A circuit **L** with output codespace **S** is *error preserving* for noncodespace $\Omega_x - \mathbf{N}$, if for any input X in $\Omega_x - \mathbf{N}$,

$$Y(X, \lambda) \notin \mathbf{S}. \qquad \square$$

A circuit that is error preserving is error secure, but the converse is not necessarily true. The term *code disjoint (CD)* is synonymous with *error preserving* [ANDE71]. That is, a code disjoint (or error preserving) circuit maps all of the members of its input noncodespace to noncodeword outputs, and this concept applies only to circuits under fault-free operation.

Next we consider the case of a circuit **L** having a fault sequence $\langle f_1, f_2, \ldots, f_n \rangle$, where $f_i \in \mathbf{F}$, $1 \leq i \leq n$, in addition to noncodeword inputs. The following definitions are based on [NICO84].

Definition 12.8 Before the occurrence of any fault, circuit **L** is code disjoint. Suppose that circuit **L** is such that for a sequence of fewer than m $(2 \leq m \leq n)$ faults in **F** and for all noncodeword inputs $\mathbf{X} \in \Omega_x - \mathbf{N}$,

$$Y(X, \langle f_1, f_2, \ldots, f_{m-1} \rangle) \notin \mathbf{S}.$$

Assume that for the m-fault sequence $\langle f_1, f_2, \ldots, f_m \rangle$, circuit **L** is self-testing. That is,

$$Y(X, \langle f_1, f_2, \ldots, f_m \rangle) \notin \mathbf{S} \qquad \text{for some } X \in \mathbf{N}.$$

Then **L** is said to be *strongly code disjoint (SCD)* for $\langle f_1, f_2, \ldots, f_m \rangle$. $\square$

Definition 12.9 A circuit **L** is *strongly code disjoint (SCD) for* $\mathbf{F} = \{f_1, f_2, \ldots, f_m, \ldots, f_n\}$ if it is SCD for all sequence of faults in **F**. $\square$

This circuit is illustrated in Figure 12.11.

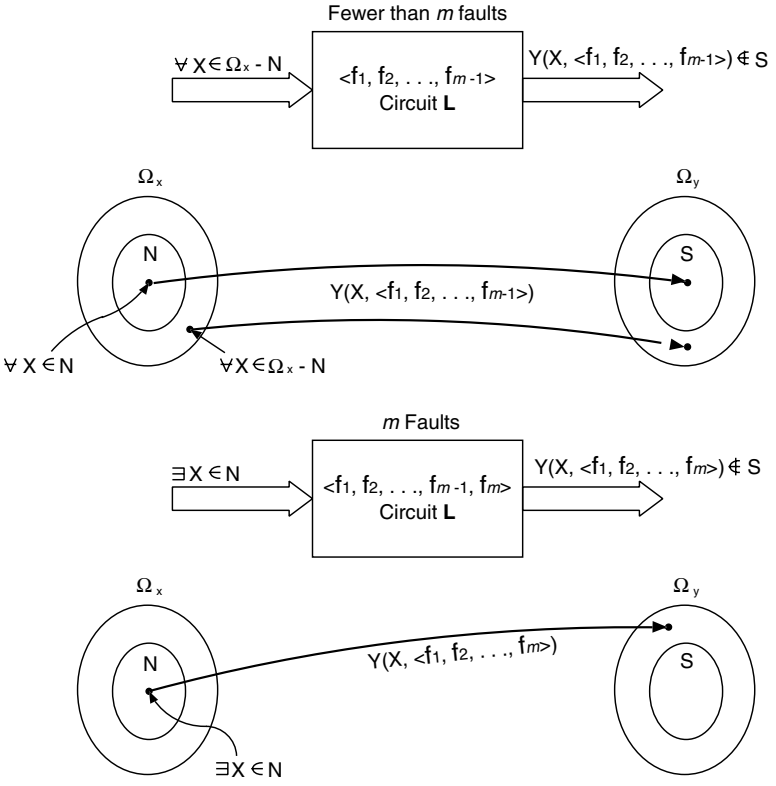


Figure 12.11 *Strongly code disjoint circuit.*

In this definition an assumption has to be made regarding the occurrence of faults in the circuit such that faults occur one at a time, and between any two fault occurrences there is sufficient interval for all code inputs to be applied to the circuit.

In Definition 12.8 we could also say that for all inputs $X \in \Omega_x - \mathbf{N}$ and an m-fault sequence fault $\langle f_1, f_2, \ldots, f_m \rangle$, the circuit satisfies $Y(X, \langle f_1, f_2, \ldots, f_m \rangle) \notin \mathbf{S}$ [NICO84]. That is to say, there is no such input $X \in \Omega_x - \mathbf{N}$ that satisfies $Y(X, \langle f_1, f_2., \ldots, f_m \rangle) \in \mathbf{S}$. This definition is stronger than Definition 12.8. Notice that the stronger property is not necessary, because it is more difficult to implement.

3. Self-checking Logic Networks

We consider the logic network of circuit blocks shown in Figure 12.12.

In this network we have to consider the faults in the interconnection between circuit blocks as well as those in each circuit block. That is, input noncodewords are given to the circuit block not only by the faults in the previous circuit blocks but also by the faults in the input connection lines.

We divide the network into two subnetworks. One is the subnetwork $\mathbf{L}_{\mathrm{I}}$ of the output interface circuit blocks, whose outputs are primary outputs of the network, and the other one is the subnetwork $\mathbf{L}_{\mathrm{II}}$ of the remaining internal circuit blocks. If the subnetwork $\mathbf{L}_{\mathrm{II}}$ is SFS for a fault set $\mathbf{F}_{\mathrm{II}}$, the outputs of $\mathbf{L}_{\mathrm{II}}$ are always correct codewords or noncodewords. Furthermore, if the subnetwork $\mathbf{L}_{\mathrm{I}}$ is SFS and SCD for a fault set $\mathbf{F}_1$ the outputs of $\mathbf{L}_{\mathrm{I}}$ are

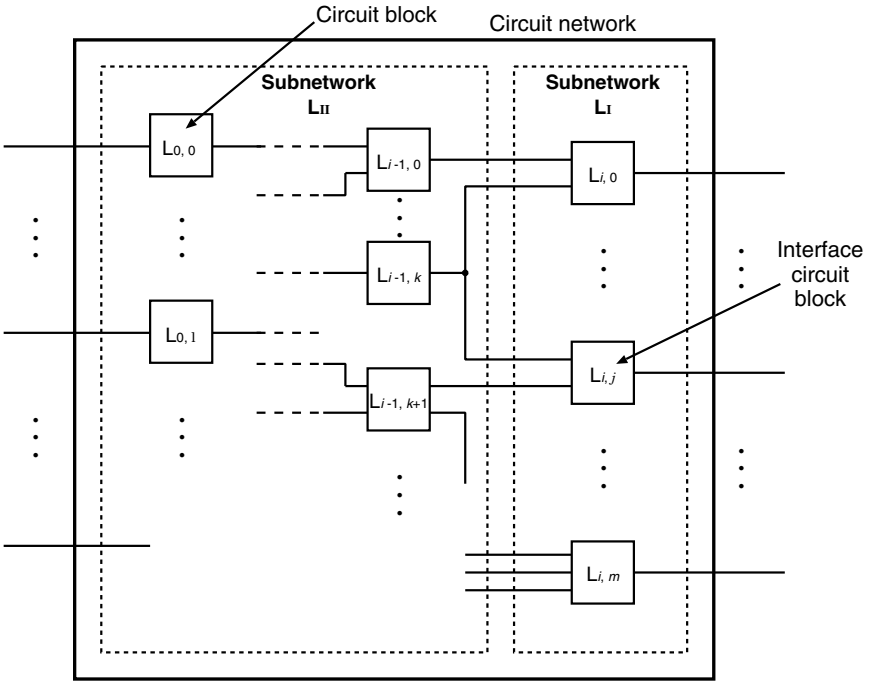


Figure 12.12 *Circuit block network.*

always correct codewords or noncodewords. This is because $\mathbf{L}_{\mathrm{I}}$ is SFS for codeword inputs and SCD for noncodeword inputs, and therefore the outputs of $\mathbf{L}_{\mathrm{I}}$ are always correct codewords or noncodewords for all inputs to $\mathbf{L}_{\mathrm{I}}$ (i.e., all outputs of $\mathbf{L}_{\mathrm{II}}$), even if some faults exist in both $\mathbf{L}_{\mathrm{I}}$ and $\mathbf{L}_{\mathrm{II}}$. Hence the total network is SFS. From the foregoing considerations, an SFS network can be obtained as follows.

Theorem 12.1 *If the subnetwork $\boldsymbol{L}_I$ consisting of the output interface circuit blocks is SFS and SCD for fault set $\boldsymbol{F}_I$, and if the remaining subnetwork $\boldsymbol{L}_{II}$ consisting of the internal circuit blocks is SFS for fault set $\boldsymbol{F}_{II}$, then the total network is SFS for all fault sequences in $\boldsymbol{F}_I$ and $\boldsymbol{F}_{II}$.*

If the faults exist in either $\mathbf{L}_{\mathrm{I}}$ or $\mathbf{L}_{\mathrm{II}}$ but not in both, then the code disjoint (CD) property can replace SCD as a requirement for subnetwork $\mathbf{L}_{\mathrm{I}}$.

Corollary 12.1 *If the subnetwork $\boldsymbol{L}_I$ of the output interface block is SFS and CD, and if the subnetwork $\boldsymbol{L}_{II}$ of the remaining internal blocks is SFS, then the total network is SFS for all fault sequences in either $\boldsymbol{L}_I$ or $\boldsymbol{L}_{II}$, but not in both.*

The SFS network has important advantages. In an SFS network it may not be necessary to place checkers at every circuit block but only at the output interface blocks. The checker placement will then be determined by the mean time between failures (MTBF) of the network. General design methods for SFS networks have not yet been fully established. However, an SFS combinational circuit network can be easily realized if it is inverter free, that is, if it contains only AND gates and OR gates, as stated in the following theorem.

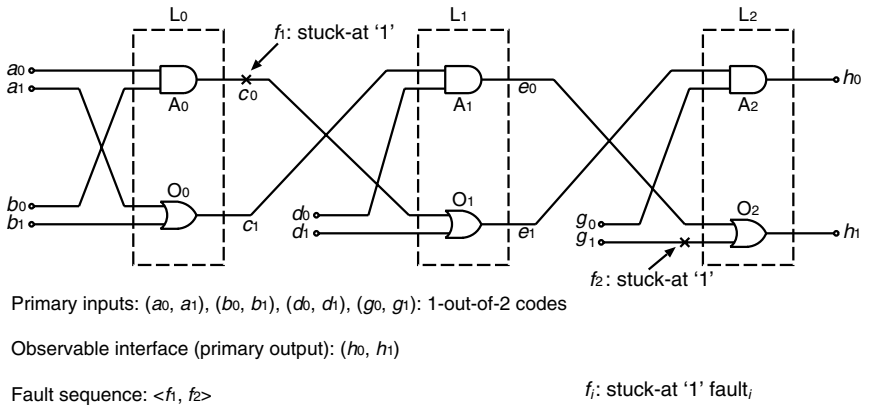


Figure 12.13 *Cascaded inverter-free network.*

Theorem 12.2 [SMIT78] *A network that consists of only inverter-free combinational circuit blocks and in which the block input / output interfaces are encoded with unidirectional error detecting codes is SFS with respect to unidirectional faults.*

Proof Unidirectional faults in an inverter-free circuit block or in the input interconnection line always cause unidirectional errors. Every combinational circuit block has an input and output encoded with a unidirectional error detecting code (e.g., the m-out-of-n code and the Berger code), so the unidirectional errors always map a codeword to a noncodeword. Further, because unidirectional faults in the inverter-free circuit block never map unidirectional error input to a codeword output, these faults at a circuit block can be propagated to the output interface of the network and be detected by a unidirectional error decoder. Q.E.D.

Example 12.3

Consider a network that consists of three inverter-free combinational circuit blocks, $\mathbf{L}_0$, $\mathbf{L}_1$, and $\mathbf{L}_2$ interconnected in cascade as shown in Figure 12.13. The 1-out-of-2 codes are used at all block interfaces as well as the primary output of the network (i.e., the observable interface).

In the example the two unidirectional faults, stuck-at '1' faults, are assumed to exist at the output of the A_0 gate first and then at one input of the O_2 gate. Outputs in each block interface are shown in Figure 12.14, where inputs (a_0, a_1) and (b_0, b_1) are, for example, equal to $(1, 0)$ and $(0, 1)$, respectively. At the first fault stage, which includes only the f_1 fault, only case IV of Figure 12.14 has the noncodeword output. If there is no such input as in case IV—that is, no such input set as $(a_0, a_1) = (1, 0)$, $(b_0, b_1) = (0, 1)$, $(d_0, d_1) = (1, 0)$, and $(g_0, g_1) = (1, 0)$—this network always has the correct codeword outputs. Because of the inputs at the second fault stage, which include the f_2 fault as well as the f_1 fault, case II also has noncodeword outputs. So the primary output (h_0, h_1) is always the correct codeword or noncodeword for these inputs. In case III of Figure 12.14 it should be noted that all circuit blocks are not always CD.

In the example above we saw that the circuit network shown in Figure 12.13 is SFS for the unidirectional fault sequence $\langle f_1, f_2 \rangle$ and for restricted inputs, even though each

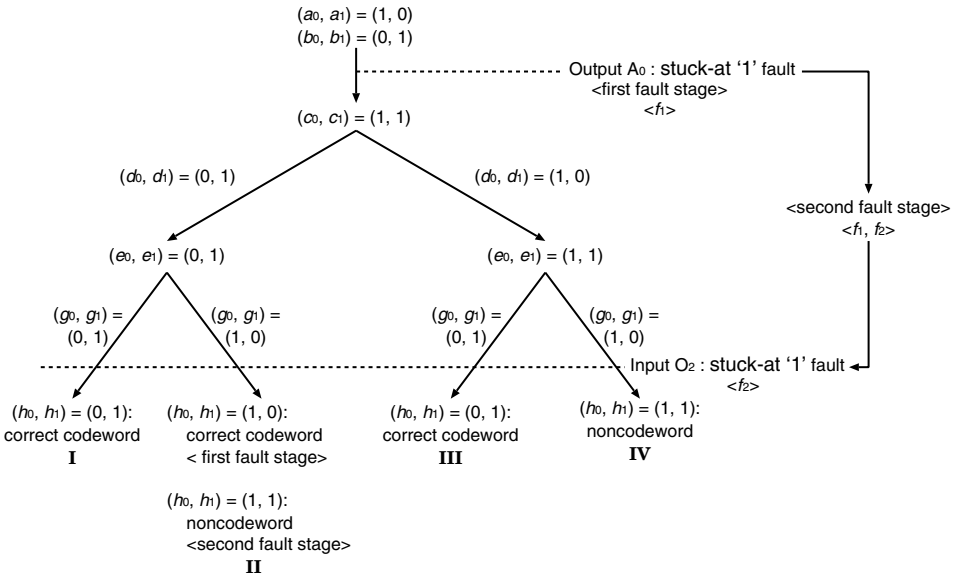


Figure 12.14 *Interface outputs in the circuit of Figure 10.13.*

circuit block is not always CD. Clearly, noncodewords can appear at embedded interfaces of the network before a noncodeword is produced as the first erroneous output. Therefore we turn next to consider some new comprehensive concepts of error secure and error propagating are defined for the circuit block interfaces.

Let $\mathbf{F}$ be the set of all faults in a network consisting of some circuit blocks and let $\langle \mathbf{F} \rangle$ be the set of all sequences of faults in $\mathbf{F}$. Given a fault sequence in $\langle \mathbf{F} \rangle$, the network is assumed to be initially fault free and capable of performing the correct function. Once the first fault in the sequence occurs, the network goes into the first fault stage; further on, when the second fault occurs, the system will go into the second fault stage, and so on.

The following definitions and theorem are based on [NANY88].

Definition 12.10 A circuit block interface is *error secure* (ES) at a given fault stage if any error that can occur at the interface never causes an incorrect codeword at any observable interface during the fault stage. □

Definition 12.11 A circuit block interface is *error propagating* (EP) at a given fault stage if any error that can occur at the interface eventually produces a noncodeword at some observable interface during the fault stage. □

It is noted that error secure (ES) in Definition 12.10 and error secure in Definition 12.6 are different in that the former is defined for block interfaces that may include some faults within them while the latter is defined for fault-free circuit blocks. Therefore the Definition 12.10 covers Definition 12.6, but the converse is not necessarily true.

The concept of error propagating includes both error preserving (i.e., code disjoint) and self-testing. Theorem 12.3 gives the relation between ES / EP and SFS in a network that consists of some circuit blocks and their interfaces.

Theorem 12.3 *A network is SFS for a fault set* $\boldsymbol{F}$ *if for every fault sequence in* $\langle \boldsymbol{F} \rangle$ *all the observable interfaces remain error secure* (ES) *until the earliest fault stage at which at least one observable interface becomes error propagating* (EP).

A design method for an SFS microprocessor based on this theorem is given in [NANY88], and will be shown in Section 12.4.

12.1.2 Checker Concept

For a functional circuit to be a TSC or an SFS circuit, the error detection circuit (i.e., the checker) must include an error indicator. Consider a circuit **G** that consists of a functional circuit under check (**L**) and a checker (**CK**), as is shown in Figure 12.2.

The model for the error detection circuit is also as given in Figure 12.2 [SELL68]. In general, a checker receives inputs X, and also W and Y from a circuit **L**. The line W generated in the process of generating outputs may not be the output Y itself. There have already been many kinds of checkers for special functional circuits [SELL68]. Some are code checkers such as parity-code checkers, residue code checkers, m-out-of-n code checkers, and Berger code checkers (Berger code will be discussed in Subsection 12.4.2). Others are prediction checkers. The self-checking concepts in Subsection 12.1.1 have been defined for code checkers, that is, for checking the coded output Y. They do not always hold for other types of checkers (e.g., prediction checkers). Here we will outline two cases:

Case 1. Output Y encoded. The checker included in this case is a *code checker.* The checker's input is always Y, which is the output of the circuit **L**. This is shown in Figure 12.15.

Case 2. Output Y nonencoded. Many types of functional circuits, such as control circuits, have their outputs nonencoded. For this type of circuit, *duplication checkers, parity prediction checkers*, and *input regeneration checkers* [SELL68] have been designed and applied to computer logic systems.

Figure 12.16 gives the concept of the *duplication checker.* Note that the functional circuit **L** is duplicated, and the outputs are compared.

Figure 12.17 gives the concept of the *parity prediction checker* for circuit **L**. A parity bit of the output Y is generated independently by the prediction circuit **P** having input X, while an equivalent output parity bit is generated from output Y. Then these parity bits are compared.

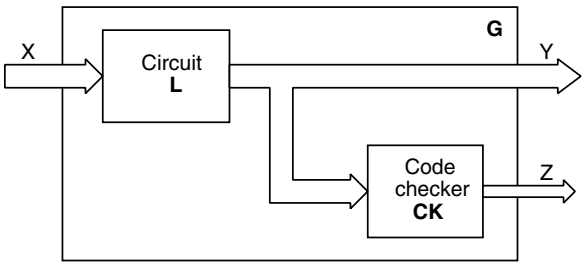


Figure 12.15 *Checking by examining outputs—that is, a code checker.*

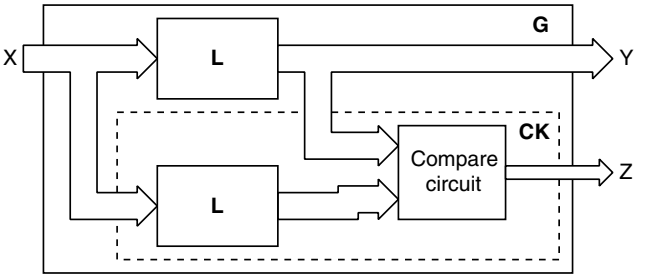


Figure 12.16 *Checking by duplication—that is, a duplication checker.*

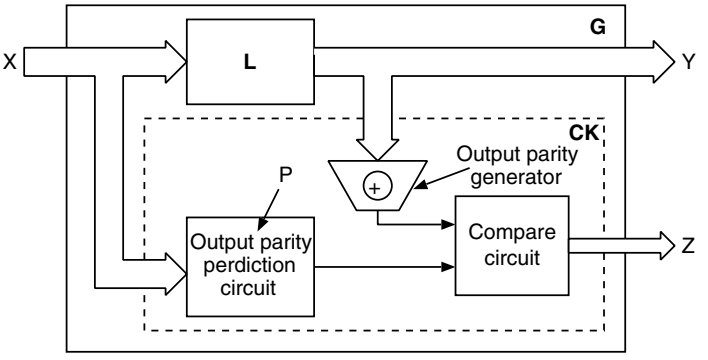


Figure 12.17 *Checking by parity prediction—that is, a parity prediction checker.*

Figure 12.18 gives the concept of the *input regeneration checker*. This checker regenerates the inputs from the outputs by inverse logic $\mathbf{L}^{-1}$ of the original circuit $\mathbf{L}$ and compares the regenerated inputs with the original inputs. So the functional logic of $\mathbf{L}$ of the input regeneration checker has a unique inverse operation. Example 12.4 illustrates this principle.

Example 12.4 [SELL68]

Let $\mathbf{L}$ be a 3-input $(x_0\ x_1\ x_2)$, 3-output $(y_0\ y_1\ y_2)$, combinational circuit defined by the following truth table:

x_0	x_1	x_2	y_0	y_1	y_2
0	0	0	1	1	1
0	0	1	0	0	1
0	1	0	1	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	1	0	1

Select x_1 to form a sub-inverse circuit $\mathbf{L}_1^{-1}$ according to the following equation:

$$x_1 = y_0 \oplus y_1 \cdot y_2.$$

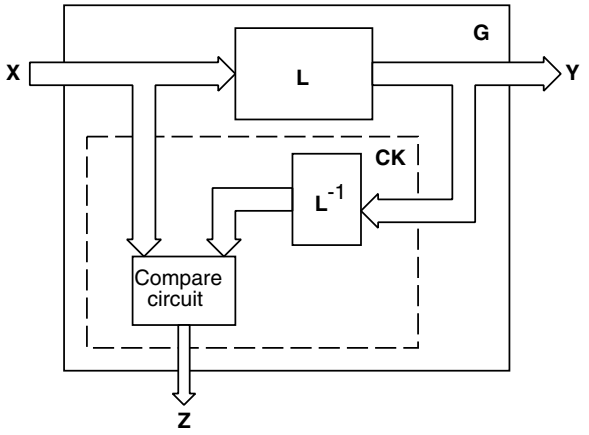


Figure 12.18 *Checking by regenerating inputs—that is, an input regeneration checker.* Source: *[SELL68].*

Then the error detection circuit **CK** is as shown in Figure 12.19. This checker's ability is limited because not all inputs are checked. This checker will detect if either y_0 or $y_1 \cdot y_2$ is in error, but not both.

The duplication checker and the input regeneration checker can also be considered special cases of the prediction checker because the circuit output Y is exactly "predicted" by the duplicated circuit in the duplication checker, and inversely, the circuit input X can be said to be "predicted" from the circuit output Y in the input regeneration checker. This will be discussed more fully in Subsection 12.2.5. Hence the checker included in case 2 is called a *prediction checker.*

Note that the use of internal logic signals W shown in Figure 12.2 sometimes plays an important role in error detection. In some cases it is difficult to check a logic circuit perfectly without such signals. For example, in parity-checked adders it is necessary to use the carries generated during addition (see Subsection 12.3.1). In this case a single fault may cause several outputs to be in error. These errors cannot be detected by only examining the output alone.

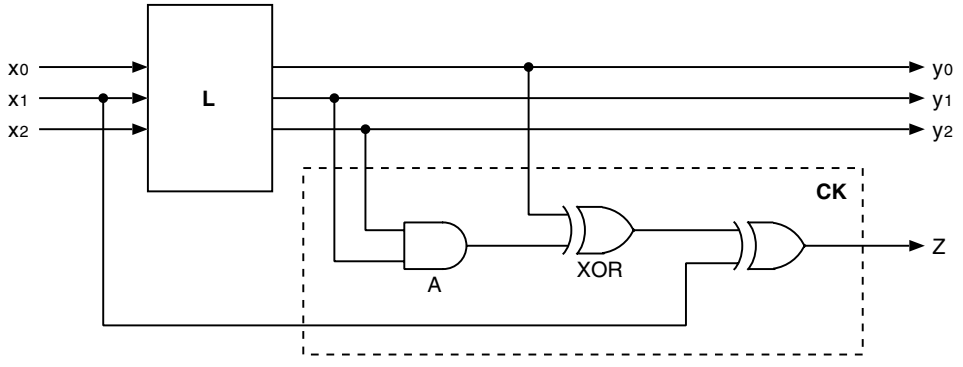


Figure 12.19 *Simplified input regeneration checker.* Source: *[SELL68].*

Further note that in case 2, if input X is encoded and the prediction checker includes the function of code checker, then this prediction checker may detect input errors under the condition that the circuit under check and the checker itself are fault free. This case satisfies the conditions for circuit **G** being code disjoint. Such cases can be seen in self-checking adders (e.g., full-sum parity-checked adders; see Subsection 12.3.1).

Next we consider the case where faults exist also in the checkers. In this case we require a self-checking checker, and the input and output codespaces should be clearly defined.

Lemma 12.1 [CART68] *For single faults it is necessary that any checker must have at least two outputs and that no outputs take constant values.*

The combinational circuit being tested is always monitored by a checker that signals the appearance of a noncodeword. For example, the output of a checker might be 0 whenever a codeword is present in the output of the circuit under test, and one when a noncodeword is present. It is also desirable that the checker be self-checking so that a fault in the checker itself produces an error signal. It is obvious that the simple encoding mentioned above is not sufficient, since a single stuck-at '0' output of the checker would never be detected. Therefore it is necessary that the checker has at least two outputs and that no outputs take constant values for codespace input. This is a brief proof for Lemma 12.1.

In order to make the checker self-checking, its output must be encoded in an error detecting code. The simplest distance-2 error detecting codes are the *1-out-of-2 code* (codeword = $\{(0, 1),\ (1, 0)\}$) and the *duplication code* (codeword = $\{(0, 0),\ (1, 1)\}$). In the 1-out-of-2 code, the outputs $(0, 0)$ and $(1, 1)$ indicate errors. The 1-out-of-2 code is generally preferred because it detects unidirectional errors, such as those produced by a loss of power.

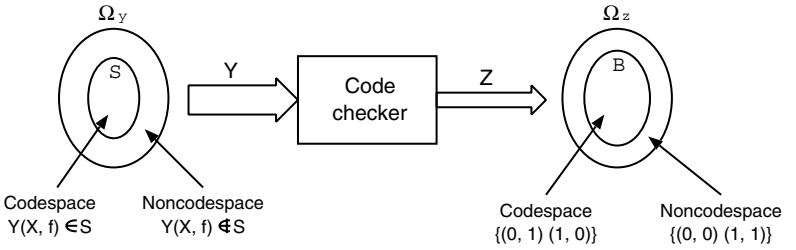
From this consideration, the *checker output space* Ω_z is defined as the set $\{(0, 0),\ (0, 1),\ (1, 0),\ (1, 1)\}$, the *checker output codespace* $\mathbf{B} = \{(1, 0),\ (0, 1)\}$, and the *checker output noncodespace* $\Omega_z - \mathbf{B} = \{(0, 0),\ (1, 1)\}$.

Next we will define the input space for the two types of checkers, the code checker and the prediction checker, shown in Figure 12.20. Note that in the prediction checker the input of the checker (i.e., the output of the circuit **L**) does not have redundancy. The output Y is only defined by the input X and the function of the circuit **L** (i.e., $Y(X)$). Because the input of the checker is input X and output Y of the circuit **L**, we can define (X, Y) as the checker input space. We let the fault-free output Y be $Y_0 = Y(X, \lambda), X \in \mathbf{N}$. Hence the input codespace of the prediction checker is (X, Y), where $Y = Y_0$, and the input noncodespace is (X, Y), where $Y \neq Y_0$. If the signal W is also applied to the checker, then the input space will be defined as (X, Y, W).

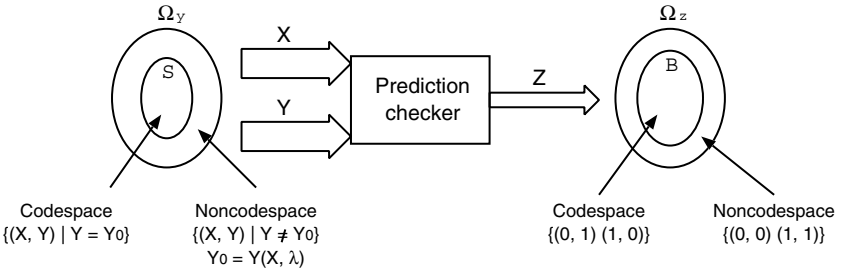
As already mentioned, the error-preserving property (i.e., the code-disjoint property) requires a checker. That is, the checker always maps an input codeword in **S** to an output codeword in $\{(0, 1),\ (1, 0)\}$, and an input noncodeword $\notin \mathbf{S}$ to an output noncodeword in $\{(0, 0),\ (1, 1)\}$.

Definition 12.12 A circuit is a *self-testing checker* if it is self-testing and code disjoint. □

Lemma 12.2 *The fault-secure property is not necessary for a checker.*



(a) Input and output spaces of the code checker



(b) Input and output spaces of the predication checkers.

Figure 12.20 *Input and output spaces of the checkers.*

It is not important that the checker has any erroneous codewords at its output. That is, we are interested only in whether or not the checker's output is a codeword or noncodeword, not which codeword it is. The self-testing property must only ensure the detection of faults inside the checker. Therefore a *self-testing checker* should be satisfactory. This is the brief proof for Lemma 12.2.

If the checker **CK** is a self-testing (ST) checker and the circuit **L** is TSC, then the total circuit **G** shown in Figure 12.2 is TSC. Note that the checker is an *embedded circuit*; that is, not all input codewords are given to the checker. Input to the checker is output of the circuit **L** that does not always provide all codewords to the checker. Hence it is difficult for the checker to satisfy the self-testing property perfectly without adopting some special design techniques [KHAK84, JHA84, HUGH84, FUJI87a].

We are interested in preserving the code-disjoint property even in the presence of faults inside the checker. In this case the checker should be *strongly code disjoint* (SCD); see Definitions 12.8 and 12.9.

If the circuit under check **L** is SFS and the checker **CK** is SCD, then the total circuit **G** is SFS. In this case note that an assumption has to be made regarding the occurrence of faults in the checker **CK** and the circuit **L**, as follows [NICO84]:

*After the occurrence of a fault affecting the checker, a sufficient time elapses such that all code inputs included in **S** are applied before a new fault can occur in the checker **CK** or in the circuit under check **L**. After the occurrence of a fault affecting the circuit under check **L**, a sufficient time elapses such that all code inputs included in **N** are applied before a new fault can occur in the checker **CK** or in the circuit under check **L**.*

12.2 SELF-TESTING CHECKERS

As we saw in the preceding section, the checker should have a code disjoint property or an error preserving property. The self-checking checker should satisfy in addition the self-testing property for all faults in the prescibed fault set **F**. Note that the latter condition is sometimes difficult to satisfy as was mentioned previously because the checker is generally an embedded circuit and all input codewords cannot be applied to the checker. In other words, the inputs to the checker are the outputs of the circuit under check, and therefore it may not be possible to apply all the input codewords to the checker. This depends on the circuit whose outputs are the inputs to the checker.

Considerable literature has appeared on implementing self-testing checkers for all input codewords and for all single faults, or for all unidirectional faults [CART70, ANDE71, ANDE73, REDD74a, REDD74b, SMIT77, ASHJ77, MARO78b, FUJI87a, LALA01]. Much work has been done on self-testing m-out-of-n code checkers and separable code checkers [ASHJ77, DAVI78a, GAIT83, GOLA84, HALA83, ITOH80, IZAW81, IZAW84a, IZAW84b, KHAK82c, LO87b, MARO78a, PIES83, TAO87, LALA01]. In this section practical self-testing checkers, such as parity code checkers, two-rail code checkers, and prediction checkers, are discussed from the standpoint of implementation.

12.2.1 Parity Code Checker

This checker can be constructed using a multiple exclusive-OR circuit ($\oplus$) in a tree structure. The self-checking parity-tree circuit is easily implemented when the input code vectors have *odd parity*. Two parity-tree circuits are obtained by dividing the inputs [CART70]. As a simple example we implement a self-checking parity code checker with five inputs. Input $Y = (y_0\ y_1\ y_2\ y_3\ y_4)$ is divided into two sets, for example, $\{y_0,\ y_1,\ y_2\}$ and $\{y_3,\ y_4\}$, and the two independent parity trees are constructed from these input sets, as shown in Figure 12.21.

Table 12.2 shows the function and lines tested by the self-checking parity code checker of Figure 12.21. All exclusive-OR (or XOR) gates in the table have all possible input pairs $\{(0, 0),\ (1, 0),\ (0, 1),\ (1, 1)\}$ and output (z_0, z_1) takes $\{(0, 1),\ (1, 0)\}$ for all input codewords. For even-parity encoded inputs (i.e., noncodeword inputs), this checker always gives output $(z_0, z_1) = \{(0, 0),\ (1, 1)\}$. Hence the circuit is code disjoint. Also apparently

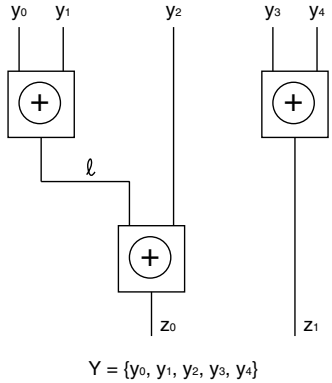


Figure 12.21 *Self-checking parity checker.*

TABLE 12.2 Function and Lines Tested of Parity-Tree Circuit

y_0	y_1	l	y_2	y_3	y_4	z_0	z_1
1	0	1	0	0	0	1	0
0	1	1	0	0	0	1	0
0	0	0	1	0	0	1	0
1	0	1	0	1	1	1	0
0	1	1	0	1	1	1	0
0	0	0	1	1	1	1	0
1	1	0	1	0	0	1	0
1	1	0	1	1	1	1	0
1	1	0	0	1	0	0	1
0	1	1	1	1	0	0	1
1	0	1	1	1	0	0	1
1	1	0	0	0	1	0	1
0	1	1	1	0	1	0	1
1	0	1	1	0	1	0	1
0	0	0	0	0	1	0	1
0	0	0	0	1	0	0	1

a single fault always causes the output to be noncodeword for some codeword input. Therefore we have the following theorem.

Theorem 12.4 *The parity-tree circuit that is divided into two independent parity-tree subcircuits is a self-testing checker for all odd-parity encoded inputs and for all single faults.*

When the input code vectors have *even parity*, one subcircuit output is designed to be inverted.

We now consider the case where not all codeword inputs are given to the checker. In this situation it is difficult to satisfy the self-testing property at all times.* Some methods for overcoming this difficulty were proposed in [KHAK82b, KHAK84, FUJI87a]. Here we consider the XOR tree circuit with a cascaded form and an extra input v that can take any value in $\{0, 1\}$ [FUJI87a]. Figure 12.22 gives an example of this type of tree structure.

Theorem 12.5 *If the cascaded XOR tree circuit having m XOR gates satisfies the following conditions, then every single fault in this circuit can be detected:*

1. *Primary input $y_i \neq$ constant, $i = 0, 1, \ldots, m-1$.*
2. *Independent of y_i's, input v can take any value in $\{0, 1\}$.*

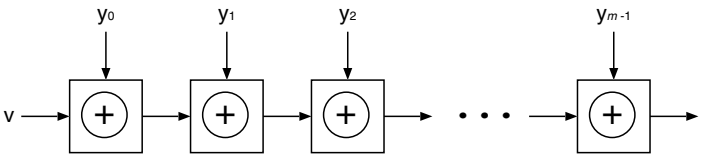


Figure 12.22 *Cascaded parity-tree circuit with one additional input v.*

*Typically, four- or five-input test patterns are necessary to test the multiple-input parity-tree circuits consisting of only two-input XOR or XNOR (exclusive-NOR) gates [BOSS70, HONG81, REDD85, MOUR86a, MOUR86b].

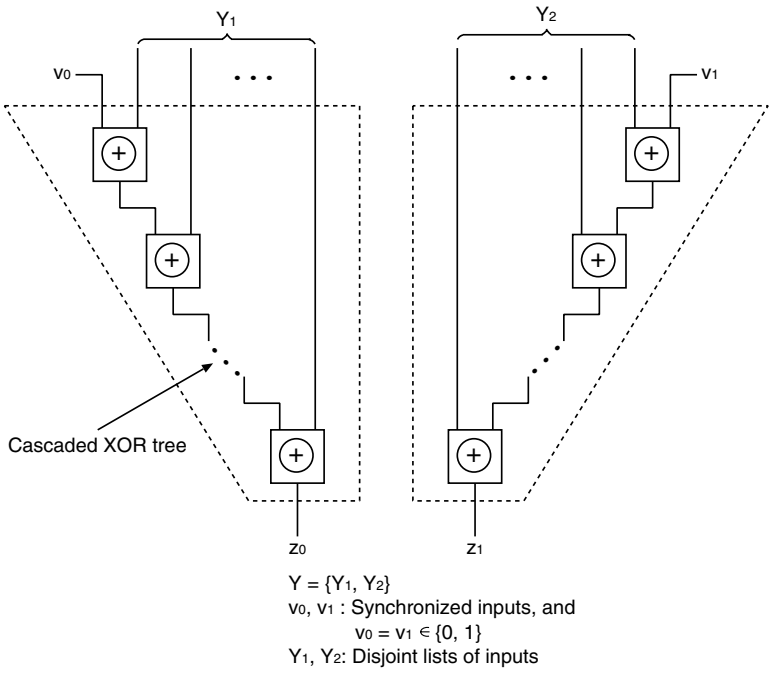


Figure 12.23 *Self-testing parity code checker.*

Any error due to a fault in this circuit can be propagated to the output, and there exists at least one input pattern to detect this fault. Therefore each divided tree circuit of Figure 12.21 can be transformed into a cascaded structure like that shown in Figure 12.22. Then the circuit is a self-testing checker for the restricted number of input codewords that satisfy the condition 1 of Theorem 12.5. This is shown in Figure 12.23. In this figure it should be noted that synchronized inputs v_0 and v_1 $(v_0 = v_1 \in \{0, 1\})$ are applied to each XOR tree. This is because the checker can also detect faults in v_0 and v_1. This self-testing checker structure requires only two more XOR gates than the previously given structure.

12.2.2 Two-Rail Code Checker

A two-rail code checker (also called a comparator) is required for *a duplication check*, meaning a *comparison check*. This is shown in Figure 12.24. In order to prevent identical failure states in both the functional circuit and its duplicated circuit, the duplicated circuit is sometimes designed in a complementary form [SEDM80b], which will be shown in Figure 12.51 in Subsection 12.4.2. In Figure 12.24 the output of one of the two identical circuits is inverted and fed into a self-checking two-rail code checker, discussed below. In a complementary duplication, no inverter gates are needed, and the outputs from both functional circuits are fed directly to a two-rail code checker.

In a two-rail encoding, bits of information occur as complementary pairs, $(0, 1)$ and $(1, 0)$; the pairs $(0, 0)$ and $(1, 1)$ indicate errors. Thus the same circuit that checks a two-rail encoding can be used to combine several self-checking checker output pairs into one pair. That is, a self-checking two-rail code checker (i.e., a self-testing comparator) will map m input pairs, referred to as $\{(a_0, b_0),\ (a_1, b_1),\ \ldots,\ (a_{m-1}, b_{m-1})\}$, to one output

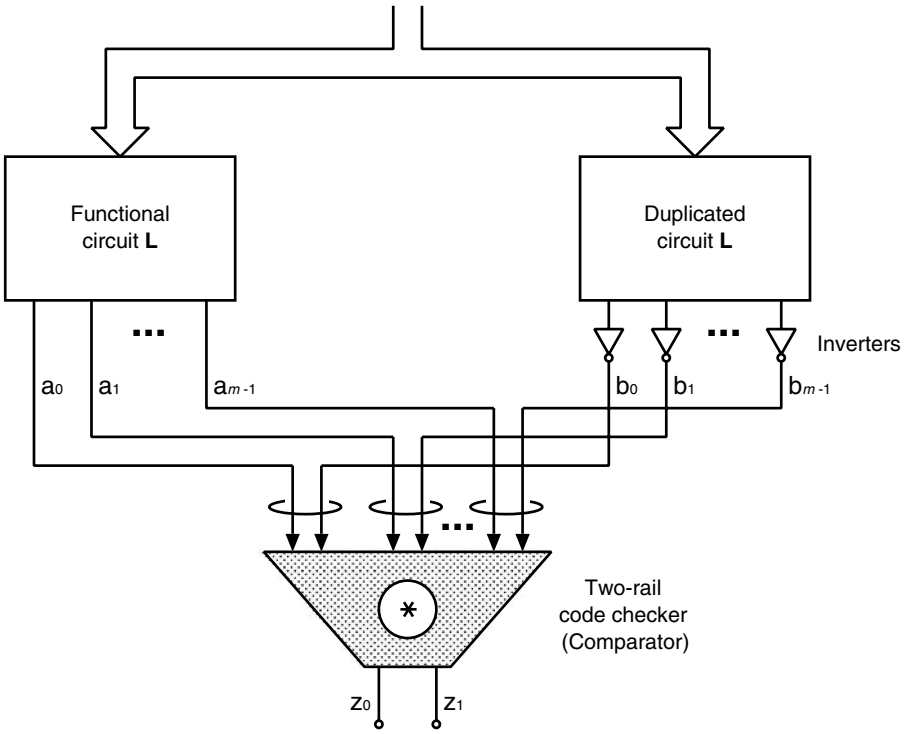
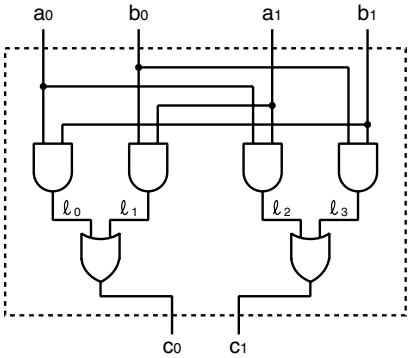


Figure 12.24 *Two-rail code checker for duplication check.*

pair, referred to as (z_0, z_1). The output pair must be complementary if and only if each and every input pair is all complementary [TAMI84].

A two-rail code checker for $m = 2$ is shown in Figure 12.25 [CART70]. This is a two-level AND-OR realization. The table in Figure 12.25 shows this circuit to be self-testing for single faults and also code disjoint.

Lemma 12.3 [FUJI87a] *Let the input to the checker be* $A_i = (a_{0,i},\ a_{1,i})$ *and* $B_i = (b_{0,i},\ b_{1,i})$, *and the output be* $C_{i+1} = (c_{0,i+1},\ c_{1,i+1})$. *Here* A_i, B_i, $C_{i+1} \in \{\mathbf{0}, \mathbf{1}, \boldsymbol{W}\}$,



a_0	b_0	a_1	b_1	ℓ_0	ℓ_1	ℓ_2	ℓ_3	c_0	c_1
0	1	0	1	0	0	0	1	0	1
0	1	1	0	0	1	0	0	1	0
1	0	0	1	1	0	0	0	1	0
1	0	1	0	0	0	1	0	0	1
0	0	x	x	0	0	0	0	0	0
x	x	0	0	0	0	0	0	0	0
1	1	1	x	x	1	1	x	1	1
1	1	x	1	1	x	x	1	1	1
1	x	1	1	1	x	1	x	1	1
x	1	1	1	x	1	x	1	1	1

'x' can take any value of 0, 1.

Figure 12.25 *A two-rail code checker for* $m = 2$. Source: *[CART70]. © 1970 IEEE.*

where $\mathbf{0} = (0,1)$, $\mathbf{1} = (1,0)$, $\boldsymbol{W} \in \{(0,0),\ (1,1)\}$. *Then* C_{i+1} *can be expressed as the following truth table:*

A_i \ B_i	0	1	W
0	0	1	W
1	1	0	W
W	W	W	W

$C_{i+1} = A_i \circledast B_i$

0 = (0,1)
1 = (1,0)
W = { (0,0), (1,1) }

The operation performed in the checker is expressed as $\circledast$, meaning $C_{i+1} = A_i \circledast B_i$, defined in the truth table.

Lemma 12.4 *Every single fault in the checker can be detected if the inputs* A_i *and* B_i *have the following four distinct patterns:*

$$(A_i,\ B_i) = \{(\mathbf{0},\mathbf{0}),\ (\mathbf{0},\mathbf{1}),\ (\mathbf{1},\mathbf{0}),\ (\mathbf{1},\mathbf{1})\}, \quad \mathbf{0} = (0,1),\ \mathbf{1} = (1,0).$$

Multi-input two-rail code checker can be implemented by interconnecting such two-level blocks to form multilevel trees of arbitrary size [ITOH82]. For example, a tree with eight input pairs formed by interconnecting seven blocks is shown in Figure 12.26.

Corollary 12.2 [FUJI87a]

1. *For* $A_i,\ B_i \in \{\mathbf{0},\mathbf{1}\}$, *the two-input two-rail code checker is equal to modulo-2 adder on* $\{\mathbf{0},\mathbf{1}\}$.
2. *As for the multi-input two-rail code checker implemented with the two-input two-rail code checkers, the output of this checker takes the values* $\mathbf{W}$ *if at least one input to this checker is equal to* $\mathbf{W}$, *where* $\mathbf{W} = \{(0,0),\ (1,1)\}$.

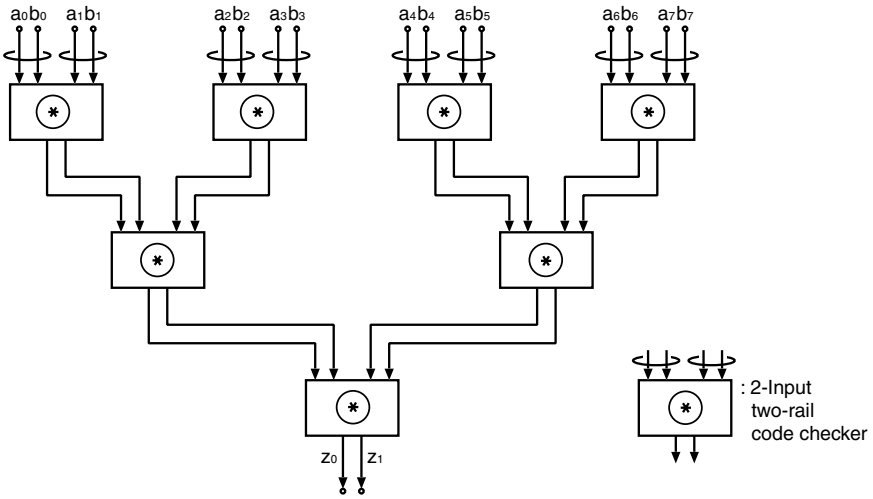


Figure 12.26 *Eight-input two-rail code checker.*

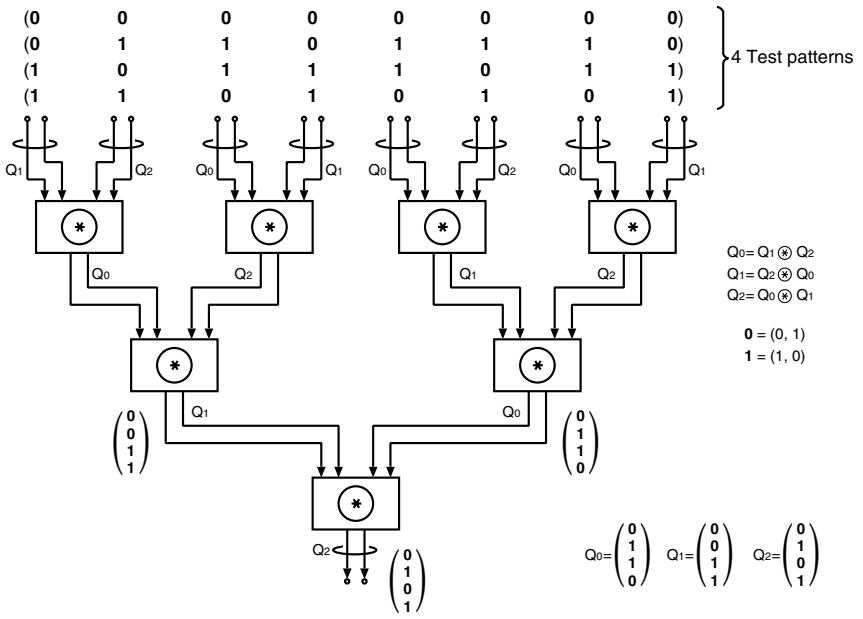


Figure 12.27 *Example of four test patterns generated systematically for 8-input two-rail code checker.*

In general, 2^m test patterns are sufficient to diagnose such multiple-input trees if each two-rail block has no more than m input pairs [ANDE71]. Therefore the two-rail tree shown in Figure 12.26 is completely diagnosed by the *four test patterns*. These test patterns are systematically obtained in [ANDE71]. Figure 12.27 shows an example of the four test patterns systematically obtained for the eight-input two-rail code checker. In this test pattern generation every two-input two-rail code checker has four distinct input patterns defined in Lemma 12.4.

In normal operation, however, all these patterns may not be applied to the multiple-input tree circuit. This is because these checkers are placed at the output of the circuit under check, as shown in Figure 12.24. That is, they are embedded, and hence a restricted number of patterns may be given to the checkers. Even for this situation some techniques have been proposed to satisfy the self-testing condition [HUGH84, FUJI87a]. The following theorem deals with the self-testing property of the multi-input two-rail code checkers.

Theorem 12.6 [FUJI87a] *If the M-input two-rail code checker having one input V shown in Figure 12.28 satisfies the following conditions,*

1. *Primary input* $A_i \neq$ *constant,* $A_i \neq \mathbf{W}$, $i = 0,\ 1,\ \ldots,\ M-1$.
2. *Independent of* A_i*'s, input* $V = (v,\ \bar{v})$ *can take any value in* $\{\mathbf{0},\ \mathbf{1}\}$,

then every single fault in this checker can be detected.

Proof The output of the i-th level two-input two-rail code checker can be expressed as follows:

$$C_{i+1} = A_i \circledast C_i, \qquad i = 0,\ 1, \ldots, M-1,$$
$$C_0 = V,$$

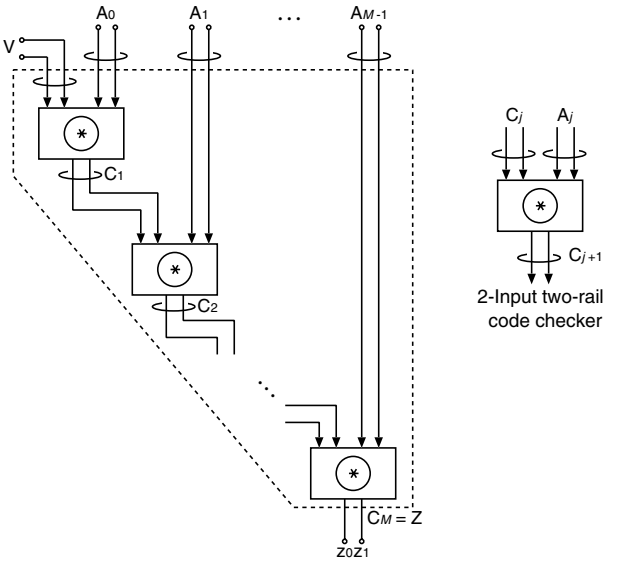


Figure 12.28 *Multi-input two-rail code checker with cascaded tree structure.*

where $\circledast$ is defined in Lemma 12.3. This can be expanded as follows:

$$C_{i+1}(V, A_0, A_1, \ldots, A_i) = V \circledast (A_0 \circledast A_1 \circledast \ldots \circledast A_i),$$
$$C_{i+1}(\overline{V}, A_0, A_1, \ldots, A_i) = \overline{V} \circledast (A_0 \circledast A_1 \circledast \ldots \circledast A_i)$$
$$= \overline{C_{i+1}(V, A_0, A_1, \ldots, A_i)},$$

where $C_{i+1}(V, A_0, A_1, \ldots, A_i)$ means that C_{i+1} is the function of V, A_0, A_1, ..., and A_i.

Here $\overline{\mathbf{1}} = \mathbf{0}$, $\overline{\mathbf{0}} = \mathbf{1}$. Hence C_{i+1} can take any value in $\{\mathbf{0}, \mathbf{1}\}$ by controlling the value of V. Furthermore, from the conditions 1 and 2 in this theorem, (A_i, C_i) can take four distinct patterns such as in Lemma 12.4.

The output of the tree circuit C_M can also be expressed as follows:

$$C_M(C_{i+1}, A_{i+1}, A_{i+2}, \ldots, A_{M-1}) = C_{i+1} \circledast (A_{i+1} \circledast A_{i+2} \circledast \ldots \circledast A_{M-1}).$$

If a single fault in this circuit causes the value of C_{i+1}, A_{i+1}, or C_{i+2} to be $\mathbf{W}$, then $C_M = \mathbf{W}$ from the truth table of Lemma 12.3. Therefore the error due to this fault can be propagated to the output of this circuit.

For all these reasons there exists at least one input satisfying the conditions 1 and 2 that can detect single faults in this checker. (Q.E.D.)

The input V can be generated from 2 J-K or toggle flip-flops whose outputs are inverted with every clock (see Figure 12.33 in Subsection 12.2.3.

12.2.3 Generalized Prediction Checker (GPC)

We consider a design for self-testing checker of the combinational circuits having nonencoded inputs and outputs. As stated in Section 12.1, we need to apply a prediction

checking concept to these circuits. First, we have a generalized prediction checker that includes a complementary duplication checker and a parity prediction checker as special cases, and then we have a self-checking prediction checker [FUJI84, FUJI87a].

A prediction checker always checks the relation between the circuit input and the output. The combinational circuit under check **L** is assumed to have a nonencoded n-bit input $X = (x_0\ x_1\ \ldots\ x_{n-1})$ and a nonencoded k-bit output $Y = (y_0\ y_1\ \ldots\ y_{k-1})$. The multi-output combinational circuit can then be expressed in a disjunctive canonical form as

$$Y = \mathbf{A} \cdot U, \tag{12.1}$$

where

$$\begin{aligned}
&Y = (y_0\ y_1\ \ldots\ y_{k-1})^T,\\
&U = (u_0\ u_1\ \ldots\ u_{2^n-1})^T: \quad \text{vector of minterms}\\
&\quad = (\bar{x}_0\bar{x}_1\cdots\bar{x}_{n-1}\ \ x_0\bar{x}_1\cdots\bar{x}_{n-1}\ \ \ldots\ \ \bar{x}_0x_1\cdots x_{n-1}\ \ x_0x_1\cdots x_{n-1})^T,\\
&\mathbf{A} = (a_{ij})_{k\times 2^n},\ a_{ij} \in \{0,1\},\\
&i\ =\ 0,\ 1,\ \ldots, k-1, \quad j = 0, 1, \ldots, 2^n - 1,
\end{aligned}$$

and the operation $\cdot$ between **A** and U shows that

$$\begin{aligned}
&y_i = \cup_{j=0}^{2^n-1} a_{ij} \cdot u_j \qquad (\cup \text{ and } \cdot \text{ mean logical OR and AND, respectively}),\\
&\quad y_i: \text{ canonical sum of minterms indicated by the } i\text{-th row elements of } A\\
&\quad \text{where } i = 0, 1, \ldots, k-1.
\end{aligned}$$

In this case, Y can also be expressed in a Reed-Muller canonic expansion, which is an easily testable realization [REDD72a, REDD72b].

Checker Implementation Compared to the conventional parity prediction checkers, the new checker can be systematically implemented by using the matrix **A**. We consider a prediction matrix **K**, a $k \times 2^n$ matrix, having the same size as that of matrix **A**, and obtain a prediction function whose output is I with k tuples:

$$I = \overline{\mathbf{K} \cdot U}. \tag{12.2}$$

By changing the elements of **K**, any desired prediction function can be obtained. We also generate another function whose output is J with k tuples:

$$J = Y \oplus (\mathbf{A} \oplus \mathbf{K}) \cdot U \tag{12.3}$$

If there exist no errors, then $Y = \mathbf{A} \cdot U$, and J will then be $\mathbf{K} \cdot U$, the complement of I. The circuit **Z** is thus a two-rail comparator of I and J; a mismatch is signaled by the outputs

$$\begin{aligned}
&Z = (z_0, z_1) = \bigcirc\!\!\!\!\!\!\bigwedge_{i=0}^{k-1} (I_i, J_i),\\
&I_i = [I]_i, J_i = [J]_i, \quad i = 0, 1, \ldots, k-1.
\end{aligned} \tag{12.4}$$

The notations used in Eq. (12.4) are as follows:

$\overline{\mathbf{K} \cdot U}$: logical inversion of $\mathbf{K} \cdot U$. This inversion can be applied either to circuit I or to circuit J,

$[Q]_i$: i-th element of vector Q,

$\bigcap_{i=0}^{k-1}{}^{\circledast} (I_i, J_i)$: comparison of k input pairs, (I_0, J_0), (I_1, J_1), $\ldots$, (I_{k-1}, J_{k-1}), which gives output Z as a 1-out-of-2 code only when all k input pairs are 1-out-of-2 codewords,

$\oplus$: XOR operation.

The basic structure of the checker is shown as circuit **CK** in Figure 12.29. When the prediction matrix **K** is equal to **A**, this checker is identical to the complementary duplication checker. This is because circuit **J** does not exist, and hence output Y goes directly to circuit **Z**. On the other hand, circuit **I** performs the function of $\overline{\mathbf{A} \cdot U}$, which is complement of the original circuit output Y shown in Eq. (12.1). Hence this checker includes the complementary duplication checker as a special case.

The checker has an input space (X, Y) and an output space Z. The error-free output of circuit **L** is defined as Y_0, meaning $Y_0 = Y(X, \lambda)$, $X \in \mathbf{N}$. The input and the output codespaces of the prediction checkers are defined and shown in Figure 12.20b.

Theorem 12.7 *The circuit defined by Eqs. (12.2) through (12.4) is code disjoint.*

Proof If circuit **CK** has input codeword $(X,\ Y_0) \in \mathbf{S}$, it is apparent that this circuit has an output $Z \in \mathbf{B} = \{10, 01\}$. On the other hand, if circuit **CK** has an input noncodeword

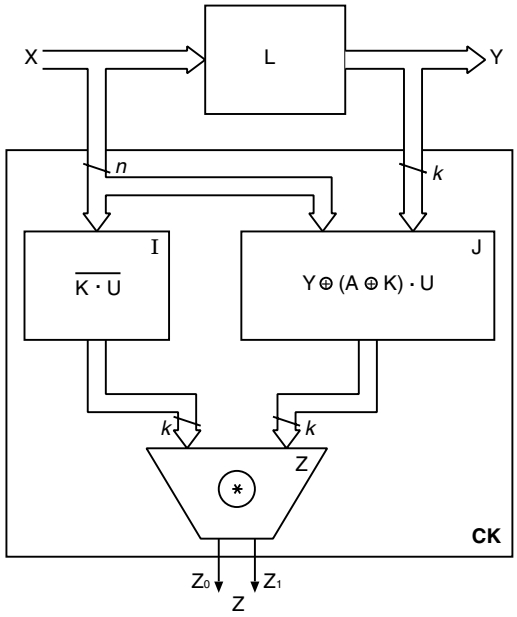


Figure 12.29 *Basic structure of generalized prediction checker.* Source: *[FUJI87a]. © 1987 IEEE.*

$(X, Y') \notin \mathbf{S}$, where $Y' = Y_0 \oplus E$ and E is a nonzero error vector, the following relation holds from Eq. (12.3):

$$\begin{aligned} J &= Y' \oplus (\mathbf{A} \oplus \mathbf{K}) \cdot U = (Y_0 \oplus E) \oplus (\mathbf{A} \oplus \mathbf{K}) \cdot U \\ &= E \oplus \mathbf{K} \cdot U. \quad (\because Y_0 = \mathbf{A} \cdot U). \end{aligned}$$

If $E \neq 0$, not all pairs of (I_i, J_i), $i = 0,\ 1,\ \ldots,\ k-1$, are 1-out-of-2 code vectors. That is, at least one pair is not equal to a 1-out-of-2 code vector, meaning (0, 0) or (1, 1). Hence, by the properties of circuit $\mathbf{Z}$, the output of circuit $\mathbf{CK}$, (z_0, z_1) is included in $\Omega_z - \mathbf{B}$. In other words, the noncodeword inputs are always mapped to the noncodeword outputs.

Q.E.D.

Here an encoding matrix is introduced to the checker. This matrix gives a simplified organization of the checker, thereby allowing circuit $\mathbf{Z}$ to make comparisons of a reduced number of (I_i, J_i) pairs, less than or equal to k pairs; it also gives expression of various error detection capabilities of the checker. The matrix is an $r \times k$ matrix denoted as $\mathbf{H}$, where $r \leq k$. Using this $\mathbf{H}$, we can express the checker shown in Eqs. (12.2) through (12.4) as follows:

$$\text{Circuit } \mathbf{I}: \ I = \overline{(\mathbf{H} \cdot \mathbf{K}) \cdot U}, \tag{12.5}$$

$$\text{Circuit } \mathbf{J}: \ J = \mathbf{H} \cdot Y \oplus \mathbf{H} \cdot (\mathbf{A} \oplus \mathbf{K}) \cdot U, \tag{12.6}$$

$$\text{Circuit } \mathbf{Z}: \ Z = (z_0, z_1) = \bigcap_{i=0}^{r-1}{}^{\circledast} (I_i, J_i). \tag{12.7}$$

In this checker circuit $\mathbf{Z}$ compares r ($\leq k$) pairs. This is shown in Figure 12.30. Circuit $\mathbf{J}$ is composed of circuits $\mathbf{J}'$, $\mathbf{J}''$, and $\mathbf{J}'''$, also shown in this figure. The checker is called a

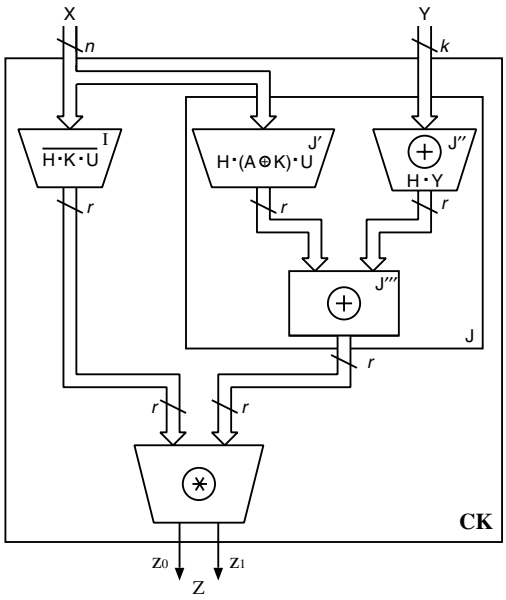


Figure 12.30 *Generalized prediction checker.* Source: *[FUJI87a]. © 1987 IEEE.*

generalized prediction checker (GPC) because it can predict any logic function and has a wide range of error detection capabilities.

In the GPC we newly define the input noncodespace.

Definition 12.13 The input noncodespace of the checker shown in Figure 12.30 is defined by $\Omega_y - \mathbf{S} = \{(X, Y)|\mathbf{H} \cdot Y \neq \mathbf{H} \cdot Y_0\}$, where Y_0 is an error-free output of the circuit under check. □

Theorem 12.8 *The circuit defined by Eqs. (12.5) through (12.7) is code disjoint for input noncodespace* $\{(X, Y)|\boldsymbol{H} \cdot Y \neq \boldsymbol{H} \cdot Y_0\}$ *and output noncodespace {(0, 0), (1, 1)}.*

The proof of this theorem is straightforward and therefore omitted.

Theorem 12.9 *The error detection ability of this checker is determined by matrix* $\boldsymbol{H}$.

Proof In general, the checker's error detection ability can be expressed in terms of syndrome F. When error E exists and output Y is expressed as $Y = Y_0 \oplus E$, then the syndrome F of the checker can be expressed as follows:

$$\begin{aligned} F &= \bar{I} \oplus J = \mathbf{H} \cdot \mathbf{K} \cdot U \oplus \{\mathbf{H} \cdot (Y_0 \oplus E) \oplus \mathbf{H} \cdot (\mathbf{A} \oplus \mathbf{K}) \cdot U\} \\ &= \mathbf{H} \cdot E. \quad (\because\ Y_0 = \mathbf{A} \cdot U). \end{aligned}$$

Errors can be detected only when $F \neq 0$. This means that F is determined directly by $\mathbf{H}$. Hence the error detection ability of this checker is determined by $\mathbf{H}$. Q.E.D.

We can use $\mathbf{H}$ matrices of various error detecting codes, such as simple parity-check codes, double-error detecting codes (i.e., SEC codes), triple-error detecting codes (i.e., SEC-DED codes), and so forth.

Example 12.5 [FUJI87a]

Let us implement the GPC for the circuit expressed by the following 4-input, 3-output logic function. The outputs of the circuit, y_0, y_1, and y_2, are expressed as

$$\begin{aligned} y_0 &= \bar{x}_0(x_1 + \bar{x}_2 x_3), \\ y_1 &= \bar{x}_0 \bar{x}_1 (x_2 + x_3), \\ y_2 &= x_0 + x_1 + x_2 + x_3. \end{aligned}$$

Matrix $\mathbf{A}$ and vectors U and Y can be expressed as

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

$$U = (\bar{x}_0\bar{x}_1\bar{x}_2\bar{x}_3 \quad x_0\bar{x}_1\bar{x}_2\bar{x}_3 \quad \ldots \quad \bar{x}_0 x_1 x_2 x_3 \quad x_0 x_1 x_2 x_3)^T,$$

$$Y = (y_0\ y_1\ y_2)^T = \mathbf{A} \cdot U.$$

Let the prediction functions in this example be AND, OR, and the parity function of the inputs for y_0, y_1 and y_2, respectively. In this case, **K** can be expressed as follows:

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

That is, the prediction function $x_0x_1x_2x_3$ is adopted for y_0, $x_0 + x_1 + x_2 + x_3$ for y_1, and $x_0 \oplus x_1 \oplus x_2 \oplus x_3$ for y_2.

Next we apply the code whose encoding matrix **H** is expressed as

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

This matrix can detect double errors, such as y_0 and y_1 errors, and y_1 and y_2 errors, as well as detect single and triple errors in Y.

Given the above **K** and **H**, the checking logic can be formulated systematically as follows:

Circuit **I**:

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

$$I = \overline{\mathbf{H} \cdot \mathbf{K} \cdot U} = \begin{bmatrix} \overline{x_0 + x_1 + x_2 + x_3} \\ \overline{x_0 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_0x_1x_2x_3} \end{bmatrix} = \begin{bmatrix} I_0 \\ I_1 \end{bmatrix}.$$

Circuit **J**:

$$\mathbf{H} \cdot (\mathbf{A} \oplus \mathbf{K}) = \begin{bmatrix} 0\,1\,1\,1\,0\,1\,1\,1\,0\,1\,1\,1\,0\,1\,1\,1 \\ 0\,0\,1\,1\,0\,1\,0\,0\,1\,1\,0\,0\,1\,0\,1\,0 \end{bmatrix},$$

$$\begin{aligned} J &= \mathbf{H} \cdot Y \oplus \mathbf{H} \cdot (\mathbf{A} \oplus \mathbf{K}) \cdot U \\ &= \begin{bmatrix} y_1 \oplus (x_o + x_1) \\ y_0 \oplus y_2 \oplus (x_0\bar{x}_1x_2\bar{x}_3 + x_1\bar{x}_2\bar{x}_3 + \bar{x}_0\bar{x}_1x_3 + \bar{x}_0x_2x_3 + \bar{x}_1\bar{x}_2x_3) \end{bmatrix} \\ &= \begin{bmatrix} J_0 \\ J_1 \end{bmatrix}. \end{aligned}$$

Circuit **Z**:

$$Z = (z_0, z_1) = (I_0, J_0) \circledast (I_1, J_1).$$

Operation "$\circledast$" is defined in Lemma 12.3.

Figure 12.31 shows the GPC having the capability of detecting some double errors as well as single and triple errors.

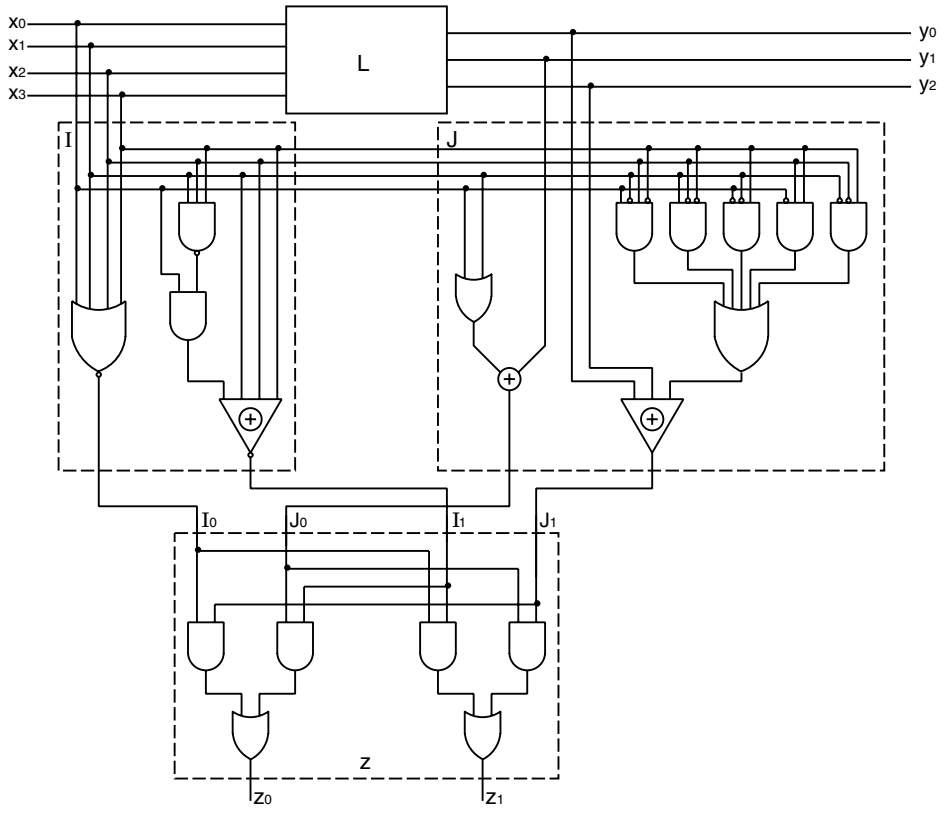


Figure 12.31 *Example of a generalized prediction checker (GPC).* Source: *[FUJI87a]. © 1987 IEEE.*

If $\mathbf{K} = \mathbf{A}$, then the circuits **I** and **J** can be expressed as

$$\text{Circuit } \mathbf{I}: \ I = \overline{\mathbf{H} \cdot \mathbf{K} \cdot U} = \overline{\mathbf{H} \cdot \mathbf{A} \cdot U},$$
$$\text{Circuit } \mathbf{J}: \ J = \mathbf{H} \cdot Y.$$

We can apply the simple parity-check code to the GPC whose encoding matrix is

$$\mathbf{H} = [1\ 1\ 1].$$

Then the circuits **I** and **J** can be expressed as follows.

$$\text{Circuit } \mathbf{I}: \ \overline{\mathbf{H} \cdot \mathbf{A} \cdot U} = \overline{(0101010111010101) \cdot U} = \bar{x}_0(x_1 + x_2 + x_3),$$
$$\text{Circuit } \mathbf{J}: \ y_0 \oplus y_1 \oplus y_2.$$

Figure 12.32 shows this checker, which is equal to the parity prediction checker shown in Figure 12.17.

We now consider the method of reducing the gate amount of this checker under given matrix **H**. The circuit **Z** and the circuit $\mathbf{J}''$ are determined by **H**. We let $\mathbf{K} = \mathbf{A}$. Then the

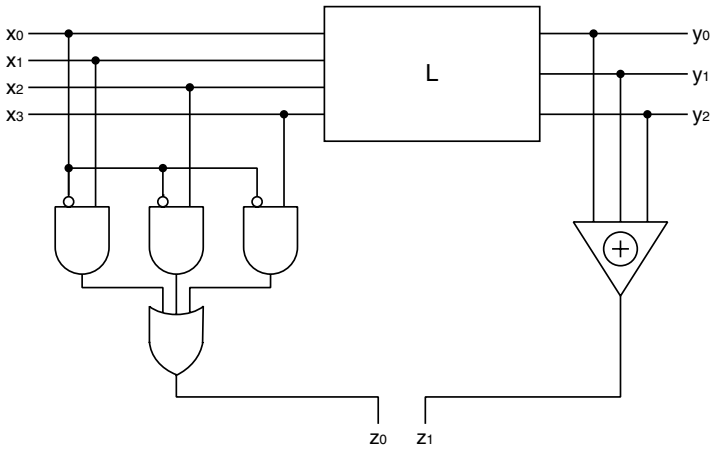


Figure 12.32 *Example of a GPC equal to a parity prediction checker.*

circuit $\mathbf{J}'$ does not exist because $\mathbf{H} \cdot (\mathbf{A} \oplus \mathbf{K}) \cdot U = 0$, so the circuit $\mathbf{J}'''$ is left out. For the circuits $\mathbf{J}'$, $\mathbf{J}''$, and $\mathbf{J}'''$ shown in Figure 12.30, this would reduce the gate amount of the circuit $\mathbf{J}$ to the minimum. In this case the checker is equivalent to the parity-based prediction checker determined by $\mathbf{H}$. In general, for $\mathbf{K} \neq \mathbf{A}$, reducing the gate amount of the checker, especially in the circuits $\mathbf{I}$ and $\mathbf{J}'$, is basically equivalent to a logic minimization problem. For the example circuit shown in Figure 12.31, the gate amount is reduced from 44 to 18 when $\mathbf{K}$ is equal to $\mathbf{A}$ for the same matrix $\mathbf{H}$.

Self-testing GPC The self-testing GPC can detect faults in the checker itself. The checker will operate correctly even if single faults occur in both the circuit $\mathbf{L}$ and the checker $\mathbf{CK}$. To make the checker self-checking, an input v that can take any value in $\{0, 1\}$ is added to the checker as follows:

$$\text{Circuit } \mathbf{I}: \; I_i = [\overline{\mathbf{H} \cdot \mathbf{K} \cdot U}]_i \oplus v, \tag{12.8}$$

$$\text{Circuit } \mathbf{J}: \; J_i = [\overline{\mathbf{H} \cdot \mathbf{Y}}]_i \oplus [\mathbf{H} \cdot (\mathbf{A} \oplus \mathbf{K}) \cdot U]_i \oplus v, \tag{12.9}$$

$$\text{Circuit } \mathbf{Z}: \; Z = (z_0, z_1) = \left[\bigcap_{i=0}^{r-1}{}^{\circledast} (I_i, J_i)\right] \circledast (v, \overline{v}), \tag{12.10}$$

$$v \in \{0, 1\}, \; i = 0, 1, \ldots, r-1.$$

Theorem 12.10 *The circuit determined by Eqs. (12.8) through (12.10) is code disjoint.*

Input v is added to both circuits $\mathbf{I}$ and $\mathbf{J}$, and therefore this does not affect any checking logic defined by Eqs. (12.5) through (12.7). Therefore Theorem 12.10 can be easily proved.

The circuit $\mathbf{Z}$ consists of several two-input two-rail code checkers. According to Theorem 12.6, if these are connected in a cascaded tree structure having $r + 1$ input pairs, meaning (I_0, J_0) (I_1, J_1) $\ldots$ (I_{r-1}, J_{r-1}), and $(v, \overline{v})$, then the circuit $\mathbf{Z}$ is self-testing for single faults, provided that every input does not have constant value and $V = (v, \overline{v})$ can take any value in $\{0, 1\}$ during normal operation.

Circuit **J** is constructed from circuits $\mathbf{J}'$, $\mathbf{J}''$, and $\mathbf{J}'''$ as shown in Figure 12.30, and it is realized by using r subcircuits $\mathbf{J}_0, \mathbf{J}_1, \ldots, \mathbf{J}_{r-1}$. The circuits $\mathbf{J}''$ and $\mathbf{J}'''$ are designed by using exclusive-OR (XOR) gates. From Theorem 12.5, if these circuits are designed in cascaded tree structure having input v that can take any value in $\{0, 1\}$ in normal operation, then this cascaded exclusive-OR circuit is self-testing for single faults. Circuit $\mathbf{J}'$ has input X, same as circuit **L**.

Circuit **I** is also realized by using r subcircuits $\mathbf{I}_0, \mathbf{I}_1, \ldots, \mathbf{I}_{r-1}$, and each subcircuit has input X. The resulting self-testing prediction checker is shown in Figure 12.33.

The control input v is generated from a J-K flip-flop, or a toggle flip-flop, whose output is inverted with every clock input during online error detection. Furthermore the duplicated flip-flops operate simultaneously with the same clock and are connected to circuits **I** and **J**, respectively, as shown in Figure 12.33. Input vector V to the circuit **Z** is also produced from the flip-flop. The checker can detect faults in the cascaded comparator circuits (i.e., the cascaded two-rail code checker) as well as in one of these flip-flops. This also detects input faults to the cascaded comparator circuit **Z**. This is because these faults

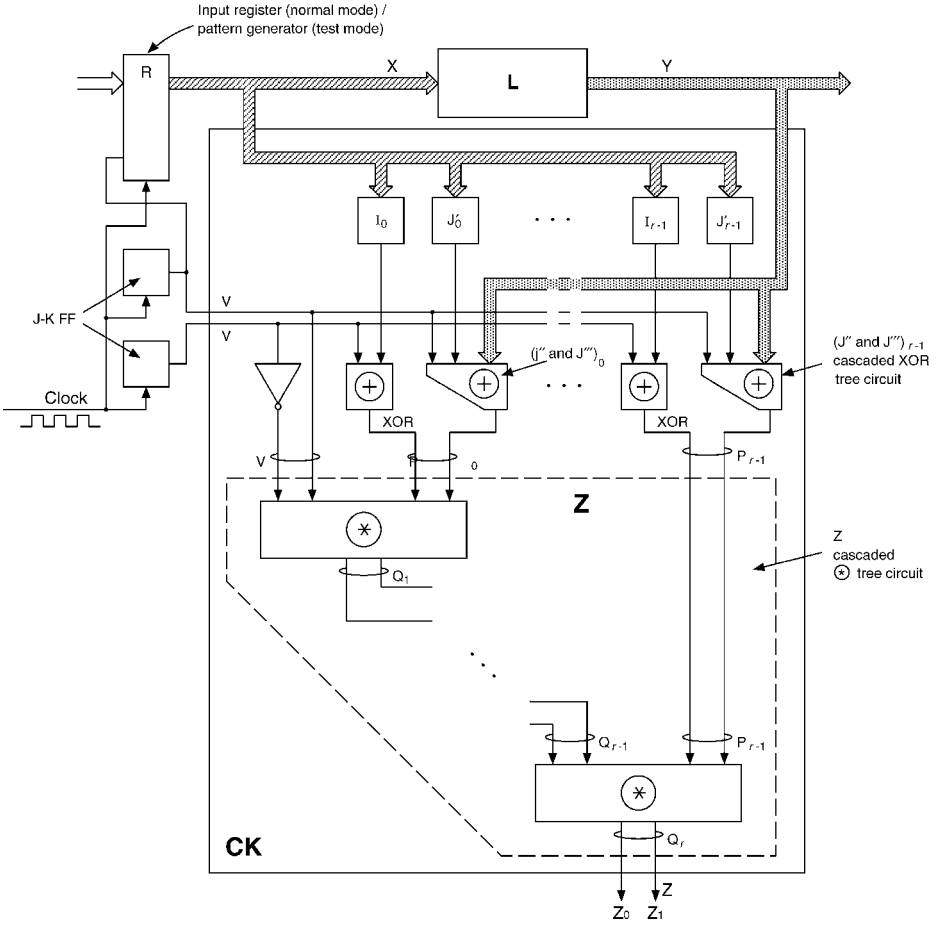


Figure 12.33 *Self-testing generalized prediction checker.* Source: *[FUJI87a]. © 1987 IEEE.*

cause at least one **W** at the circuit **Z** inputs, where **W** is defined in Lemma 12.3, that can finally be detected according to condition 2 of Corollary 12.2.

Theorem 12.11 *The circuit **CK** in Figure 12.33 is a self-testing checker for single faults if input X can take all patterns.*

Proof Circuit **CK** shown in Figure 12.33 is implemented according to Eqs. (12.8) through (12.10). Hence this circuit is code disjoint by Theorem 12.6. Circuits **I** and $\mathbf{J}'$ are self-testing for single faults because input X to these circuits can take all patterns. It is apparent from Theorem 12.5 that the multi-input XOR tree circuit having a cascaded structure and one control input v satisfy the self-testing property for single faults. This is satisfied unless the circuit output Y has a constant value. The cascaded tree circuit **Z** satisfies conditions 1 and 2 of Theorem 12.6, and therefore it is self-testing. Then the noncodeword output of circuits **I** and **J** always cause the noncodeword output, that is, **W**, in the cascaded circuit **Z**. Therefore the circuit **CK** is a self-testing checker. Q.E.D.

Applications to Built-in Testing Online error detection can be combined with off-line functional testing of combinational circuits by using the checker hardware for both purposes [SEDM79, SEDM80a, KHAK82a, FUJI84, FUJI87a]. This built-in testing method takes advantage of the checker's fault detection capability and eliminates the need for sophisticated test systems and precalculated test patterns or signatures [KONE80, WILL82, MACC85, CART86]. It also allows testing at online operation speeds. In Figure 12.34 the circuit **R** is an input register in the normal mode and also a pattern

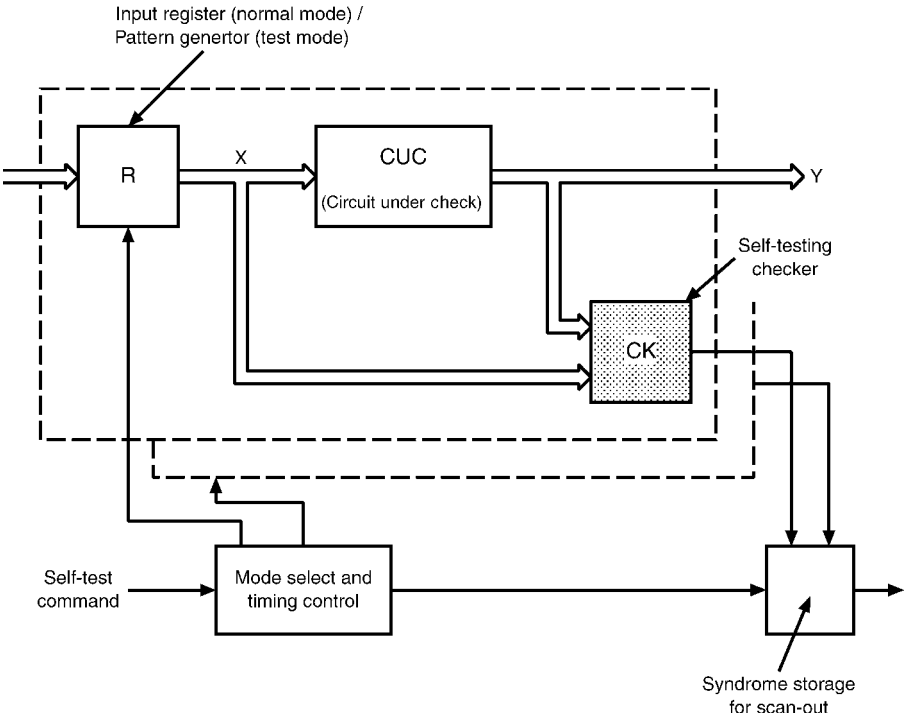


Figure 12.34 *Testing scheme using self-testing checker.* Source: *[FUJI87a]. © 1987 IEEE.*

generator (which generates 2^n patterns for an n-input vector X) in the test mode. One $J-K$ flip-flop or one toggle flip-flop is connected to the lowest level of the pattern generator as shown in Figure 12.33, which realizes an $(n+1)$-bit pattern generator. The faults in circuit **L** and checker **CK** can be detected only when the output of checker $Z \in \{(0,0),\ (1,1)\}$. It is important in this test scheme that if we use such a self-testing checker, *the faults in the checker itself or in the circuit L can be detected in the test mode as well as in the normal mode*. Figure 12.34 shows the built-in testing scheme using a self-testing checker.

12.3 SELF-CHECKING ALU

Considerable research has already been done on how to check functional circuits, such as adder, multiplier, decoder, and error checking / correcting circuits. A variety of checking circuits that use simple error detecting codes and *prediction concepts* have been proposed [CART70, LANG70, AVIZ71, RAO77, WANG79, FUJI81, MAK82]. Here we study an error checking scheme for the arithmetic logic unit (ALU).

We give attention to the relation between faults and errors. In the special circuits such as *fan-out-free circuits*, faults and errors are in one-to-one correspondence. That is, single faults always cause single errors. However, in other circuits, in general, single faults may cause multiple errors. This depends on the circuit's structure. Therefore in this section the faults in the set **F** are defined as those that cause detectable errors. That is, if the checker, or the encoded output of the circuit under check (CUC) has single-error detection capability, then we can detect faults that cause single errors at the CUC output, and **F** consists of only such faults.

Considerable research appears on error detection / correction in ALU. Earlier work on application of simple parity-check codes and residue codes to ALU includes [GARN58, GARN59, GARN66, PETE58, PETE59, BROW60, RAO68b, GADD70]. Extended residue codes such as AN codes [RAO74], *multiresidue codes* [RAO70, RAO71, MAND72], *systematic AN codes* [RAO72], and *byte-error correcting AN codes* [NEUM75], effective parity prediction methods [SELL68], and *alternate data retry (ADR)* methods [SHED78] for application to adder, multiplier, divider, and general logic have been extensively studied [KHOD79, TAKE80, FUJI81, PATE83, FURU83a, FURU83b, FURU83c, HUAN84, TOHM86].

In other important works, *checksum codes* [WAKE76] and *combination codes* using both parity checks and residue checks [RAO77] have been proposed for byte error detection in adders and cost-effective error detection / correction in ALU, respectively. Effective methods using *Reed-Muller codes* [PRAD72a, PRAD72b, PRAD74] and residue codes [MONT72] have been proposed for checking errors in logical operations as well as in arithmetic operations.

Below we study error detection mechanism for adders and ALUs [OBER79] using simple parity-check codes and checksum codes.

12.3.1 Parity-Checked Adder

The adders considered here add two operands A and B together to give a resultant sum S. The addition is done on a bit-by-bit basis. The sum bit s_i for the i-th stage depends not only on the input bits a_i and b_i for that stage but also on the carry c_{i-1} from the previous stage.

The following equations express sum and carry in the i-th stage of an adder:

$$s_i = a_i \oplus b_i \oplus c_{i-1}, \tag{12.11}$$
$$c_i = a_i \cdot b_i + (a_i + b_i) \cdot c_{i-1}, \tag{12.12}$$
$$0 \le i \le n-1,\ c_{-1} = c_{\text{in}},$$
$$A = (a_{n-1}\ \ldots\ a_i\ \ldots\ a_1\ a_0),$$
$$B = (b_{n-1}\ \ldots\ b_i\ \ldots\ b_1\ b_0).$$

The following functions are sometimes convenient to express the adder functions:

Half sum : $H_i = a_i \oplus b_i$.
Generation function : $G_i = a_i \cdot b_i$.
Transmission function : $T_i = a_i + b_i,\ \ 0 \le i \le n-1$.

Using these functions, we can express Eqs. (12.11) and (12.12)

$$\begin{aligned} s_i &= G_i \oplus T_i \oplus c_{n-1} \\ &= H_i \oplus c_{n-1}, \end{aligned} \tag{12.13}$$
$$c_i = G_i + T_i \cdot c_{i-1}. \tag{12.14}$$

Adders are classified into two categories according to how the carries are handled. In the *ripple adder*, shown in Figure 12.35, each carry bit depends on the preceding carry bit. This requires that the overall carry circuitry forms a serial string. The carry equation is given by Eq. (12.14).

It can be seen that addition is slow when the carries ripple through the entire adder. Thus, to speed up the addition process, the carry path can be shortened by using the parallel implementation of Eq. (12.14), called *look-ahead:*

$$\begin{aligned} c_{n-1} &= G_{n-1} + T_{n-1} \cdot c_{n-2} = G_{n-1} + T_{n-1}(G_{n-2} + T_{n-2} \cdot c_{n-3}) = \ldots \\ &= G_{n-1} + T_{n-1}G_{n-2} + T_{n-1}T_{n-2}G_{n-3} + T_{n-1}T_{n-2}T_{n-3}G_{n-4} + \ldots. \end{aligned} \tag{12.15}$$

Equation (12.15) shows that each carry bit does not depend on the previous carry but on the transmission and generation functions. Figure 12.36 shows an implementation of an adder using *carry look-ahead*.

Next we have to refer to the error characteristic of these adders. We call the circuit that satisfies Eq. (12.11) or (12.13) a ''sum circuit,'' and the circuit that satisfies Eq. (12.12) or (12.14) a ''carry circuit.'' So we have the following error characteristics:

1. Sum-circuit errors do not propagate and thus cause single errors only.
2. A carry-bit error always propagates to cause an error in the next sum bit.
3. Any faults in the ripple-carry circuit can cause error bursts, whereas only T_i and G_i failures in the look-ahead circuit can cause error bursts.
4. In a look-ahead carry circuit, all carries are independent of the previous ones.

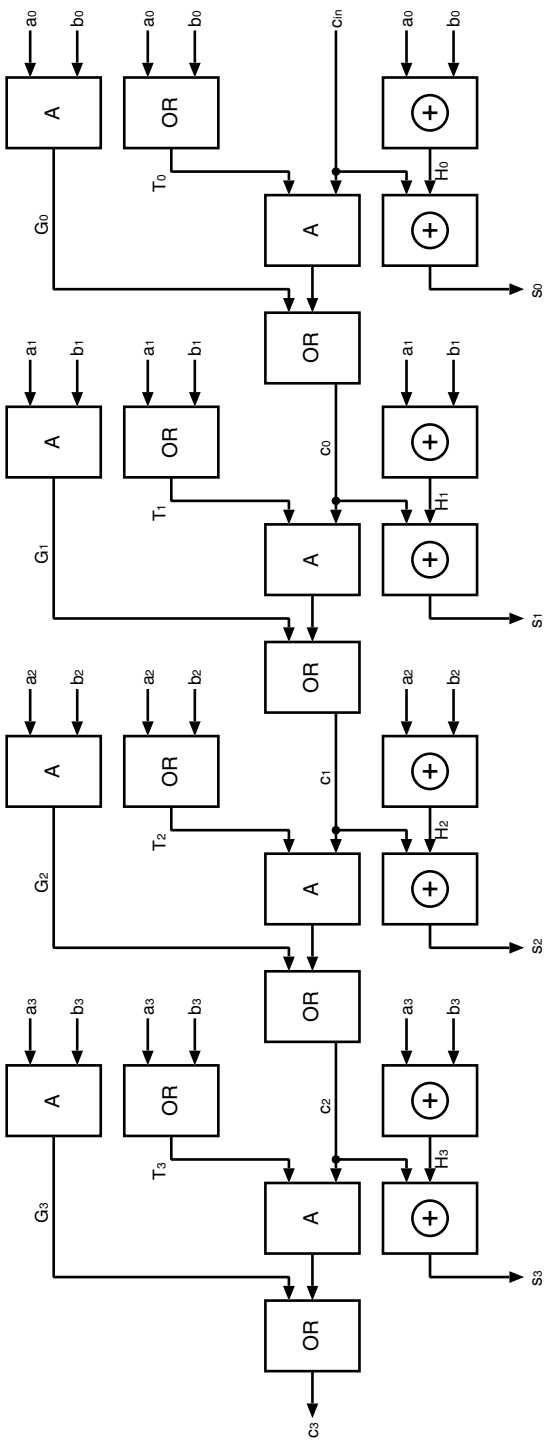


Figure 12.35 *Ripple adder.* Source: *[SELL68]. © 1968 McGraw-Hill Companies.*

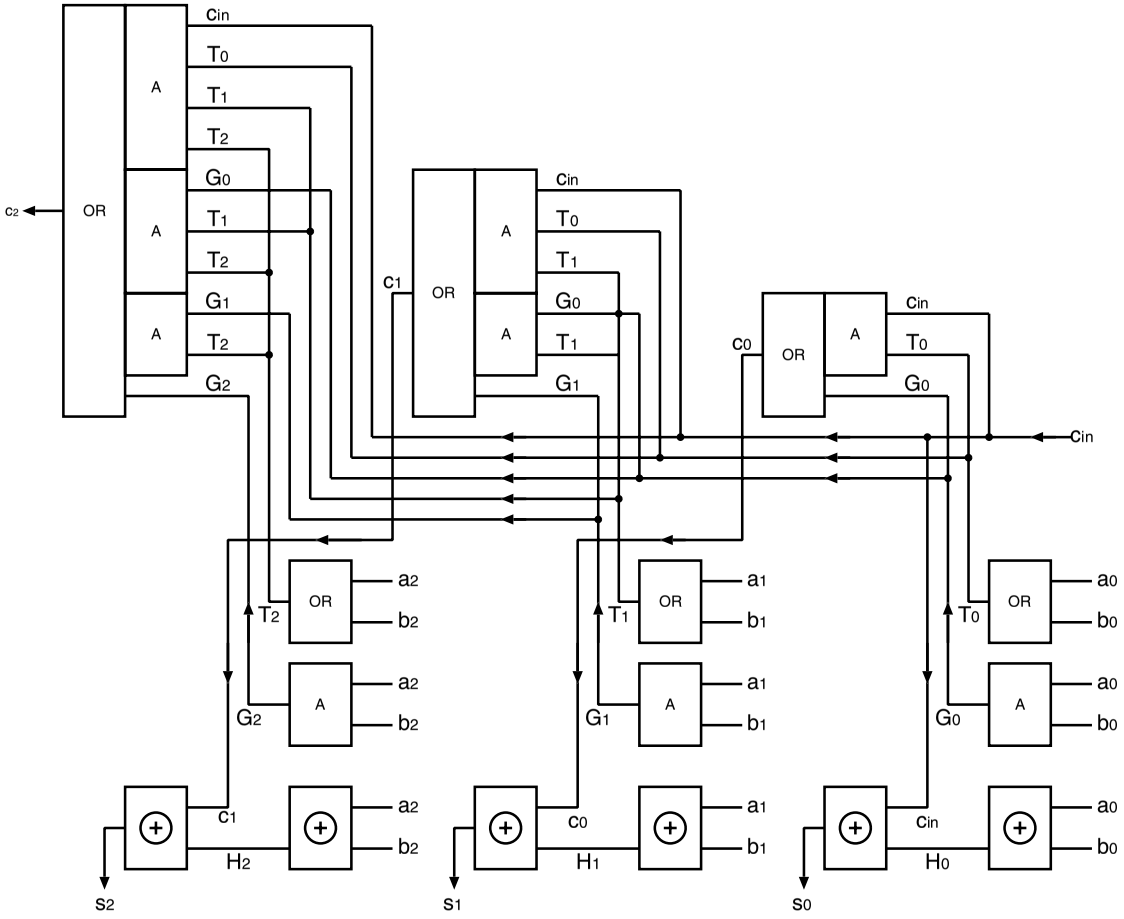


Figure 12.36 *Carry-look-ahead adder.* Source: *[SELL68]. © 1968 McGraw-Hill Companies.*

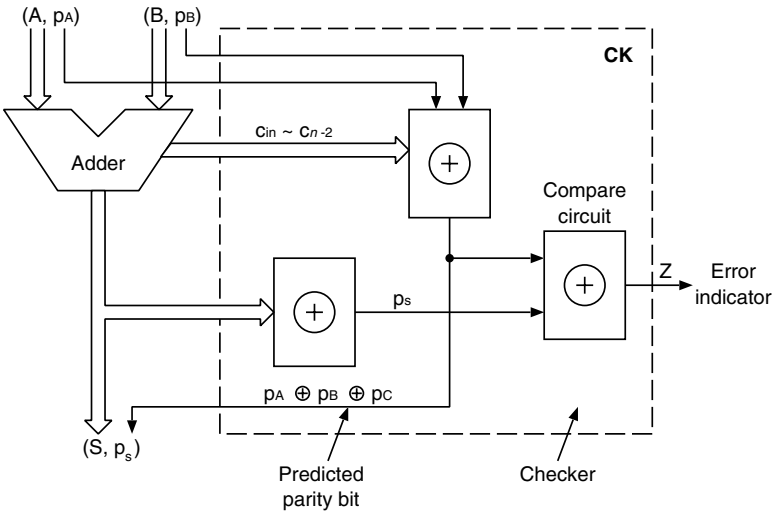


Figure 12.37 *Full-sum parity check.*

Based on these characteristics, we now consider the parity-checked adder. Let p_A and p_B be the parity check bits of the two input operands. The parity of the sum is given by

$$\begin{aligned} p_s &= s_{n-1} \oplus s_{n-2} \oplus \ldots \oplus s_1 \oplus s_0 \\ &= (a_{n-1} \oplus b_{n-1} \oplus c_{n-2}) \oplus (a_{n-2} \oplus b_{n-2} \oplus c_{n-3}) \oplus \ldots \oplus (a_0 \oplus b_0 \oplus c_{\text{in}}) \\ &= (a_{n-1} \oplus a_{n-2} \oplus \ldots \oplus a_0) \oplus (b_{n-1} \oplus b_{n-2} \oplus \ldots \oplus b_0) \oplus (c_{n-2} \oplus c_{n-3} \oplus \ldots \oplus c_{\text{in}}) \\ &= p_A \oplus p_B \oplus p_c. \end{aligned} \tag{12.16}$$

Here p_c denotes the parity of the carries within the adder. The check defined by Eq. (12.16) will be referred to as *the full-sum parity check*. The predicted parity $p_A \oplus p_B \oplus p_C$ is compared with the actual parity of the sum p_s, shown in Figure 12.37.

This checking circuit is equal to the circuit shown in Figure 12.2. That is, the input parities p_A and p_B in the input $X = \{(A, p_A), (B, p_B)\}$, the sum output $S = (s_0\ s_1\ \ldots\ s_{n-1})$, and the internal signals $W = (c_{\text{in}}, c_0, \ldots, c_{n-2})$, which are carry signals, are passed on to the checker. The sum parity is predicted from p_A, p_B, and W, and compared with the parity of the output sum.

Example 12.6

Let inputs A and B be (0010) and (0110), respectively. Hence input parities are $p_A = 1$ and $p_B = 0$. Assume that the output sum S equals (1010). Figure 12.38 shows how to detect sum bit error.

In the checking method a problem arises because carry errors are undetectable. This is because every carry error also causes a sum bit error and hence always results in an even number of carry plus sum bit errors.

Use of duplicate carries and a carry-dependent sum adder have been proposed to overcome this problem [HSIA63, SELL68]. An example of the method of *duplicate carry*

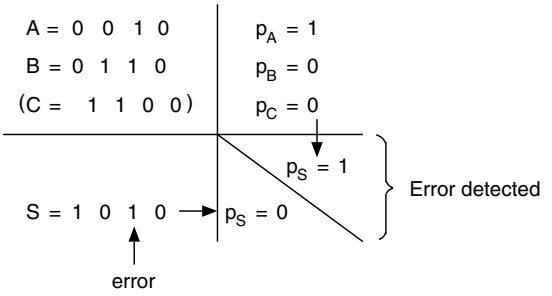


Figure 12.38 *Parity checking for addition.*

with parity check is shown in Figure 12.39. In this method the carry generation circuit is duplicated and p_{cd} is generated from these duplicated carry bits. Therefore the parity check performed by $p_s = p_A \oplus p_B \oplus p_{cd}$ in a ripple adder can detect all carry errors caused by a single fault and in a look-ahead adder can detect all carry errors caused by an error in T_i or G_i.

The next method is the *carry-dependent sum adder* [HSIA63] shown in Figure 12.40.

If a carry error could be made to cause an odd error burst, then parity checking alone could be used for complete adder checking. The adder that satisfies this requirement is called a carry-dependent sum adder. An even error burst c_i, s_{i+1}, ..., c_{i+t-1}, s_{i+t} caused by a carry error in c_i can be made "odd" if the carry bit error in c_i also causes the corresponding sum bit s_i to be in error. This can be satisfied by introducing the following equation relating to the sum bit s_i:

$$s_i = f_i \oplus c_i, \tag{12.17}$$

where f_i is a function of a_i, b_i, and c_{i-1}. This function f_i can be derived from the following table:

a_i	b_i	c_{i-1}	s_i	c_i	$f_i = s_i \oplus c_i$
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	0	1	1
1	0	0	1	0	1
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	1	0

From this table f_i can be expressed as the following equation:

$$\begin{aligned} f_i &= \overline{a_i b_i c_{i-1} + \overline{a}_i \overline{b}_i \overline{c}_{i-1}} \\ &= \overline{G_i c_{i-1} + \overline{T}_i \overline{c}_{i-1}}. \end{aligned}$$

Therefore Eq. (12.17) becomes

$$s_i = G_i c_{i-1} c_i + \overline{G}_i c_{i-1} \overline{c}_i + T_i \overline{c}_{i-1} \overline{c}_i + \overline{T}_i \overline{c}_{i-1} c_i.$$

This type of parity-checked adder is shown in Figure 12.40.

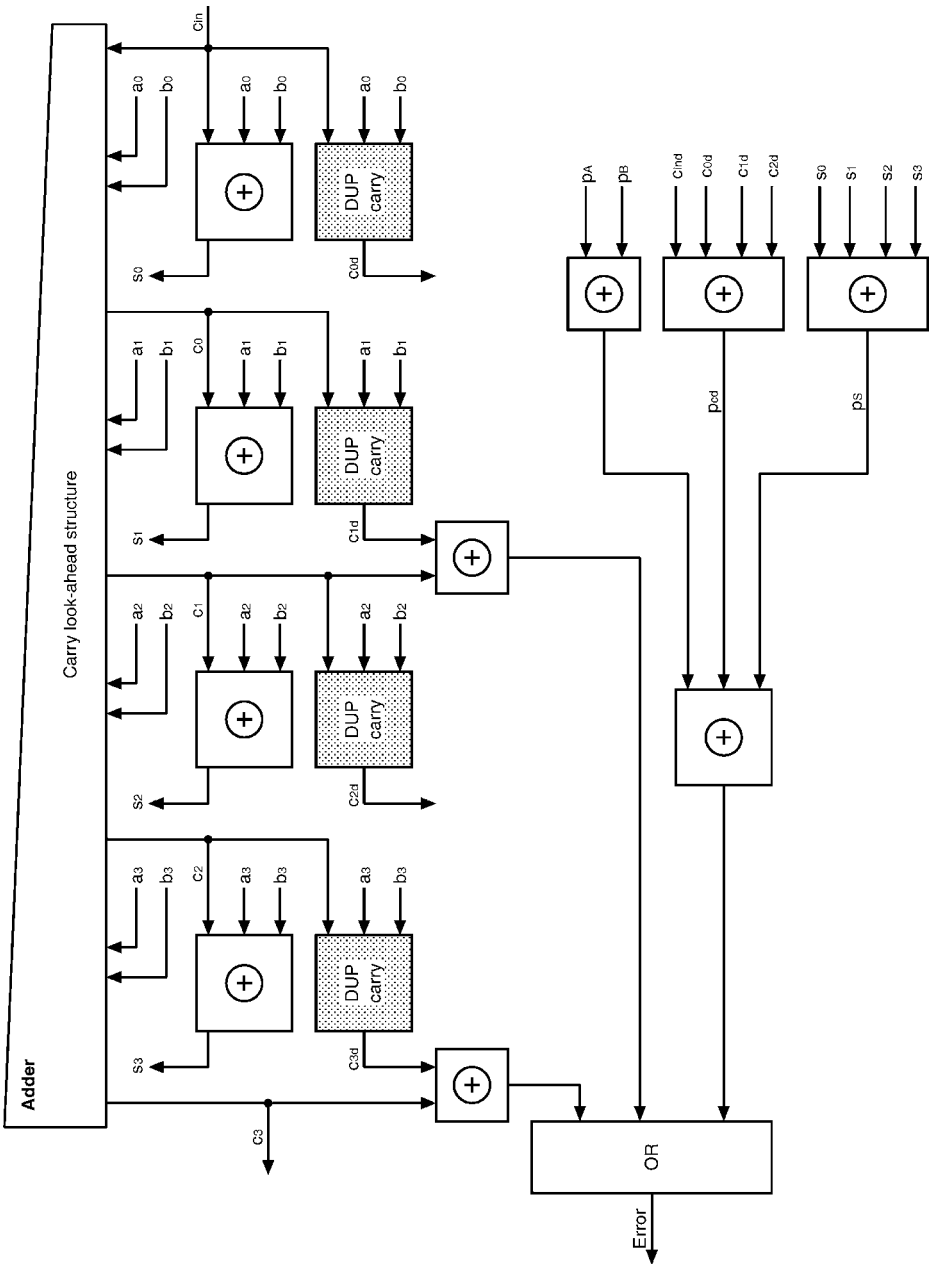


Figure 12.39 *Duplicate carry with parity check applied to carry look-ahead adder.* Source: *[SELL68]. © 1968 McGraw-Hill Companies.*

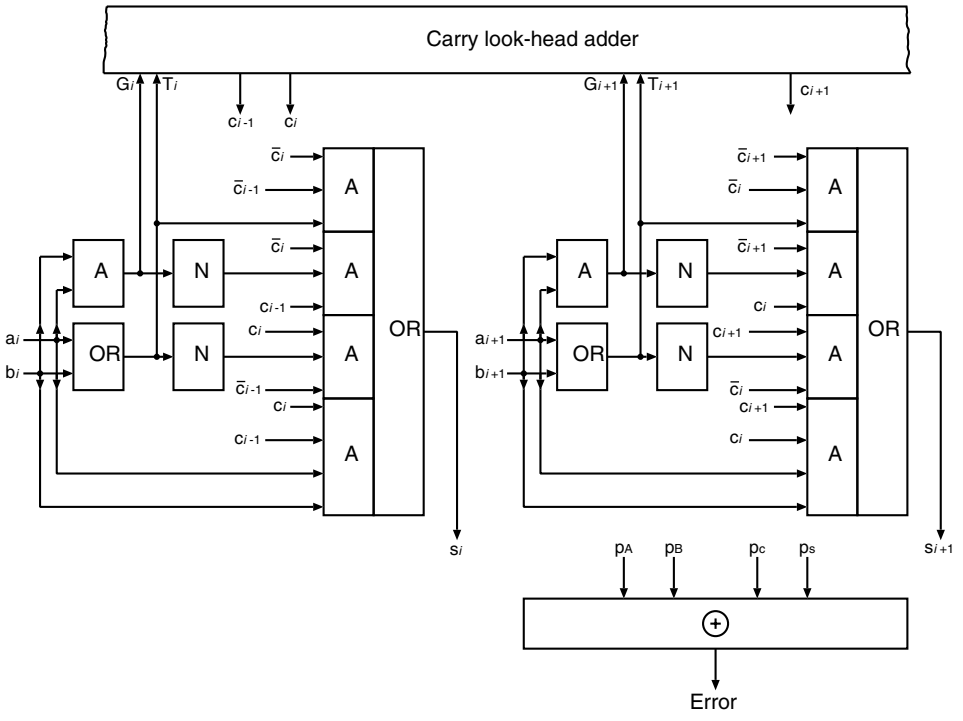


Figure 12.40 *Carry-dependent sum adder.* Source: *[SELL68]. © 1968 McGraw-Hill Book Company.*

Let us consider the case where an error exists in the input but not in the adder. From Eq. (12.16) we can predict the parity p_s of the sum output S. Figure 12.41 shows the circuit whose output is encoded in a simple parity-check code (S, p_s).

If a single error is in (A, p_A) or (B, p_B), where p_A and p_B are parity bits of the operands A and B, respectively, and we assume there are no faults in circuit **G** shown in Figure 12.41, then a single input error is always propagated to the output. This is explained as follows: If there is an error in p_A or p_B, this error is always propagated to the output parity p_s, and not

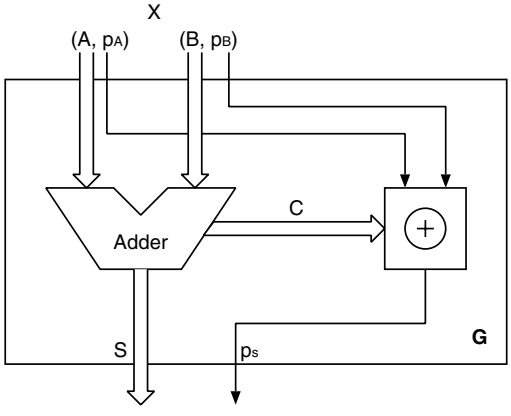


Figure 12.41 *Code-disjoint adder.*

to S. Therefore the output (S, p_s) is a noncodeword. Next we assume that an error exists in A or B. Without loss of generality we assume that an error exists in A. Let the i-th bit in A (i.e., a_i) be in error. Then this error is always propagated to s_i because s_i is a linear function of a_i (i.e., $s_i = a_i \oplus b_i \oplus c_{i-1}$). On the other hand, the error is sometimes propagated and other times not propagated to carry bit c_i. This depends on the value of b_i and c_{i-1}. If this error is propagated to c_i, then this is always propagated to sum output s_{i+1} of the next stage. Similarly, if the error is propagated to the carry of another stage (e.g., c_{i+t}), then s_{i+t+1} is always in error. From this, the single input error can be propagated to $s_i,\ (c_i, s_{i+1}),\ \ldots,\ (c_{i+t}, s_{i+t+1}), \ldots$. The result is an odd number of errors in the output sum and carry bits. This means that if an even number of errors occurs in sum, an odd number of errors exists in carries, and vice versa. Therefore the output $(S,\ p_s)$ will be a noncodeword. We assume that the input noncodeword space is defined as

$$\Omega_x - \mathbf{N} = \{(A, p_A),\ (B, p_B) \mid (A, p_A) \text{ or } (B, p_B) \text{ includes single error}\}.$$

Then circuit $\mathbf{G}$ always maps a noncodeword input to a noncodeword output. Therefore we have proved the following theorem.

Theorem 12.12 *The circuit* $\boldsymbol{G}$ *shown in Figure 12.41 is code disjoint for single errors in the input space.*

12.3.2 Addition with Checksum Codes

Checksum codes, also called *digit parity* [GARN58], can be used to detect single-byte errors. A checksum code is a set of vectors of $n + 1$ symbols from the set Z_q (integers mod q). Each vector has a component, called a check symbol, that equals sum mod q of the other components, called information symbols in the vector.

Definition 12.14 [WAKE76] A *checksum code* is the set

$$\left\{(x_c\ x_{n-1}\ \ldots\ x_0) \mid (x_c,\ x_{n-1},\ \ldots,\ x_1,\ x_0 \in Z_q), \quad \text{and} \quad \left(x_c = \sum_{0 \le i \le n-1} x_i \bmod q\right)\right\}.$$

□

By this definition, it is apparent that the minimum Hamming distance of a checksum code is two. Of particular interest are the codes where q equals 2^b for some integer b greater than 1. If $b = 1$, it is an even parity code. In these codes each symbol from Z_{2^b} may be encoded as a byte of b bits. Hence each codeword has $n \cdot b$ information bits and b check bits. All errors confined to a single b-bit byte are detected.

We now consider the addition modulo 2^b of vectors with any number of components from Z_{2^b}. In this self-checking adder we can detect single-byte errors. Therefore any faults in the adder that cause single-byte errors can be detected. These faults belong to the fault set $\mathbf{F}$.

Let two codewords, namely input vectors, of the checksum codes have the form

$$A = (a_c\ a_{n-1}\ a_{n-2}\ \ldots\ a_0),$$
$$B = (b_c\ b_{n-1}\ b_{n-2}\ \ldots\ b_0),$$

where a_c and b_c are the check symbols, and $(a_{n-1}\ a_{n-2}\ \ldots\ a_0)$ and $(b_{n-1}\ b_{n-2}\ \ldots\ b_0)$ are the information parts,

$$A_d \equiv (a_{n-1}\ a_{n-2}\ \ldots\ a_0),$$
$$B_d \equiv (b_{n-1}\ b_{n-2}\ \ldots\ b_0).$$

The information parts A_d and B_d are considered to be the binary representation of the integers, expressed as $[A_d]$, and $[B_d]$, respectively, such that

$$[A_d] = \sum_{0 \le i \le n-1} a_i 2^{bi},$$
$$[B_d] = \sum_{0 \le i \le n-1} b_i 2^{bi}.$$

Ordinary addition of the information parts of codewords is expressed as

$$\begin{aligned} S_d &= A_d + B_d + C, \\ [S_d] &= [A_d] + [B_d] \mod M, \\ C &= (c_{n-2}\ \ldots\ c_0\ c_{\text{in}}), \\ c_i &= \begin{cases} 0 & \text{if } a_i + b_i + c_{i-1} < 2^b, \\ 1 & \text{if } a_i + b_i + c_{i-1} \ge 2^b, \end{cases} \\ & 0 \le i \le n-2,\ c_{-1} = c_{\text{in}}. \end{aligned}$$

In the equations above, M equals 2^{bn} and c_{in} equals 0 for two's complement addition; M equals $2^{bn} - 1$ and c_{in} equals c_{n-1} for one's complement addition.

The check symbol of the sum of two information parts is equal to the result of an addition of the obtained sum symbols:

$$s_c = \sum_{0 \le i \le n-1} s_i \mod 2^b. \tag{12.18}$$

On the other hand, s_c can be predicted by using the check symbols of the inputs and of the carry as

$$s_c = a_c + b_c + c_c \mod 2^b. \tag{12.19}$$

Here

$$c_c = \sum_{0 \le i \le n-1} c_{i-1} \mod 2^b,$$

where $c_{-1} = c_{\text{in}}$.

Example 12.7

Let $n = 4, b = 3, A = (7\ 2\ 2\ 4\ 7), B = (5\ 3\ 3\ 4\ 3)$, and $M = 2^{12}$ (two's complement). Then $C = (0\ 1\ 1\ 0)$ and $S_d = (2\ 2\ 4\ 7) + (3\ 3\ 4\ 3) + (0\ 1\ 1\ 0) = (5\ 6\ 1\ 2)$, from

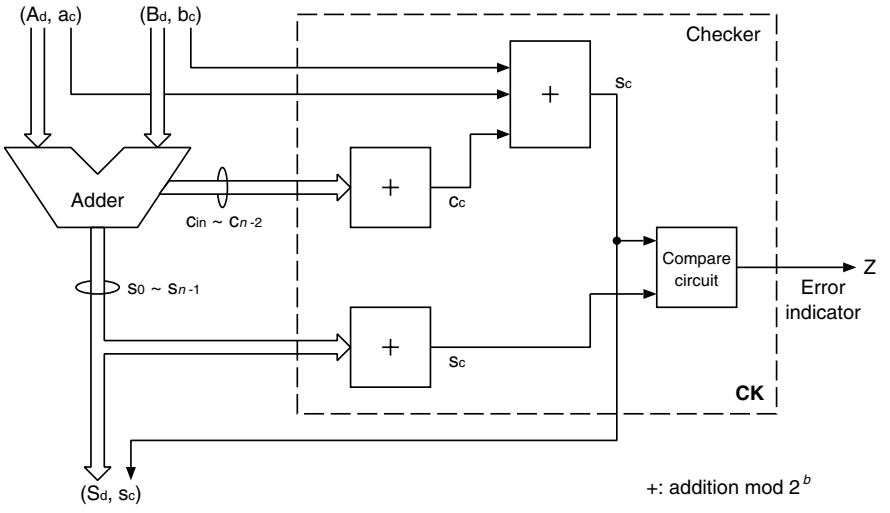


Figure 12.42 *Check symbol prediction checker.*

which we get $s_c = (s_3 + s_2 + s_1 + s_0) \bmod 2^3 = 6$. On the other hand, from Eq. (12.19) for $a_c = 7,\ b_c = 5$, and $c_c = 2$, s_c can be predicted as $s_c = (a_c + b_c + c_c) \bmod 2^3 = 6$.

As with the full-sum parity check, the obtained check symbols from Eqs. (12.18) and (12.19) are compared, and the result is shown as an error indication (Figure 12.42). This checker belongs to the class of prediction checkers. Like result of Theorem 12.12 the circuit including the adder and the check symbol prediction circuit is code disjoint for single symbol errors in the input.

The checker has the same problem as that found in the full-sum parity check. That is, faults in the carry generation circuit are not detected because they produce compensating errors in both s_c and c_c, and although the output is incorrect, there is no error indication. This problem can also be overcome by using the *duplicated carry logic*, as stated before. This checking scheme can be effectively applied to the adder consisting of byte-sliced adders, as shown in Figure 12.43 [WAKE76,78].

12.3.3 ALU with Parity-Based Codes

Parity-based codes have proved to be very efficient and cost effective for detecting / correcting errors in the memory and the data transfer circuits in computer systems. If these codes are successfully applied to the arithmetic logic units (ALU), a single code can be used throughout the system. This is attractive because it eliminates the need for self-checking code translators and reduces a number of different types of code checkers required.

In general, however, parity-based codes are not preserved* in logical operations, except exclusive-OR (XOR) and exclusive-NOR (XNOR) operations. The technique given here is

*As typical logic operations, AND and OR operations are not linear. Therefore linear codes cannot be applied to these nonlinear operations. That is, check bits of the output of these nonlinear operations are not determined by modulo-2 addition of the input information.

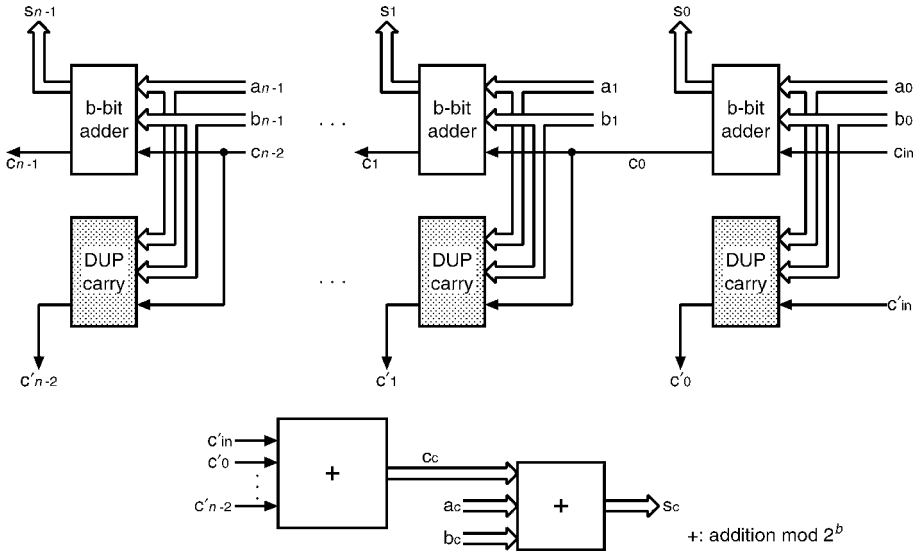


Figure 12.43 *Check symbol prediction circuit for checksum code using byte-sliced adders [WAKE78].*

based on the idea that the result of an arbitrary operation Φ can be linearly transformed into that of an XOR operation. This enables a parity prediction of the output of the operation Φ [FUJI81].

Let two input data words A and B each having k bits be given as follows:

$$A = (a_{k-1}\ a_{k-2}\ \ldots\ a_1\ a_0),$$
$$B = (b_{k-1}\ b_{k-2}\ \ldots\ b_1\ b_0).$$

Parity bits p_A and p_B are generated from the relation

$$p_A = \sum_{i=0}^{k-1}{}^{\oplus} a_i,$$
$$p_B = \sum_{i=0}^{k-1}{}^{\oplus} b_i,$$

where $\sum^{\oplus}$ denotes modulo-2 sum. The circuit has two inputs, (A, p_A) and (B, p_B), and an output, $(Y,\ p_Y)$. In this case output from ALU is defined as follows:

$$Y = (y_{k-1}\ y_{k-2}\ \ldots\ y_1\ y_0)$$

Parity bit p_Y of Y satisfies the following relation:

$$p_Y = \sum_{i=0}^{k-1}{}^{\oplus} y_i.$$

Figure 12.44 shows this circuit model.

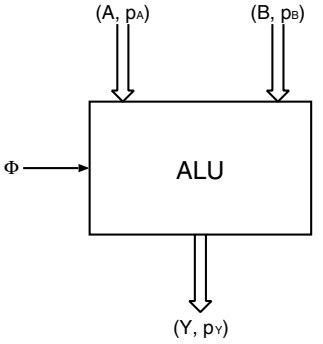


Figure 12.44 *ALU circuit model.*

Parity checking, that is, syndrome S generation, by XOR operations is performed as follows:

$$S = \left(\sum_{i=0}^{k-1}{}^{\oplus} y_i \right) \oplus p_A \oplus p_B,$$

$$y_i = a_i \oplus b_i \ (i = 0,\ 1,\ \ldots,\ k-1),$$

$$\begin{cases} S = 1: & \text{error detection,} \\ S = 0: & \text{error-free.} \end{cases}$$

The i-th transformation function $f_\Phi(y_i)$ expresses the transformation of the i-th result of the ALU operation, y_i, into the ith result of the XOR operation (i.e., $a_i \oplus b_i$) shown in Figure 12.45. Hence the output of the function $f_\Phi(y_i)$ is given as $f_\Phi(y_i) = a_i \oplus b_i$, $i = 0,\ 1,\ \ldots,\ k-1$.

If the function $f_\Phi(y_i)$ satisfies the relation mentioned in the following Theorem 12.13, an error in y_i is propagated to the output of $f_\Phi(y_i)$, and the parity of the ALU output can be predicted.

Definition 12.15 [SELL68] The *Boolean difference* of a function $F = F(x_0,\ x_1,\ \ldots,\ x_i,\ \ldots,\ x_{n-1})$ with respect to x_i is defined as

$$\frac{dF}{dx_i} = F(x_0,\ x_1,\ \ldots,\ x_i,\ \ldots,\ x_{n-1}) \oplus F(x_0,\ x_1,\ \ldots,\ \bar{x}_i,\ \ldots,\ x_{n-1}) \qquad \square$$

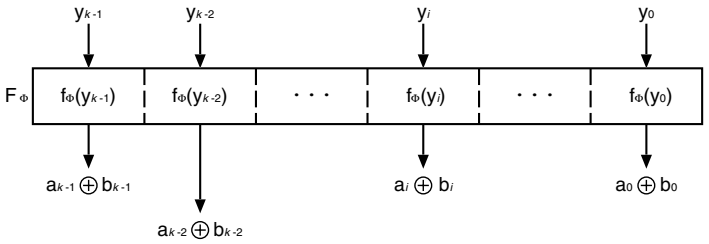


Figure 12.45 *Transformation function.* Source: *[FUJI81]. © 1981 IEICE Japan.*

Theorem 12.13 *If the transformation function $f_\Phi(y_i)$ satisfies the relation*

$$\frac{d(f_\Phi(y_i))}{dy_i} = 1, \qquad i = 0,\ 1,\ \ldots,\ k-1, \tag{12.20}$$

$$f_\Phi(y_i) = a_i \oplus b_i, \tag{12.21}$$

then parity bit of the ALU output, p'_Y, can be predicted as follows:

For $r_i = y_i \oplus a_i \oplus b_i$,

$$p'_Y = p_A \oplus p_B \oplus \left(\sum_{i=0}^{k-1}{}^{\oplus} r_i \right). \tag{12.22}$$

For $r_i = y_i \oplus \overline{a_i \oplus b_i}$,

$$p'_Y = \begin{cases} p_A \oplus p_B \oplus \left(\sum_{i=0}^{k-1}{}^{\oplus} r_i \right) & k\text{: even}, \\ \overline{p_A \oplus p_B} \oplus \left(\sum_{i=0}^{k-1}{}^{\oplus} r_i \right) & k\text{: odd}. \end{cases} \tag{12.23}$$

In order to prove this theorem, the following lemma is prepared.

Lemma 12.5 *Let F be a function of two independent variables Y and R. Then $F = Y \oplus R$ and $\overline{Y \oplus R}$ are the solutions to satisfy the following relation:*

$$\frac{dF}{dY} = 1. \tag{12.24}$$

Proof Assume that $F = Y * R$, where $*$ expresses an arbitrary ALU operation, satisfies Eq. (12.24). Then, by Definition 12.15, the following relation should be satisfied:

$$\frac{dF}{dY} = (1 * R) \oplus (0 * R) = 1. \tag{12.25}$$

The following four cases are the only solutions to satisfy Eq. (12.25):

Case	$1 * R$	$0 * R$	Solutions
(1)	1	0	$\begin{cases} R = 1 & \text{and} \quad * = \text{AND} \\ R = 0 & \text{and} \quad * = \text{OR} \end{cases}$
(2)	0	1	$\begin{cases} R = 1 & \text{and} \quad * = \text{NAND} \\ R = 0 & \text{and} \quad * = \text{NOR} \end{cases}$
(3)	R	$\overline{R}$	$*$ = exclusive-NOR (XNOR)
(4)	$\overline{R}$	R	$*$ = exclusive-OR (XOR)

Since R is a variable, the cases (3) and (4) are the solutions satisfying Eq. (12.25). Hence $F = Y \oplus R$ and $F = \overline{Y \oplus R}$ are the solutions that satisfy Eq. (12.24). Q.E.D.

It is apparent that $F = \overline{Y} * R$, $Y * \overline{R}$, and $\overline{Y} * \overline{R}$, where the $*$ shows an exclusive-OR (XOR) operation or an exclusive-NOR (XNOR) operation, are also solutions that satisfy Eq. (12.24).

Proof of Theorem 12.13 From Lemma 12.5 the following relation satisfies Eq. (12.20):

$$f_\Phi(y_i) = y_i \oplus r_i. \tag{12.26}$$

By Eqs. (12.21) and (12.26), r_i can be obtained such that

$$r_i = y_i \oplus a_i \oplus b_i.$$

Modulo-2 addition with respect to i for both sides of Eqs. (12.21) and (12.26) produces the relations

$$\sum_{i=0}^{k-1}{}^{\oplus} f_\Phi(y_i) = \left(\sum_{i=0}^{k-1}{}^{\oplus} a_i \oplus b_i\right) = p_A \oplus p_B,$$

$$\sum_{i=0}^{k-1}{}^{\oplus} f_\Phi(y_i) = \left(\sum_{i=0}^{k-1}{}^{\oplus} y_i \oplus r_i\right) = \left(\sum_{i=0}^{k-1}{}^{\oplus} y_i\right) \oplus \left(\sum_{i=0}^{k-1}{}^{\oplus} r_i\right).$$

From these equations the predicted parity bit p'_Y is obtained:

$$p'_Y = \sum_{i=0}^{k-1}{}^{\oplus} y_i = p_A \oplus p_B \oplus \left(\sum_{i=0}^{k-1}{}^{\oplus} r_i\right).$$

This satisfies Eq. (12.22).

On the other hand, from Lemma 12.5 the following relation also satisfies Eq. (12.21):

$$f_\Phi(y_i) = \overline{y_i \oplus r_i}.$$

In the same manner, for $r_i = y_i \oplus \overline{a_i \oplus b_i}$, the predicted parity bit p'_Y can be obtained as Eq. (12.23). Q.E.D.

From the definition of the Boolean difference, Eq. (12.20) demonstrates that an error in y_i is always propagated to the output of the function $f_\Phi(y_i)$. Equations (12.20) and (12.21) are important from the point of producing the predicted parity bits as well as giving the condition for propagating an error in y_i to the output of $f_\Phi(y_i)$.

Table 12.3 shows the function r_i for the basic arithmetic and logic operations, where R is defined as follows.

$$R = (r_{k-1}\ r_{k-2}\ \ldots\ r_1\ r_0).$$

Theorem 12.14 describes the parity checking for an arbitrary ALU operation using the predicted parity bit p'_Y shown in Eq. (12.22).

TABLE 12.3 Function r_i

Operation Φ	y_i	r_i	
		$* =$ XOR	$* =$ XNOR
AND	$a_1 \cap b_i$	$a_i \cup b_i$	$\bar{a}_i \cap \bar{b}_i$
OR	$a_i \cup b_i$	$a_i \cap b_i$	$\bar{a}_i \cup \bar{b}_i$
XOR	$a_i \oplus b_i$	0	1
XNOR	$\overline{a_i \oplus b_i}$	1	0
ADD	$a_i + b_i$	c_{i-1}	$\overline{c_{i-1}}$

Source: *[FUJI81]. © 1981 IECE Japan.*
Note: c_{i-1} is a carry bit that enters the position i.

Theorem 12.14 *For inputs A and B, each having k bits, the parity checking (i.e., syndrome generation) for an ALU operation Φ is performed as*

$$\begin{aligned} S_\Phi &= p_Y \oplus p'_Y \\ &= \left(\sum_{i=0}^{k-1}{}^{\oplus} y_i\right) \oplus \left\{\left(\sum_{i=0}^{k-1}{}^{\oplus} r_i\right) \oplus p_A \oplus p_B\right\} \\ &= \left(\sum_{i=0}^{k-1}{}^{\oplus} f_\Phi(y_i)\right) \oplus p_A \oplus p_B, \end{aligned} \tag{12.27}$$

where

$f_\Phi(y_i) = y_i \oplus r_i = a_i \oplus b_i$,
$S_\Phi = 1$: *error detected*,
$S_\Phi = 0$: *error free*.

Figure 12.46 shows the parity-checking scheme for an ALU operation Φ.

Theorem 12.15 *The parity-checking scheme shown in Fig. 12.46 detects any single errors in input A or B, if there exist no faults in both ALU and checker.*

Proof Equation (12.27) can be also written as

$$\begin{aligned} S_\Phi &= \left(\sum_{i=0}^{k-1}{}^{\oplus} f_\Phi(y_i)\right) \oplus p_A \oplus p_B \\ &= \left(\sum_{i=0}^{k-1}{}^{\oplus} a_i \oplus b_i\right) \oplus p_A \oplus p_B. \end{aligned}$$

Therefore we have

$$\frac{dS_\Phi}{da_i} = 1 \quad \text{and} \quad \frac{dS_\Phi}{db_i} = 1.$$

This shows that any single input error can always be detected. Q.E.D.

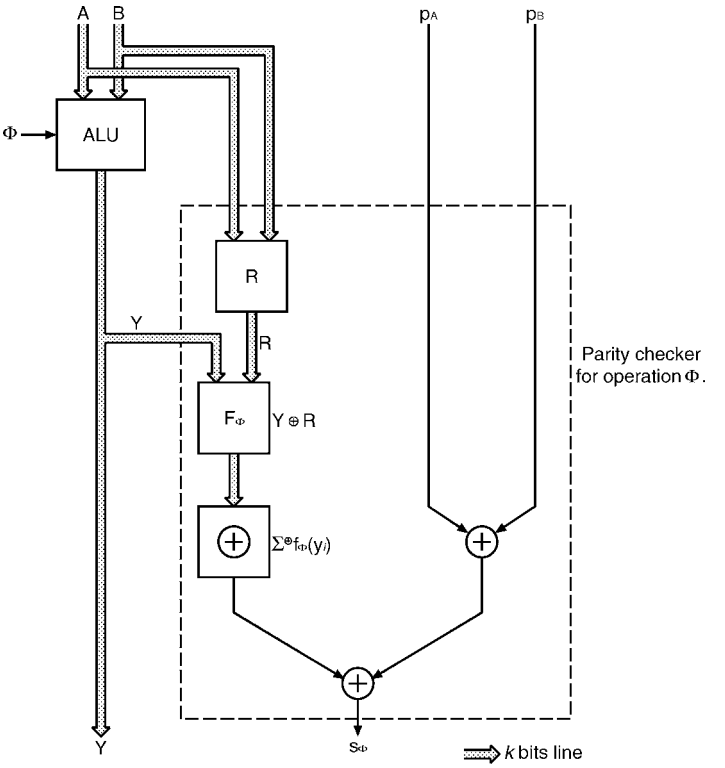


Figure 12.46 *Parity-checking scheme for arbitrary operation Φ.* Source: *[FUJI81]. © 1981 IEICE Japan.*

Next we consider an error correction of ALU operation based on the foregoing principle. Parity checking is performed according to the check group indicated by the **H** matrix row. That is, the check group is defined as a set of input data determined by the row pattern of **H**. There exist $n - k = r$ check bits in the $(n,\ k)$ code

$$\mathbf{H} = [\mathbf{H}_e \,|\, \mathbf{I}_r]_{r \times n}.$$

Here $\mathbf{H}_e$ is an $r \times k$ encoding matrix and $\mathbf{I}_r$ is an $r \times r$ identity matrix. Two input codewords are shown as (A, C_A) and (B, C_B), where C_A and C_B are check-bit vectors, such that

$$C_A = (c_{A,0}\ c_{A,1}\ \ldots\ c_{A,r-1}),$$
$$C_B = (c_{B,0}\ c_{B,1}\ \ldots\ c_{B,r-1}).$$

We also define C such that

$$C = C_A \oplus C_B,$$
$$C = (c_0\ c_1\ \ldots\ c_{r-1}),$$
$$c_i = c_{A,i} \oplus c_{B,i}, \qquad i = 0,\ 1,\ \ldots,\ r-1.$$

The output of ALU is shown as (Y, C_Y), where C_Y is a check-bit vector for output Y:

$$C_Y = (c_{Y,0}\ c_{Y,1}\ \ldots\ c_{Y,r-1}).$$

Recall that an error correction procedure consists of three main steps: (1) syndrome generation, (2) determination of error location (syndrome decoding), and (3) inversion of the erroneous bit.

For syndrome generation for output Y, we have the following sequence of steps:

Step 1. Check-bit generation,

$$C_Y = Y \cdot \mathbf{H}_e^T.$$

Step 2. Check-bit prediction,

$$C_p = R \cdot \mathbf{H}_e^T \oplus C,$$
$$C_p = (c_{p,0}\ c_{p,1}\ \ldots\ c_{p,r-1}).$$

Step 3. Syndrome generation,

$$\begin{aligned} S &= C_Y \oplus C_p \\ &= Y \cdot \mathbf{H}_e^T \oplus (R \cdot \mathbf{H}_e^T \oplus C) \\ &= F_\Phi \cdot \mathbf{H}_e^T \oplus C, \quad \text{where} \quad F_\Phi = Y \oplus R. \end{aligned}$$

The error location is determined from the syndrome, such that

$w(S) = 0$: No error.
$w(S) = 1$: Error in check-bit part C_p.
$w(S) \geq 2$: Error in ALU output Y,

where $w(S)$ means the weight of syndrome S. Location of the erroneous bits, especially the erroneous bits in the output Y, is determined precisely by the column vectors of the $\mathbf{H}$ matrix.

An error pointer $E = (E_Y, E_c)$ specifies the error pattern to be corrected, where E_Y is the output Y error and E_c the check-bit error. The corrected output $(\hat{Y}, \hat{C}_Y)$ is obtained by the error pointer such that

$$\hat{Y} = Y \oplus E_Y,$$
$$\hat{C}_Y = C_p \oplus E_c.$$

Figure 12.47 shows an error correction circuit (**EC**) for ALU. Note particularly that the circuit $\mathbf{EC}_0$ in **EC**, enclosed by the broken line, is the same error decoding circuit as that for the high-speed memories.

Therefore any type of error detecting / correcting parity-based code, that is, any linear code, can be applied to an error detection / correction in ALU operations. In [FUJI81],

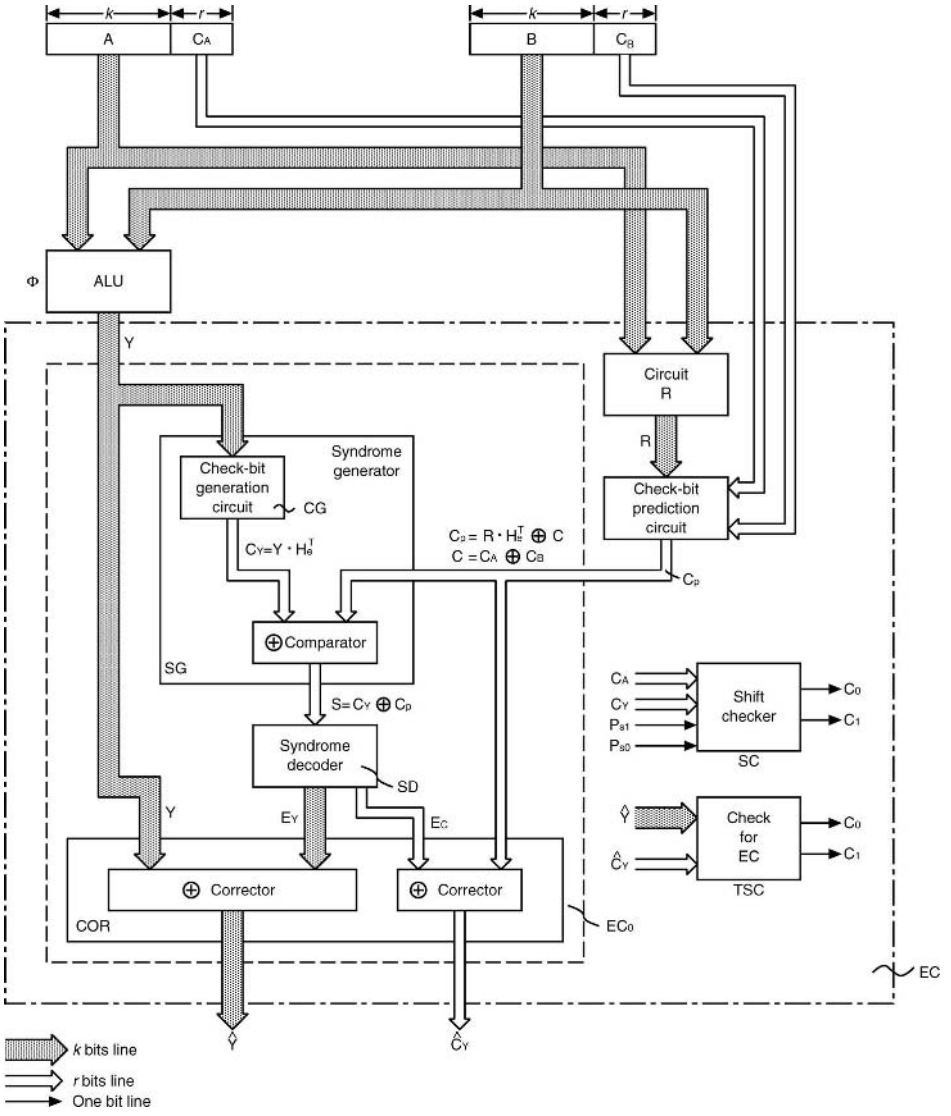


Figure 12.47 *Error correction scheme for ALU using* $(k + r, k)$ *parity-based codes.* Source: *[FUJI81]. © 1981 IEICE Japan.*

where a comparison with the triplication (TMR) is given for total hardware amount, the TMR is about 1.5 times larger than the scheme above using single 4-bit byte error correcting (S4EC) code and 4-bit byte-sliced ALUs for $k = 16$ bits and $r = 8$ bits. Nevertheless, for operational speed and error correction capability, the TMR is superior to this scheme.

12.4 SELF-CHECKING DESIGN FOR COMPUTER SYSTEMS

The original concept for self-checking computers was first presented by [CART68]. The specific design methods for TSC systems or SFS networks were developed by [SMIT78, 83].

From self-checking design point, the code detecting or correcting hard errors as well as transient errors on the spot has proved to be essential not only to special / general purpose computer systems but also to recent high-speed microprocessors.

12.4.1 Coding for Dependable Computer Systems

1. Dependable Special Purpose Systems

STAR Computer A special purpose ultra-reliable computer that makes extensive use of the technique of modularity and standby sparing, as well as error detection methods, is the Jet Propulsion Lab's self-testing and repair (STAR) computer [AVIZ71]. In the STAR computer, low-cost arithmetic error detecting codes are used in the data word, or instruction word. For the byte-organized computer word with four-bit bytes, modulo-15 arithmetic checking is especially effective. All words are encoded as shown in Figure 12.48.

Note in the figure that the 32-bit numeric operand word consists of a 28-bit binary number b and a 4-bit check $C(b)$. The check byte, which is a binary number, has the value

$$C(b) = 15 - |b|_{15},$$

where $|b|_{15}$ means residue modulo-15 of b. The 32-bit instruction word consists of a 12-bit operation code and a 20-bit residue-coded address part. The 16-bit address is encoded in the same residue-check code as the operands. The operation code is divided into three bytes, and each byte is encoded in a 2-out-of-4 code. This code permits each byte to be checked individually.

Electronic Switching Systems In electronic switching systems (ESS), the self-checking hardware has been integrated into the design so that faults are detected during normal system operation [TOY78]. In the microprogram controller, an efficient nonsystematic 4-out-of-8 code is applied to the control information in order to detect all multiple unidirectional errors. Also a TSC 4-out-of-8 code checker is implemented.

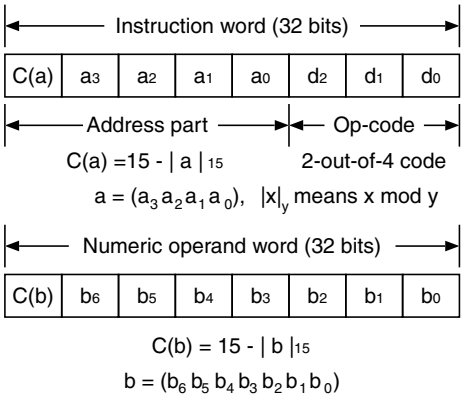


Figure 12.48 *STAR Instruction word and operand word format.* Source: *[AVIZ71]. © 1971 IEEE.*

2. (4, 2) Concept Machine

An important fault-tolerant system concept, that is, (*4, 2*) *concept*, [KROL86], has practically applied to the Philips digital telephone exchange system [GEEL85]. It is very unique that the coding is applied at the system level, not applied to a specific circuit or module. This system consists of four modules. The 16-bit processor is quadrupled while the memory consists of four parts each with half a data word length, namely 8 bits. The information stored in the four memory modules is protected by a single-symbol error correcting code. This code consists of four 8-bit symbols, two of which are information symbols and the other two are check symbols, which is the (4, 2) code. The basic (4, 2) concept architecture is shown in Figure 12.49. In this system the data word has a length of 16 bits. Each module comprises a 16-bit microprocessor (P), a memory (M) with 8-bit data input / output, and a decoder (DEC) of the (4, 2) single-symbol error correcting code. The memory contents are protected by the code, where symbol expresses an 8-bit data. Each symbol is stored in a different module. Four processors process identical 16 bits information and run synchronously. Therefore the system basically tolerates any faults / errors in any one of the four modules.

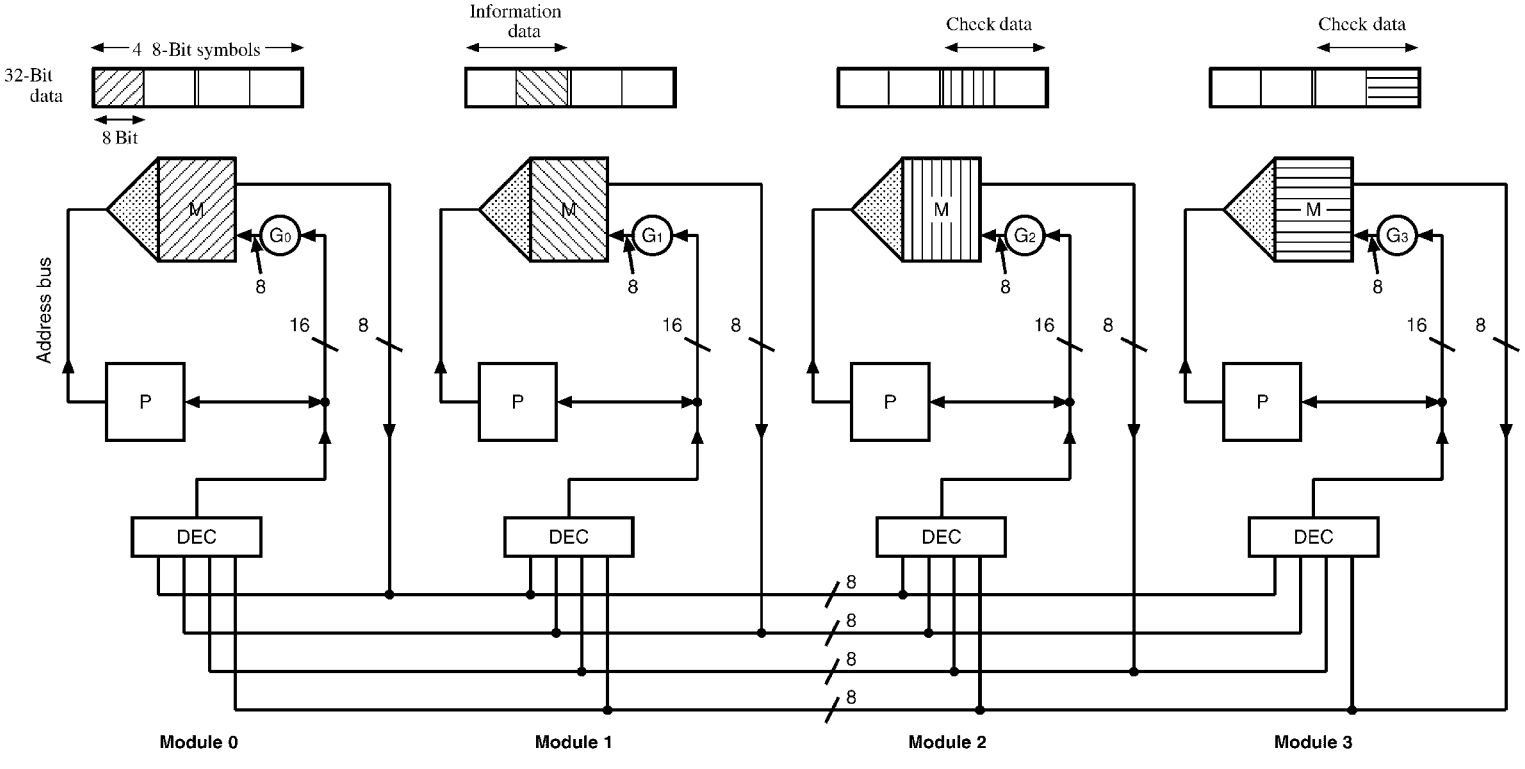
When the information is sent from the processor (P) to the memory (M) in write operation, the 16-bit data word is encoded into 8 bits in each module and written into the memory by way of the circuit G_i, $i = 0,\ 1,\ 2,\ 3$. Note that the hardware implementation in each module differs only in the encoders G_0, G_1, G_2, and G_3. Each encoder receives 16-bit information data, namely two information symbols. Each of the two encoders (e.g., G_2, and G_3 in Figure 12.49) among the four G_i's generates two check symbols and selects one 8-bit check symbol out of the two generated symbols that is finally stored in the corresponding memory M. The remaining two G_i's (e.g., G_0, and G_1 in Figure 12.49) select one 8-bit symbol among the input two information symbols that is stored in the corresponding M.

When the information is transferred from the memory to the processor in the read operation, each module receives at the decoder (DEC) not only its own 8-bit symbol but also the other three symbols transmitted from the other modules. Therefore the decoder in each module receives the complete word of the four 8-bit symbols. If any one of these symbols is in error, then the error is corrected by the decoder (DEC) in each module. In this case, whatever faults occur in one module, these affect only one 8-bit symbol that is also sent to the other modules. So the decoder in each module may contain single-symbol errors that are finally corrected. That is, any hardware faults existed in one module that may cause single-symbol errors do not affect the operation of the total system.

The code adopted is built up from two interleaved codes, each consisting of four 4-bit bytes. The code can correct any single-byte errors and any double-bit errors; that is, the code is an S*b*EC-DEC code as presented in Section 6.3. However, the single-byte errors and double-bit errors cannot be corrected simultaneously. The S4EC-DEC code was designed by an exhaustive computer search and is given in the following **H** matrix:

$$\mathbf{H} = \begin{bmatrix} \mathbf{T}^7 & \mathbf{T}^{11} & \mathbf{I} & \mathbf{0} \\ \mathbf{T}^{11} & \mathbf{T}^7 & \mathbf{0} & \mathbf{I} \end{bmatrix}, \tag{12.28}$$

where **T** is a companion matrix defined by the 4-th degree binary primitive polynomial $\mathbf{g}(x) = x^4 + x + 1$.



P: Processor
M: Memory
DEC: Decoder of the (4, 2) single-symbol error correcting code
G_i: Encoder of the code ($i = 0, 1, 2, 3$)

Figure 12.49 *The (4, 2) concept machine.* Source: *[KROL86]. © 1986 IEEE.*

The total interleaved code has various bit or byte (or symbol) error control capabilities and is at least capable of correcting single 8-bit symbol errors that occur in any one module and double-bit errors that occur in two different modules. More extensive work on these combined byte and bit error control codes is presented in [GILS86a, GILS86b, GILS87].

This concept can be expanded to an $(n,\ k)$ concept, in general, where an $(n,\ k)$ symbol error correcting code is applied [KROL86]. This makes it possible to choose a redundancy ratio between the numbers of processors and the memories, and this way to optimize the total amount of redundancy. In the $(n,\ k)$ concept the processor data are encoded into an n-folded repetition code, but the memory data are encoded into an $(n,\ k)$ symbol error correcting code. That is, a copy of the processor data exists in each module, and each memory of the module contains one symbol of the codeword. However, the symbol size of the $(n,\ k)$ code is k times smaller than that of the n-folded repetition code. The $(n,\ k)$ symbol error correcting code is capable of correcting up to $\lfloor (n-k)/2 \rfloor$ symbols errors and requires a factor of $(n-k)/k$ additional memory hardware.

3. Dependable General Purpose Systems

(1) Self-Checking Computer Design for IBM S/360 General Purpose Computer

The self-checking computer system was first studied for cost-effectiveness by Carter [CART77]. The target machine was constructed with a small number of LSI chip types, about 10. The design objectives were as follows:

1. To make the processing unit (PU) completely checkable and self-testing, and to consider hardware trade-offs.
2. To achieve in the processing unit a retry capability without speed degradation and without significant additional hardware.

It was found that complete checking of a PU is relatively inexpensive (35% of additional hardware over the unchecked PU and 6.5% over the S/360 checked PU, with only one additional new chip type) and that a simple instruction retry could be achieved with one additional chip and with no apparent speed degradation.

Since 1990s the general purpose data-processing systems have adopted a variety of dependable techniques of concurrent error detection, fault isolation, and recovery. High availability has been achieved by minimizing the component failure rate through improvements in the base technologies, and by applying the design techniques that enable hard and soft error detection / correction, recovery and isolation, and component replacement concurrent with system operation.

(2) RAS Design for Recent General Purpose System and Server Machine

Here we take a brief look, from the standpoint of coding, at the dependable design techniques of a recent general purpose system and a server machine.

(a) The IBM Enterprise System/9000. The system design includes a variety of dependable techniques to detect, recover, and isolate failures of circuits and components. Its two points of merit are continuous availability and short repair time [CHEN92]. A big objective of the design of the system is to provide high levels of reliability, availability, and

serviceability (RAS). All system hardware components, including logic, memory, and power, have significantly increased levels of coverage in error detection, fault isolation, error correction, coverage effectiveness, and concurrent maintenance.

The basis for all the hardware fault tolerance and data integrity is a concurrent error detection. As a result the system achieves nearly the error detection effectiveness of logic duplication and comparison but with less than 30% circuit overhead. The additional hardware design accomplishes fault isolation. The built-in hardware error detection and replaceable unit isolation have been proved to isolate faults quickly. In order to achieve this, the key design characteristic has to (1) detect errors during normal operation, (2) capture machine status information at the time of error detection, and (3) isolate the failing unit by analysis of the data captured at the detection of the error. Error detection and retry are used for recovery from logic errors. Concurrent error correction and standby spares are applied to memory fault tolerance.

From coding point of view, the following techniques are applied to this system.

For logic circuits. Concurrent error detection

- Parity checking for data flow registers
- Parity checking for control registers
- Parity prediction checking for transformation logic
- Parity prediction checking for sequential controls
- Decoder output check (*one and only one check*) and invalid logic output checks
- Residue checking for arithmetic functions (e.g., *residue modulo-3 check* for high-speed multipliers)

For semiconductor memories. Error control codes for error correction and detection, retry, and fault isolation

- Level-1 instruction cache and data cache: simple parity-check code and spare word lines for hard fault-tolerance
- Level-2 cache: SEC-DED codes and faulty word line deletion
- Control memory: simple parity-check code and spare word lines for replacing the faulty line
- Main memory: (72, 64) SEC-DED code, spare DRAM chip for replacing faulty chip, memory background scrubbing (mentioned later), and fetch with retry request
- Expanded main memory: (144, 128) DEC-TED code, spare memory chip, and background scrubbing

DEC-TED BCH Codes The (144, 128) double-bit error correcting and triple-bit error detecting (DEC-TED) codes are applied to the expanded memory. The code is obtained by shortening a (255, 239) cyclic BCH code whose generator polynomial contains α and α^3 as roots, where $\alpha \in GF(2^8)$ is a root of binary primitive polynomial of degree 8. In addition the code is modified to have extra property of detecting all single 4-bit symbol errors, which is useful in isolating the memory support logic faults.

Decoding the DEC-TED code is performed as follows [CHEN92]: Assume that there are two errors, at positions X_1 and X_2, and that the syndrome can be represented by two

8-bit vectors, S_1 and S_3, with $S_1 = X_1 + X_2$, and $S_3 = X_1^3 + X_2^3$. Given the syndrome (S_1, S_3), the following relation holds:

$$\begin{aligned} D &= S_1^3 + S_3 \\ &= X_1^3 + X_1^2X_2 + X_1X_2^2 + X_2^3 + X_1^3 + X_2^3 \\ &= X_1^2X_2 + X_1X_2^2 \\ &= X_1X_2(X_1 + X_2) \\ &= S_1^2X_1 + S_1X_1^2. \end{aligned}$$

The expression $S_1^2X_1 + S_1X_1^2$ can be represented by $S_1\mathrm{T}_x$, where T_x is an 8×8 binary matrix uniquely determined by the position X. The double-error decoding algorithm is shown as follows:

Step 1. Compute $D = S_1^3 + S_3$.

Step 2. For each of the 144 codeword position X, compute $\mathrm{Q}_x = S_1\mathrm{T}_x$. The computation of all 144 Q_x values can be carried out in parallel in hardware involving XOR circuits.

Step 3. If $\mathrm{Q}_x = D$ for particular X, then X is a position of error. The data at position X are then corrected by an inversion of the data bit.

(b) Recent Server Machines A typical server machine of IBM eServer z900 and z990, whose dependable system structure is based on the former server IBM S/390 G5 and G6, has the strategy to enable *continuous reliable operation*, supported by the following building blocks [MUEL99, ALVE02, FAIR04]:

- Error prevention
- Error detection
- Error recovery
- Problem determination
- Service / support
- Change management
- Measurement

These blocks provide the capabilities of *self-protecting*, *self-healing*, *self-configuring*, *and self-optimizing*. This design strategy is illustrated in Figure 12.50. Major enhancements in RAS design, *concurrent upgrade* and *concurrent repair* for the system have been made in the processor, memory, I/O, power / cooling, service / support subsystem, and so forth.

The dependable techniques of duplication, $N + 1$ redundancy and coding are extensively applied to the subsystems of processor, storage, I/O, and so forth. In the processor subsystem the processor units (PUs) each having the level-1 cache (L1 cache) and the secondary cache (L2 cache) are duplicated and are tightly coupled. Parity-prediction and carry checking are applied to the adders and the arithmetic logic units (ALUs). Also residue checking is applied to the modular exponentiation engines in the cryptographic coprocessor element.

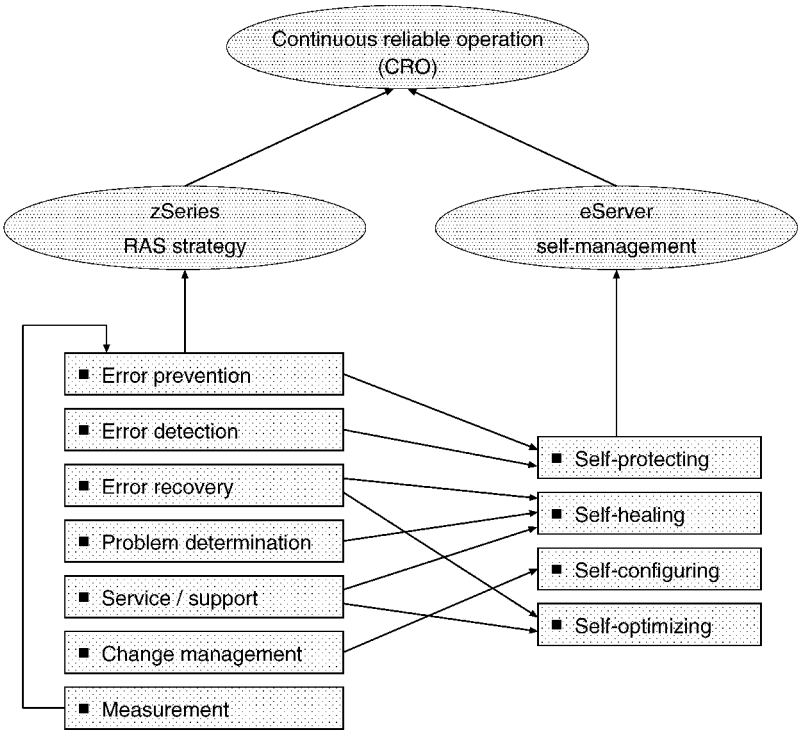


Figure 12.50 *zSeries RAS strategy building blocks and eServer self-management.* Source: *[ALVE02]. © 2002 by International Business Machines Corporation; republished by permission.*

In memories some type of bit or byte error correcting / detecting codes are applied to the following various storages:

- Level-1 cache (L1 cache): simple parity-check code
- Level-2 cache (L2 cache): (25, 19) ECC for address field in the directory, (11, 5) ECC for ownership field in the directory, and (72, 64) SEC-DED code for memory data field
- Main memory: minimum-weight & equal-weight-row (140, 128) S2EC-D2ED code [CHEN96] for data field* and background scrubbing

For address information, two memory address parity bits are added to prevent data from being fetched from an erroneous location. However, these address parity bits are not stored in the main memory. In another memories and the bus-line circuit, the following codes are applied:

- Address translation buffer: (16, 10) SEC-DED code for each of duplicated stored data, and background scrubbing

*The (76, 64) S4EC-DED code is applied to the former Server G3 and G4 to ensure that all single 4-bit errors in a DRAM chip with 4-bit I/O data are corrected and also random double-bit errors are detected [DOET97, SPAI99]. This type of code is presented in Section 6.2.

- Bus interface between L2 cache and memory controller: simple parity-check code for command and status bus
- Cryptographic key storage: triplicated key data, each appended by simple parity-check bit

Background Memory-Scrubbing and Sparing In main memories the dynamic form of *sparing* is performed via *background scrubbing*, a process of error avoidance. The memory-scrubbing process serves two functions. The first function is to eliminate the accumulation of soft errors in the memory chip. The purpose is to reduce the likelihood of the alignments of existing soft errors and future hard or soft errors. The second function of scrubbing is to identify and record hard errors in the memory chip. Multiple hard errors are prime targets to line up in the same ECC word because they can result in uncorrectable error events. Error counts are accumulated while scrubbing, and DRAMs with high counts are spared. Once a memory chip with multiple hard errors is identified, a spare chip replacement is invoked to transfer data from the failing chip to the spare chip. The failing chip then is set to become inactive. The scrubbing process runs in the background, with minimal interference with the normal system operations. With up to 32 spare DRAMs per memory card, a memory card will rarely need to be replaced because of DRAM failure.

12.4.2 Coding for VLSI Processors / Microprocessors

In today's microprocessors a variety of coding techniques are being applied not only to the ALU logics and the cache memories but also to the data transfer circuits such as bus-line circuits, the register arrays, the key storage, and the address translation arrays.

1. Duplicate VLSI Processors

Sedmak [SEDM80b] studied the fault tolerance of a general purpose computer utilizing very large scale integration chips. A major method of achieving fault tolerance is by internal redundancy using *duplicate complementary logic*. This kind of complementary duplication is used in the VLSI chips shown in Figure 12.51. In these figures it is easy to see that for a given combination logic element in the functional portion, the signals into and out of the gate are polarities, opposite to those of the complementary portion. This technique serves two purposes. First, it eliminates a problem associated with applying the same mask or cell type twice internally. The problem is that failures (designs, process, and wearout) undetected by the comparators could occur in noncomplementary duplication where a mask or cell fault might materialize in both the functional and duplicate circuits and thus create an identical failure state. Second, the design with complementary duplication will be much less susceptible to bridging faults than noncomplementary duplication. This is because there will be far fewer occurrences of long nets of metalization with the same Boolean function and the same polarity signal, which, if bridged, might result in an undetected error should a subsequent failure occur.

Each of the code checkers and comparators is implemented using a self-checking design approach. The self-checking comparators shown in Figures 12.24 and 12.25 are extensively used. As a result of the cost-effectiveness of the design, the logic overhead that consists of duplicate complementary gates, comparators, and other fault detection circuits, comprises approximately 55% of the total gates in the CPU. Compared to the conventionally checked VLSI machine, the increase in chip count is only 5.5%. In this

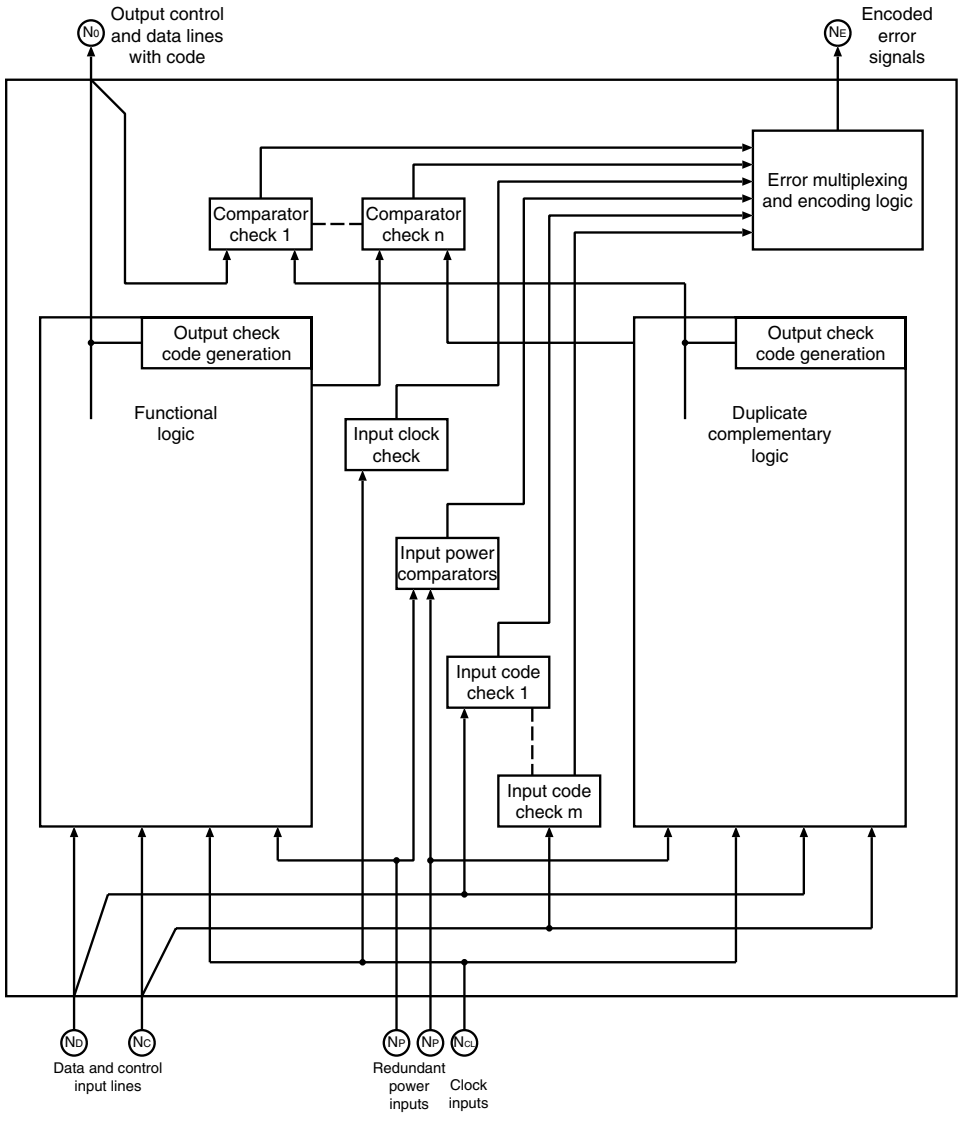


Figure 12.51 *Generalized VLSI chip.* Source: *[SEDM80b]. © 1980 IEEE.*

case a conventionally checked VLSI machine is assumed to have simple parity-check code for memories and complete checking for single faults in data paths, both with no fault detection in the control logic.

2. Self-checking Microprocessors

Microprocessors have been extensively applied to recent digital systems. They are also sometimes used in ultra-reliable process controls for industry and medicine. A considerable body of work on a self-checking version of the commercial microprocessors or microprogram control unit has been done, as can be found in [DISP81, TSAO82, WONG83, HALB84, HALB84, NICO85a, YEN87, NANY88].

The techniques adopted commonly are classified as follows:

1. Duplication of circuits (e.g., a control circuit, or ALU) and an output comparison check using a self-checking comparator (or self-checking two-rail code checker).
2. Coding for control signals (encoded in two-rail codes, or Berger codes) and for address and data signals (encoded in simple parity-check codes, or m-out-of-n codes).
3. Extensive use of programmable gate arrays together with the self-checking techniques.

A diagram of a VLSI self-checking microprocessor is given in Figure 12.52 [DISP81]. The hardware overhead of this self-checking microprocessor is 30% to 60% greater as compared to the microprocessor without self-checking capability.

The SFS microprocessor's target architecture is Intel's i8080 8-bit microprocessor [NANY88]. This design is related to the error-secure / error-propagating (ES / EP) concepts of Theorem 12.3 and Definitions 12.10 and 12.11. The new SFS processor design requires only a few TSC checkers (i.e., four checkers at embedded interfaces) and no duplicated subsystems except a few registers. Therefore a hardware overhead of only 38% additional gates is required for the SFS version, as compared with the non-SFS version of the i8080.

The SFS design approach includes the following features.

1. A complete set of functional building blocks that has been defined and investigated for ES and EP properties in all interfaces of each block. Each section of the processor consists of only these building blocks. This feature facilitates the verification of the SFS property in each section and drastically reduces the number of TSC checkers required for the SFS processor as a whole.
2. Every interface of the blocks is encoded in an *unordered code* [LALA01] that detects unidirectional errors. For the register section and the microprogram section, the *Berger code* [BERG61], which will be presented in Definition 12.16, is considered to be the most cost-effective to use, while the two-rail code is applied to the ALU section, the timing controller, and the external control signals.

The modeled faults in the processor are single stuck-at faults in gate outputs; single stuck-at line faults, the single crosspoint faults, and the bridging faults between adjacent lines in programable gate arrays; single-bit slice faults in registers; and unidirectional faults in buses.

Some new design techniques are applied to every building block in order to satisfy the ES and EP properties at each interface. A unique design is utilized for a sequential circuit with its next-state function d and output function w such that, for unidirectional faults in d and w, the circuit is SFS if the outputs of w are encoded in an *unordered code* [LALA01], and d and w are implemented with inverter-free circuits. This brings a significant advantage to the practical design of complex SFS systems [NANY87]. For further details, refer to [NANY88].

The overall organization of the SFS processor is shown in Figure 12.53. It consists basically of an ALU section, a control section, a register section, and an internal data bus. Four TSC checkers of TSC 2-rail code checker and TSC Berger code checkers are used as indicated in the figure.

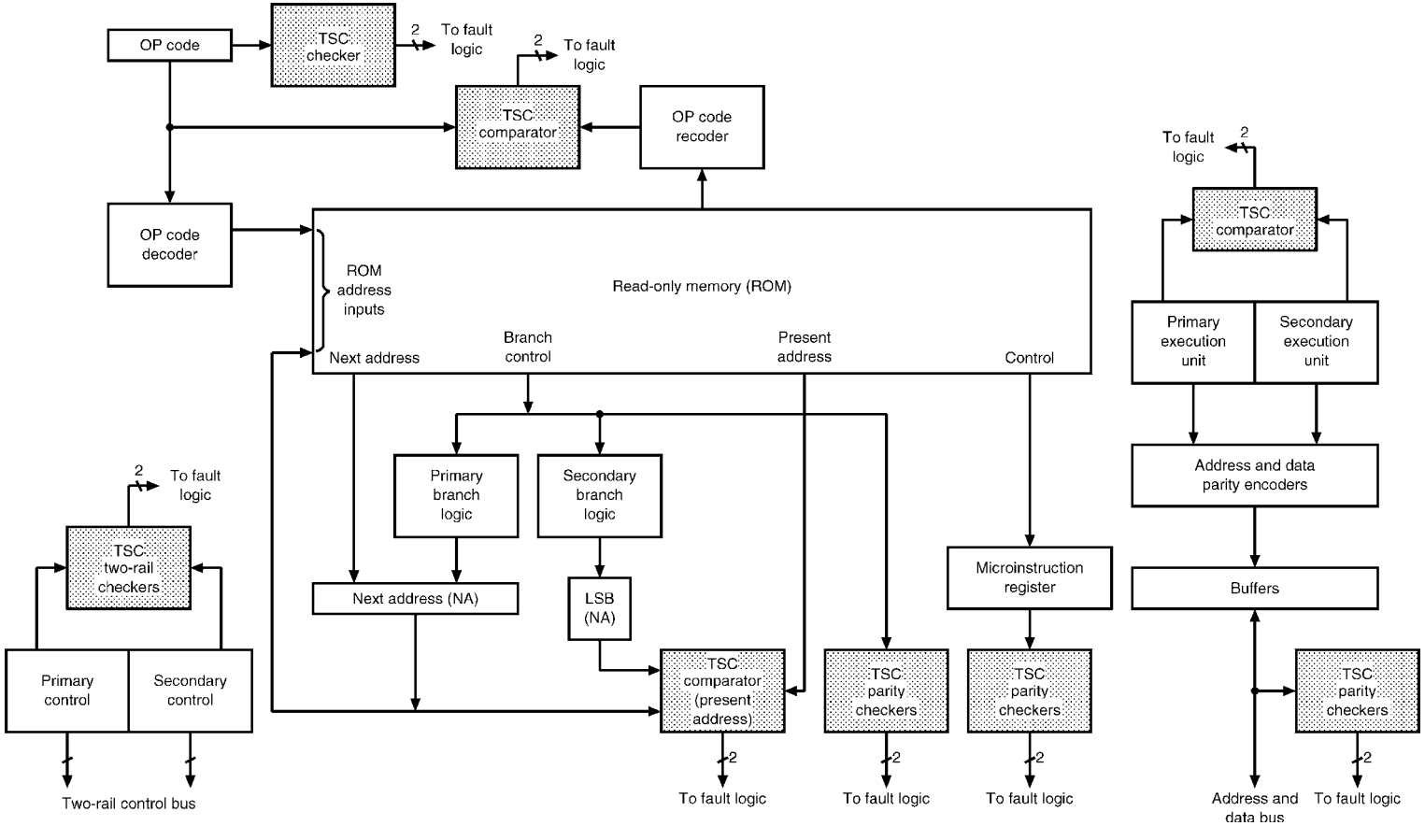


Figure 12.52 *Example of self-checking microprocessor.* Source: *[DISP81]. © 1981 IEEE.*

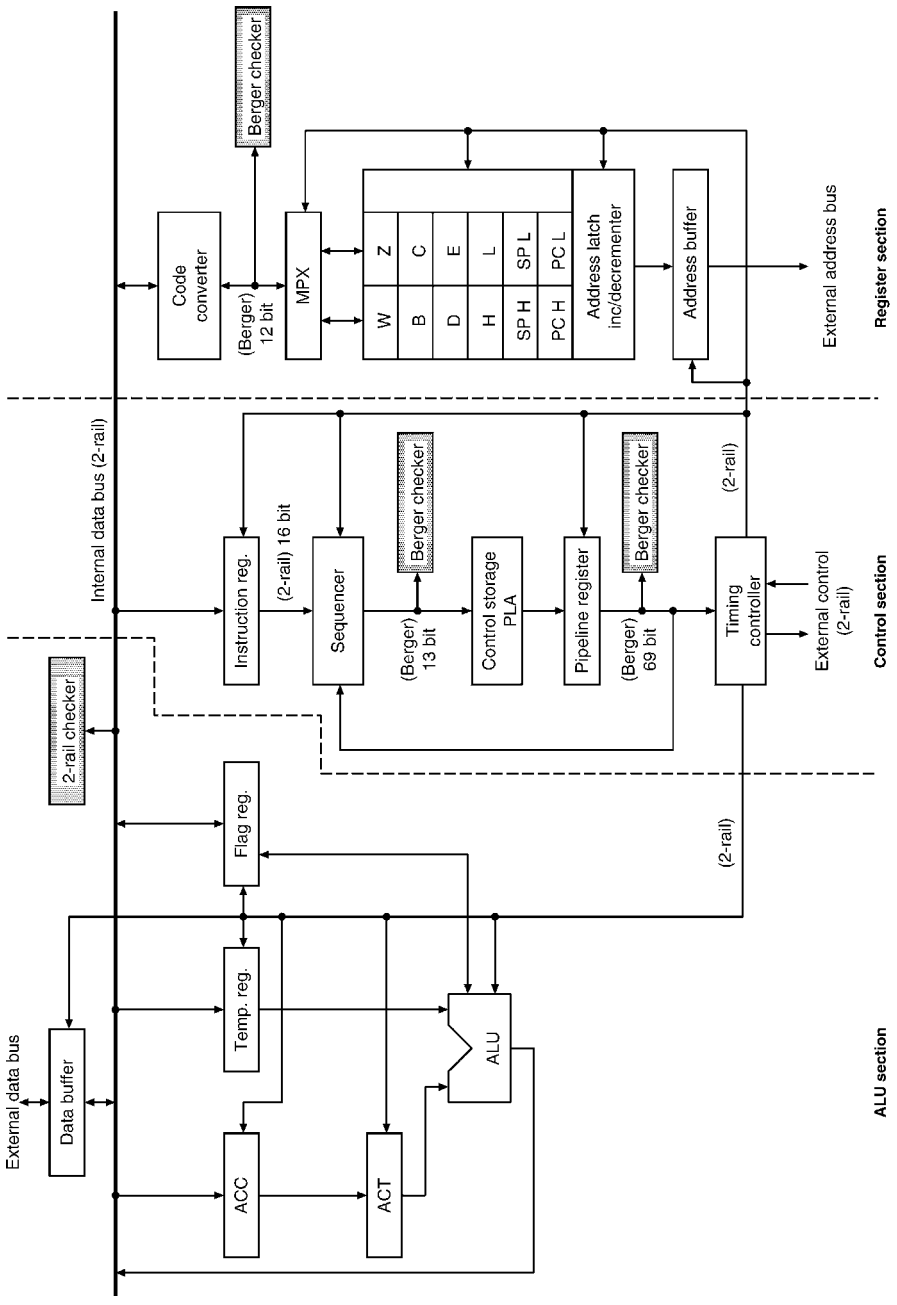


Figure 12.53 *SFS processor organization.* Source: *[NANY88]. © 1988 IEEE.*

Unordered Codes A word $A = (a_0, a_1, \ldots, a_{k-1})$ *covers* another word $B = (b_0, b_1, \ldots, b_{k-1})$, written as $A \geq B$, if $b_i = 1$ implies $a_i = 1$ for $i = 0, 1, \ldots, k-1$. In other words, the positions of 1's in B are subset of the positions of 1's in A. For example, if $A = (1\ 0\ 1\ 0\ 1\ 0)$ and $B = (0\ 0\ 1\ 0\ 1\ 0)$, then $A > B$. It is noted that if A covers B, then $N(B, A)$, which means the number of $1 \longrightarrow 0$ crossovers from B to A, equals 0. If A does not cover B, and B does not cover A, then A and B are *unordered*. A code in which no codeword is covered by any other codeword is said to be an *unordered code*. An unordered code is capable of detecting all unidirectional errors. The *m-out-of-n code*, whose codewords have exactly m 1's and $n - m$ 0's, and the *Berger code* are typical unordered codes.

Berger Codes for All Unidirectional Error Detection (AUED) The Berger code [BERG61] is known as an optimal *systematic AUED code*. For the codeword $(a_0, a_1, \ldots, a_{k-1}, a_k, \ldots, a_{n-1})$, $(a_0, a_1, \ldots, a_{k-1}) = A$ is an information part and $(a_k, \ldots, a_{n-1}) = f(A)$ is a check part induced from A, where the function $f(A)$ is the binary representation of number of 0's in A. Hence the number of check bits r required is given by $r = n - k = \lfloor \log_2 k + 1 \rfloor$, where $\lfloor y \rfloor$ means the largest integer smaller than or equal to y. For example, for a 6-bit $(k = 6)$ information $A = (1\ 1\ 0\ 1\ 0\ 0)$, then $r = \lfloor \log_2 6 + 1 \rfloor = 3$ and the number of 0's is 3, and hence $f(A) = 011$. Thus $(1\ 1\ 0\ 1\ 0\ 0 \vdots 0\ 1\ 1)$ is a codeword. Consider a unidirectional error, for example, with 1-errors. In this situation the number of 0's in the information part may only increase, and not the check part because of 1-errors. Therefore any number of 1-errors cannot make a codeword into another codeword. This also holds for 0-errors.

Definition 12.16 The *Berger code* is a *systematic all unidirectional error detecting code* whose codeword consists of a k-bit information part A and a $\lfloor \log_2 k + 1 \rfloor$ bits check part $f(A)$, where $f(A)$ gives the binary representation of the number of 0's in A. □

3. On-Chip ECCs in Recent Microprocessors

A substantial error detection and correction mechanism is now installed in recent microprocessors [BISH96, TEND02, RUSU03, STIN03, CHAN05, MCIN05, TAKA05, NAKA05, SHIN05]. For example, in the 64-bit RISC microprocessor the on-chip data cache and instruction cache, as well as the multiport register array, employ ECCs such as the SEC code, the SEC-DED code, or a simple parity-check code that allow most single-bit errors to be corrected or detected. The floating-point unit in the chip is checked by a residue-check code, and the majority of the data-flow circuits also maintain parity check, which adds error-detection capabilities.

The microprocessor chip includes an interface control circuit block which controls the interface between the processor chip and the outside cache memories or the main memories. The bit / byte error control codes presented in Chapters 4 through 6 are employed for these memories, and therefore encoder and decoder circuits are included in the control circuit block. Hence the errors exist in the outside cache or the main memory, or the errors in the interface circuits, such as bus-line circuits connected with these memories, are corrected or detected at the processor chip.

Also in recent microprocessors, the SEC-DED codes and the SEC-DED-S*b*ED codes are extensively applied to the inside bus-line circuits and to the inside cache memories. Intermittent errors caused by signal coupling between adjacent bus lines can be corrected or detected.

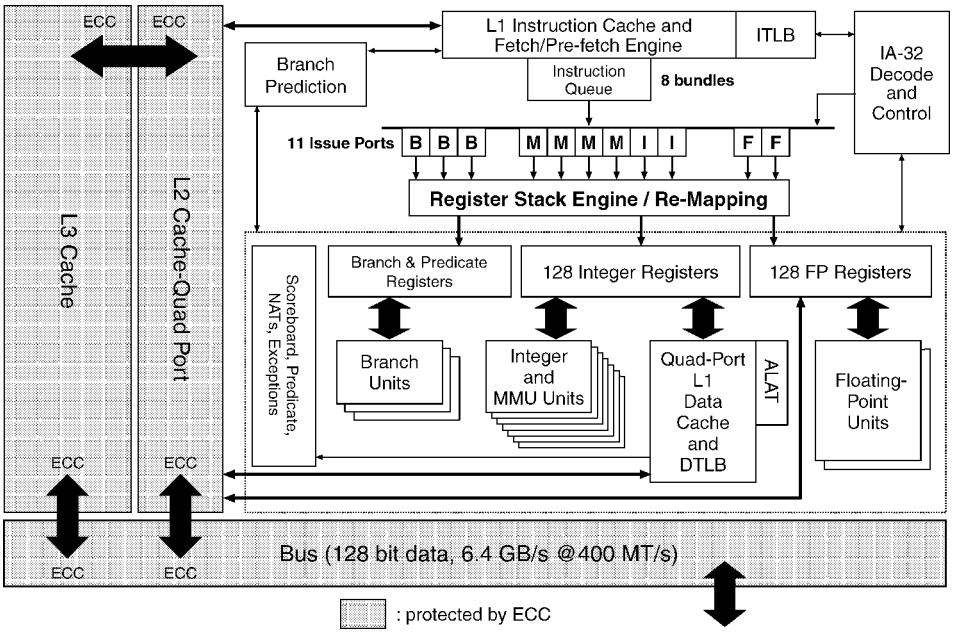


Figure 12.54 *Circuit block diagram of a microprocessor.* Source: *[RUSU03]. © 2003 IEEE.*

Figures 12.54 and 12.55 show the block diagram of a recent microprocessor and its RAS (reliability, availability, and serviceability) features, where ECCs such as bit error control codes or bit / byte error control codes are applied substantially to the inside cache memories and bus-line circuits. In this example the microprocessor chip, the

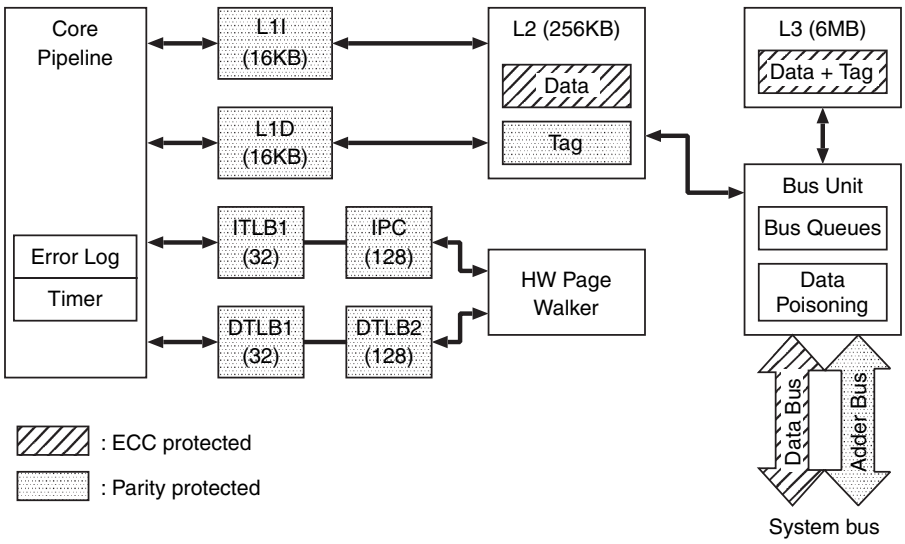


Figure 12.55 *ECC / parity-protected area in the microprocessor chip.* Source: *[RUSU03]. © 2003 IEEE.*

level-1 cache (both instruction cache L1I and data cache L1D), the translation lookaside buffers (TLBs), the level-2 cache tag (L2 tag), and the 44-bit system address bus are protected by a simple parity-check code. The level-2 cache (L2) data array, the level-3 cache (L3 data and tag), and the 128-bit system data bus are protected by ECCs such as the bit / byte error control codes. The recent level-3 cache with 9 MB capacity of this microprocessor employs ECCs with 10 check bits for 256 information data bits [CHAN05].

EXERCISES

12.1 Show that the circuit in Figure 12.6 is self-testing for all faults affecting fewer than $n + 1$ bits output if the input has an even-parity codeword.

12.2 Find the tested and secure fault sets of the parallel exclusive-OR circuit in Figure 12.6 when the inputs are encoded in a distance-3 Hamming code.

12.3 Let the input codespace of the circuit shown below be $N \in \{(a_0,\ b_0)\ (a_1,\ b_1) \mid (a_0,\ b_0),\ (a_1,\ b_1) = \text{1-out-of-2 codes}\}$. Let the output codespace be $S \in \{(c_0,\ c_1) \mid (c_0,\ c_1) = \text{1-out-of-2 code}\}$. Show that the circuit is a TSC checker for all single faults, provided that it receives all four possible codeword inputs.

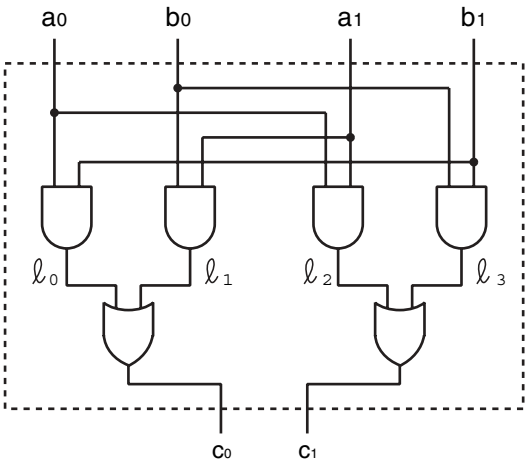


Figure P12.3 *Two-input two-rail code checker.*

12.4 Verify that the cascaded inverter-free network shown in Figure 12.13, whose primary inputs are encoded in 1-out-of-2 codes, is SFS with respect to unidirectional faults.

12.5 Consider the bit-sliced system in Figure P12.5 where each slice consists of three registers and a multiplexer that loads unordered codewords from one of I_1 and I_2, depending on a two-rail encoded load signal $(C,\ \overline{C})$ into D, as shown in the

figure [NANY88]. Show that the system is SFS for unidirectional faults, even though it does not satisfy the code-disjoint (CD) property.

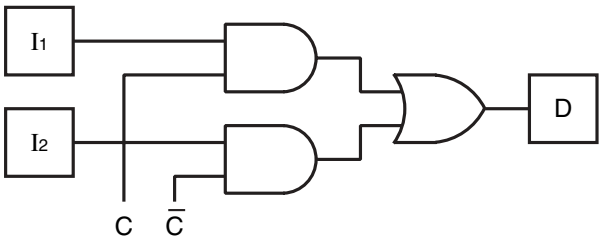


Figure P12.5 *Bit-sliced circuit.*

12.6 Show that the fault-free system shown in Exercise 12.5 is error secure (ES).

12.7 Answer the following questions for the circuit in Figure 12.19.

(a) Show that this checking circuit **CK** can detect errors in y_0 or $y_1 \cdot y_2$, but not both.

(b) Errors due to y_1 or y_2 can be detected with certain input conditions. Find these conditions.

12.8 Design the parity prediction checker for the combinational circuit with 3-input $(x_0\ x_1\ x_2)$, and 3-output $(y_0\ y_1\ y_2)$ defined by the following truth table.

x_0	x_1	x_2	y_0	y_1	y_2
0	0	0	1	1	1
0	0	1	0	0	1
0	1	0	1	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	1	0	1

Next design the self-testing parity prediction checker by slightly modifying the checker above. In this case verify that every stuck-at-1 fault and stuck-at-0 fault at every gate output can be detected by at least one input $(x_0\ x_1\ x_2)$.

12.9 Design the input regeneration checker for the 1-out-of-8 decoder.

12.10 Verify that the 1-out-of-8 decoder shown in (a) of Figure P12.10 is self-testing for all single faults, while the decoder shown in (b) is not self-testing [CART71b, WAKE78]. Here the output codespace is defined as $\{(d_0,\ d_1,\ \ldots,\ d_7)\ (z_0,\ z_1) \mid (d_0,\ d_1,\ \ldots,\ d_7) =$ 1-out-of-8 *codeword*, and $(z_0,\ z_1) =$1-out-of -2 *codeword*$\}$.

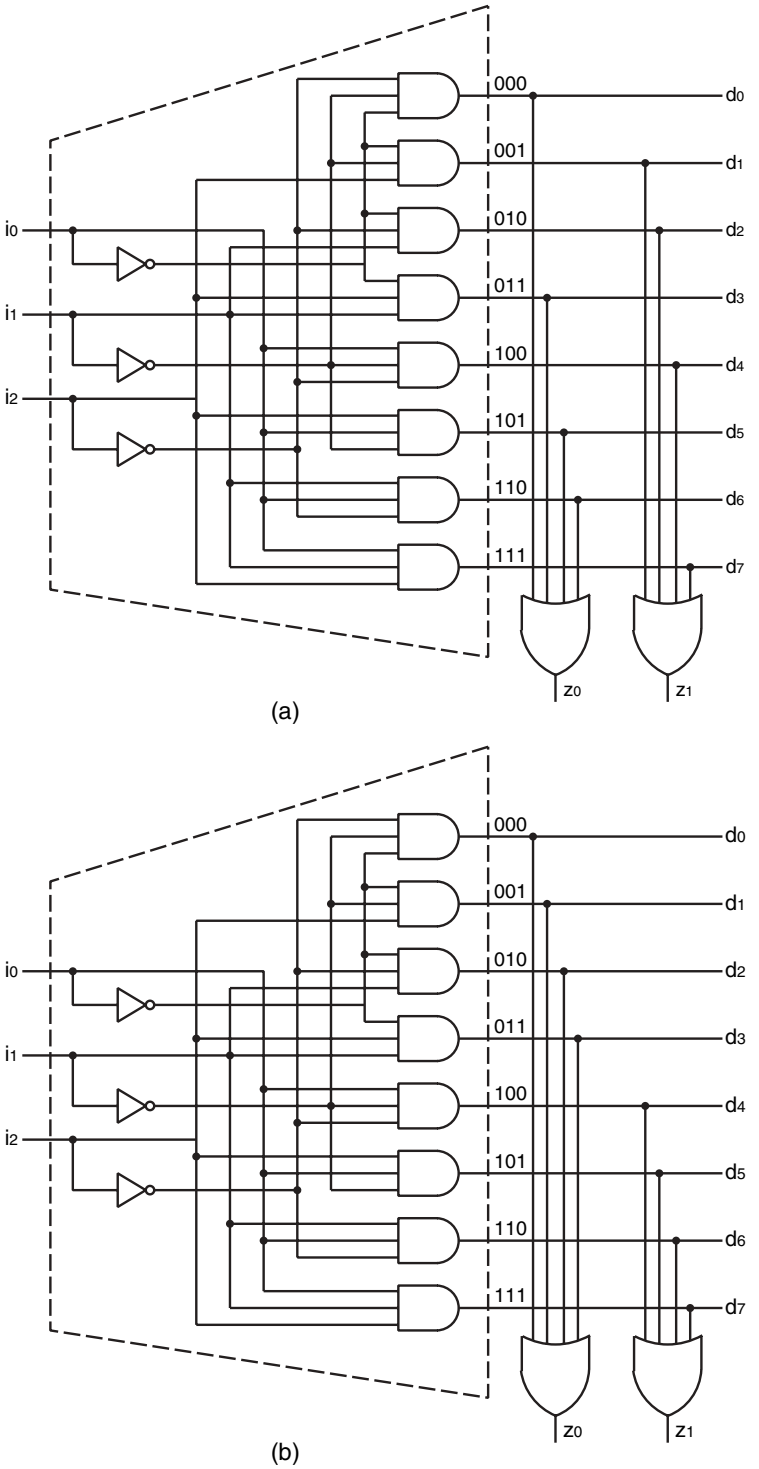


Figure P12.10 *1-Out-of-8 decoder checks.*

12.11 Find the test patterns sufficient to test the self-testing (odd) parity checker shown in Figure 12.21.

12.12 Prove Theorem 12.4.

12.13 Prove Theorem 12.5.

12.14 Design the complementary duplication checker for the 4-input, 3-output circuit shown in Example 12.5. Also show the circuit designed in a complementary form of this circuit.

12.15 Verify that the two-rail code checker in Figure 12.25 is self-testing for all unidirectional multiple faults.

12.16 Find the minimum number of test patterns for the following two-rail code checker.

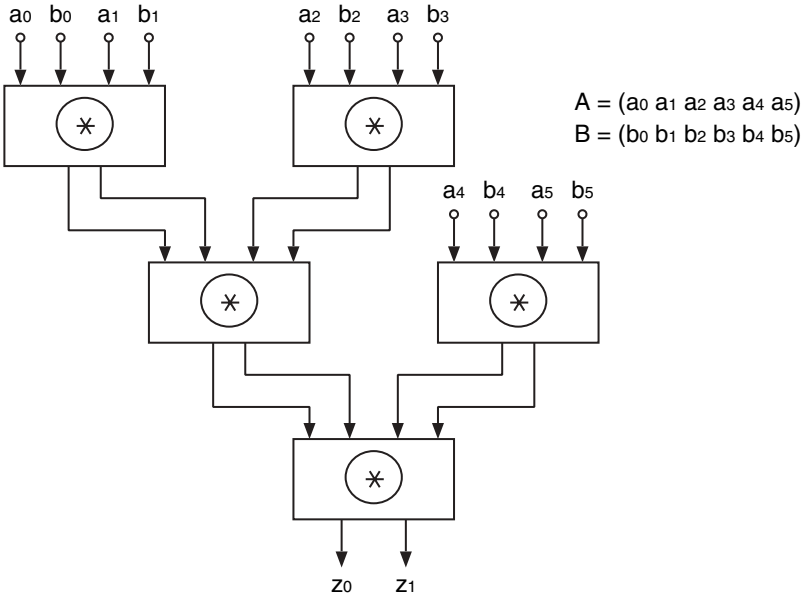


Figure P12.16 *Six-input two-rail code checker.*

12.17 Design the self-testing checker for a Berger code with length $k = 4$, and $n = 7$.

12.18 Design the GPCs with the following prediction functions and **H** matrices for the four-input, and three-output combinational circuit shown in Example 12.5.

(a) Prediction function: $\mathbf{K} \cdot U =$ simple parity-check function of inputs,

$$\mathbf{K} = \begin{bmatrix} 0110100010000000 \\ 0110100010000000 \\ 0110100010000000 \end{bmatrix}.$$

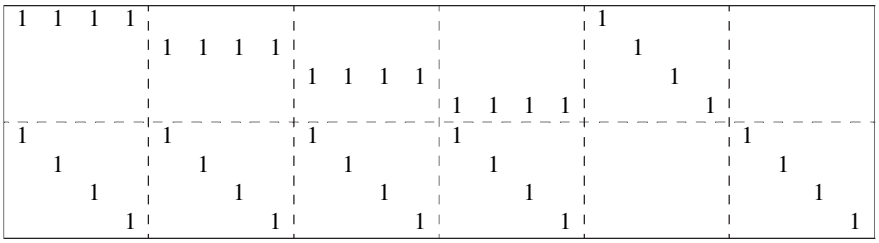
$$\mathbf{H} = [1\ 1\ 1]$$

(b) Prediction function: $\mathbf{K} \cdot U =$ OR function of inputs,

$$\mathbf{K} = \begin{bmatrix} 0111111111111111 \\ 0111111111111111 \\ 0111111111111111 \end{bmatrix},$$
$$\mathbf{H} = [1\ 1\ 1].$$

12.19 Using the following two-dimensional cross-parity code, design the 4-modularized error correction circuit for the 16 bits adder:

$\mathbf{H} =$



12.20 Suppose that two binary operands, $(a_0\ a_1\ a_2\ a_3)$ and $(b_0\ b_1\ b_2\ b_3)$, are encoded by the following $\mathbf{H}$ matrix:

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

Design the error correction circuit for the 4-bit binary adder, and demonstrate that a single error at an output of the adder can be corrected.

12.21 Show that an adder error can be masked by an alternate data retry (ADR), where ADR is defined as an extra operation retried by using the bitwise complement input data, as shown in Subsection 1.3.2.

12.22 Find the necessary and sufficient conditions of the self-complementing checksum code. A self-complementing code is defined such that the bitwise complement of the codeword is also a codeword, as shown in Definition 3.6.

12.23 Prove Theorem 12.12.

12.24 Obtain the parity prediction in the AND or OR operation under the condition that input operands are encoded in a simple parity-check code. Show the difference between the straightforward duplication-comparison scheme and this method.

12.25 By extending the results of Theorem 12.14, find the generalized parity prediction scheme for combinational logic.

12.26 Verify that the code presented in Eq. (12.28) corrects any single-symbol errors and any double-bit errors. (Note that double-bit errors and single-symbol errors

cannot be corrected simultaneously.) Also verify that this corrects any single-bit errors in the presence of a symbol erasure.

12.27 Prove that an unordered code can detect all unidirectional errors.

REFERENCES

[ABRA86] J. A. Abraham and W. K. Fuchs, ''Fault and Error Models for VLSI,'' *Proc. IEEE*, 74 (May 1986): 639–654.

[ALVE02] L. C. Alves, M. L. Fair, P. J. Meaney, C. L. Chen, W. J. Clarke, G. C. Wellwood, N. E. Weber, I. N. Modi, B. K. Tolan, and F. Freier, ''RAS Design for the IBM eServer z900,'' *IBM J. Res. Dev.*, 46 (July–September 2002): 503–521.

[ANDE71] D. A. Anderson, ''Design of Self-Checking Digital Networks Using Coding Techniques,'' Report of Coordinated Science Labs, University of Illinois, R-527 (October 1971).

[ANDE73] D. A. Anderson and G. Metze, ''Design of Totally Self-Checking Check Circuits for *m*-out-of-*n* Codes,'' *IEEE Trans. Comput.*, C-22 (March 1973): 263–269.

[ANDE81] T. Anderson and P. A. Lee, *Fault Tolerance, Principle and Practice*, Prentice Hall (1981).

[ASHJ77] M. J. Ashjaee and S. M. Reddy, ''On Totally Self-Checking Checkers for Separable Codes,'' *IEEE Trans. Comput.*, C-26 (August 1977): 737–744.

[AVIZ71] A. Avizienis, G. C. Gilley, F. P. Mathur, D. A. Rennels, J. A. Rohr, and D. K. Rubin, ''The STAR (Self-Testing and Repairing) Computer: An Investigation of the Theory and Practice of Fault-Tolerant Computer Design,'' *IEEE Trans. Comput.*, C-20 (November 1971): 1312–1321.

[BERG61] J. M. Berger, ''A Note on Error Detecting Codes for Asymmetric Channels,'' *Info. Contr.*, 4 (March 1961): 68–73.

[BISH96] J. W. Bishop, M. J. Campion, T. L. Jeremiah, S. J. Mercier, E. J. Mohring, K. P. Pfarr, B. G. Rudolph, G. S. Still, and T. S. White, ''Power PC AS A10 64-bit RISC Microprocessor,'' *IBM J. Res. Dev.*, 40 (July 1996): 495–505.

[BOSE84] B. Bose and D. J. Lin, ''PLA Implementation of *k*-out-of-*n* Code TSC Checker,'' *IEEE Trans. Comput.*, C-33 (June 1984): 583–588.

[BOSS70] D. C. Bossen, D. L. Ostapko, and A. M. Patel, ''Optimum Test Patterns for Parity Networks,'' *Proc. Fall Joint Computer Conf.*, AFIPS (1970): 63–68.

[BOSS82] D. C. Bossen and M. Y. Hsiao, ''Model for Transient and Permanent Error-Detection and Fault-Isolation Coverage,'' *IBM J. Res. Dev.*, 26 (January 1982): 67–77.

[BROW60] D. T. Brown, ''Error Detecting and Correcting Binary Codes for Arithmetic Operations,'' *IEEE Trans. Electron. Comput.*, EC-9 (September 1960): 333–337.

[CARL86] R. O. Carlson and C. A. Neugebauer, ''Future Trends in Wafer Scale Integration,'' *Proc. IEEE*, 74 (December 1986): 1741–1752.

[CART68] W. C. Carter and P. R. Schneider, ''Design of Dynamically Checked Computers,'' *Proc. IFIPS 68, Edinburgh, Scotland*, 2 (August 1968): 878–883.

[CART70] W. C. Carter, D. C. Jessep, and A. Wadia, ''Error-Free Decoding for Failure-Tolerant Memories,'' *Proc. IEEE Int. Computer Group Conf.* (June 1970): 229–239.

[CART71a] W. C. Carter, D. C. Jessep, A. B. Wadia, P. R. Schneider, and W. G. Bouricius, ''Logic Design for Dynamic and Interactive Recovery,'' *IEEE Trans. Comput.*, C-20 (November 1971): 1300–1305.

[CART71b] W. C. Carter, K. A. Duke, and D. C. Jessep, ''A Simple Self-Testing Decoder Checking Circuit,'' *IEEE Trans. Comput.*, C-20 (November 1971): 1413–1414.

[CART77] W. C. Carter, G. R. Putzolu, A. B. Wadia, W. G. Bouricius, D. C. Jessep, E. P. Hsieh, and C. J. Tan, "Cost Effectiveness of Self-Checking Computer Design," *Dig. 7th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1977): 117–123.

[CART86] W. C. Carter, "Improved Parallel Signature Checker/Analyzers," *Dig., 16th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1986): 416–421.

[CHAN05] J. Chang, S. Rusu, J. Shoemaker, S. Tam, M. Huang, M. Haque, S. Chiu, K. Truong, M. Karim, G. Leong, K. Desai, R. Goe, and S. Kulkarni, "A 130-nm Triple-V_t 9MB Third-Level On-Die Cache for the 1.7-GHz Itanium 2 Processor," *IEEE J. Solid-State Circ.*, 40 (January 2005): 195–203.

[CHEN92] C. L. Chen, N. N. Tendolkar, A. J. Sutton, M. Y. Hsiao, and D. C. Bossen, "Fault-Tolerance Design of the IBM Enterprise System / 9000 Type 9021 Processors," *IBM J. Res. Dev.*, 36 (July 1992): 765–779.

[CHEN96] C. L. Chen, "Symbol Error Correcting Codes for Memory Applications," *Proc. 26th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1996): 200–207.

[CHUA78] H. Y. H. Chuang and S. Das, "Design of Fail-Safe Sequential Machines Using Separable Codes," *IEEE Trans. Comput.*, C-27 (March 1978): 249–251.

[DAVI78a] R. David, "A Totally Self-Checking 1-Out-of-3 Checker," *IEEE Trans. Comput.*, C-27 (June 1978): 570–572.

[DAVI78b] R. David and P. Thevenod-Fosse, "Design of Totally Self-Checking Asynchronous Modular Circuits," *J. Design Autom. Fault Tolerant Comput., 2* (October 1978): 271–278.

[DIAZ74a] M. Diaz, J. C. Geffroy, and M. Courooisier, "On-Set Realization of Fail-Safe Sequential Machines," *IEEE Trans. Comput.*, C-23 (February 1974): 133–138.

[DIAZ74b] M. Diaz, "Design of Totally Self-Checking and Fail-Safe Sequential Machines," *Dig., 4th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1974): 3-19–3-24.

[DIAZ79] M. Diaz, P. Azema, and J. M. Ayache, "Unified Design of Self-Checking and Fail Safe Combinational Circuits and Sequential Machines," *IEEE Trans. Comput.*, C-28 (March 1979): 276–281.

[DISP81] C. P. Disparte, "A Self-Checking VLSI Microprocessor for Electronic Engine Control," *Dig. 11th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1981): 253.

[DOET97] G. Doetting, K. J. Getzlaff, B. Leppla, W. Lipponer, T. Pflueger, T. Schlipf, D. Shmunkamp, and U. Wille, "S/390 Parallel Enterprise Server Generation 3: A Balanced System and Cache Structure," *IBM J. Res. Dev.*, 41 (July–September 1997): 405–428.

[EFST83] C. Efstathiou and C. Halatsis, "Modular Realization of Totally Self-Checking Checker for *m*-out-of-*n* Codes," *Dig. 13th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1983): 154–161.

[ENGL66] W. A. England, "Improving Reliability by the Application of Selected Redundant Techniques," *Proc. Workshop on Reliability Technique, UCLA (April 1966).*

[FAIR04] M. L. Fair, C. R. Conclin, S. B. Swaney, W. J. Clanke, et al., "Reliability, Availability, and Serviceability (RAS) of the IBM eServer z990," *IBM J. Res. Dev.*, 48 (May–July 2004): 519–534.

[FRAN66] H. Frank and S. S. Yau, "Improving Reliability of a Sequential Machine by Error-Correcting State Assignments," *IEEE Trans. Comput.*, C-15 (1966): 111–113.

[FUJI79] E. Fujiwara and K. Haruta, "Design of Totally Self-Checking Checker for Main Storage Error Checking and Detecting Circuit" (in Japanese), *Trans. IECE Japan*, 62-D (June 1979): 419–426.

[FUJI81] E. Fujiwara and K. Haruta, "Fault-Tolerant Arithmetic Logic Unit Using Parity-Based Codes," *Trans. IECE Japan*, E64 (October 1981): 653–660.

[FUJI84] E. Fujiwara, N. Mutoh, and K. Matsuoka, "A Self-Testing Group-Parity Prediction Checker and Its Use for Built-In Testing," *IEEE Trans. Comput.*, C-33 (June 1984): 578–583.

[FUJI87a] E. Fujiwara and K. Matsuoka, "A Self-Checking Generalized Prediction Checker and Its Use for Built-In Testing," *IEEE Trans. Computers*, C-36 (January 1987): 86–93. (Also in *Dig., 15th IEEE Int. Symp. on Fault-Tolerant Computing* [June 1985]: 384–389.)

[FUJI87b] E. Fujiwara and K. Matsuoka, "Fault-Tolerant k-out-of-n Logic Unit Networks" (in Japanese), *Trans. IECE Japan*, J70-D (September 1987): 1791–1800.

[FUJI87c] E. Fujiwara and K. Matsuoka, "A Hardware Implementation of Permuter" (in Japanese), *Trans. IECE Japan*, J70-D (October 1987): 1995–1998.

[FURU83a] K. Furuya, K. Takeda, and Y. Tohma, "Logic Design of Fault-Tolerant Arithmetic Units with Check-Sum Code Based on Alternate-Data-Retry Strategy" (in Japanese), *Trans. IECE Japan*, J66-D (March 1983): 243–250.

[FURU83b] K. Furuya, K. Takeda, and Y. Tohma, "Extension of Alternate-Data-Retry Strategy to Adders with Biresidue Code and Estimates of Several Configurations" (in Japanese), *Trans. IECE Japan*, J66-D (March 1983): 251–258.

[FURU83c] K. Furuya, Y. Akita, and Y. Tohma, "Logic Design of Fault-Tolerant Dividers Based on Data Complementation Strategy," *Dig., 13th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1983): 306–313.

[GADD70] T. G. Gaddess, "An Error-Detecting Binary Adder: A Hardware-Shared Implementation," *IEEE Trans. Comput.*, C-19 (January 1970): 34–38.

[GAIT83] N. Gaitanis and C. Halatsis, "A New Design Method for m-out-of-n TSC Checkers," *IEEE Trans. Comput.*, C-32 (March 1983): 273–283.

[GAIT84] N. Gaitanis, "Totally Self-Checking Checkers for Low Cost Arithmetic Codes," *Dig., 14th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1984): 260–264.

[GARN58] H. L. Garner, "Generalized Parity Checking," *IRE Trans. Electron. Comput.*, EC-7 (September 1958): 207–213.

[GARN59] H. L. Garner, "The Residue Number System," *IRE Trans. Electron. Comput.*, EC-8 (September 1959): 140–147.

[GARN66] H. L. Garner, "Error Codes for Arithmetic Operations," *IEEE Trans. Electron. Comput.*, EC-15 (October 1966): 763–770.

[GEEL85] M. B. Geelhoed and M. J. Jordan, "SOPHO S2500, The Heigh Range Communication Swith," *Philips Telecommun. Rev., 43* (June 1985): 92–113.

[GILS86a] W. J. van Gils, "An Error-Control Coding System for Storage of 16-Bit Words in Memory Arrays Composed of Three 9-Bit Wide Units," *Philips J. Res.*, 41 (1986): 391–399.

[GILS86b] W. J. van Gils, "A Triple Modular Redundancy Technique Providing Multiple-Bit Error Protection without Using Extra Redundancy," *IEEE Trans. Comput.*, C-35 (July 1986): 623–631.

[GILS87] W. J. van Gils and J.-P. Boly, "On Combined Symbol-and-Bit Error-Control [4, 2] Codes over $\{0,1\}^8$ to Be Used in the $(4,2)$ Concept Fault-Tolerant Computer," *IEEE Trans. Info. Theory*, IT-33 (November 1987): 911–917.

[GOLA84] P. Golan, "Design of Totally Self-Checking Checker for 1-out-of-3 Code," *IEEE Trans. Comput.*, C-33 (March 1984): 285.

[HALA83] C. Halatasis, N. Gaitanis, and M. Sigala, "Fast and Efficient Totally Self-Checking Checkers for m-out-of-$(2m+1)$ Codes," *IEEE Trans. Comput.*, C-32 (May 1983): 507–511.

[HALB84] M. P. Hallbert and S. M. Bose, "Design Approach for a VLSI Self-Checking MIL-STD-1750A Microprocessor," *Dig., 14th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1984): 254–259.

[HONG81] S. J. Hong and D. L. Ostapko, "A Simple Procedure to Generate Optimum Test Patterns for Parity Logic Networks," *IEEE Trans. Comput.*, C-30 (May 1981): 356–358.

[HSIA63] M. Y. Hsiao and F. F. Sellers, "The Carry Dependent Sum Adder," *IEEE Trans. Electron. Comput.*, EC-12 (June 1963): 265–268.

[HUAN84] K. Huang and J. A. Abraham, ''Algorithm-Based Fault-Tolerance for Matrix Operations,'' *IEEE Trans. Comput.*, C-33 (June 1984): 518–528.

[HUGH84] J. L. A. Hughes, E. J. McCluskey, and D. J. Lu, ''Design of Totally Self-Checking Comparators with an Arbitrary Number of Inputs,'' *IEEE Trans. Comput.*, C-33 (January 1984): 546–550.

[ITOH80] H. Itoh and M. Nakamichi, ''Design of Self-Checking Checkers for Berger Code and *m*-out-of-*n* Code'' (in Japanese), *Trans. IECE Japan*, J63-D (April 1980): 326–331.

[ITOH82] H. Itoh and M. Nakamichi, ''Self-Checking Checker Designs for Various 2-Rail Codes,'' *Trans. IECE Japan*, E65 (November 1982): 665–671.

[IZAW81] N. Izawa, ''3-Level Realization of Self-Checking 1-out-of-*n* Code Checkers'' (in Japanese), *Dig., 1981 IECE Nat. Convention of Information Systems, IECE of Japan*, 504 (October 1981).

[IZAW84a] N. Izawa, ''Design of Totally Self-Checking Checkers for Unordered Codes'' (in Japanese), *Trans. IECE Japan*, J67-D (May 1984): 585–592.

[IZAW84b] N. Izawa, ''Reduced-Gate Design of Totally Self-Checking Checkers for Unordered Codes'' (in Japanese), *Trans. IECE Japan*, J67-D (December 1984): 1395–1402.

[JANS85] I. Jansch and B. Courtois, ''Strongly Language Disjoint Checking,'' *Dig., 15th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1985): 390–395.

[JHA84] N. K. Jha and J. A. Abraham, ''The Design of Totally Self-Checking Embedded Checkers,'' *Dig., 14th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1984): 265–270.

[KHAK82a] J. Khakbaz and E. J. McCluskey, ''Concurrent Error Detection and Testing for Large PLA's,'' *IEEE J. Solid-State Circ.*, SC-17 (April 1982): 386–394.

[KHAK82b] J. Khakbaz, ''Self-Testing Embedded Parity Trees,'' *Dig., 12th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1982): 109–116.

[KHAK82c] J. Khakbaz, ''Totally Self-Checking Checker for 1-out-of-*n* Code Using Two-Rail Codes,'' *IEEE Trans. Comput.*, C-13 (July 1982): 667–681.

[KHAK84] J. Khakbaz and E. J. McCluskey, ''Self-Testing Embedded Parity Checkers,'' *IEEE Trans. Comput.*, C-33 (August 1984): 753–794.

[KHOD79] B. Khodadad-Mostershiry, ''Parity Prediction in Combinational Circuit,'' *Dig., 9th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1979): 185–188.

[KONE80] B. K*ö*nemann, J. Mucha, and G. Zwiehoff, ''Built-in Test for Complex Digital Integrated Circuits,'' *IEEE J. Solid-State Circ.*, SC-15 (June 1980): 315–319.

[KORE86] I. Koren and D. K. Pradhan, ''Yield and Performance Enhancement Through Redundancy in VLSI and WSI Multiprocessor Systems,'' *Proc. IEEE*, 74 (May 1986): 699–711.

[KROL86] T. Krol, ''(N, K) Concept Fault Tolerance,'' *IEEE Trans. Comput.*, C-35 (April 1986): 339–349.

[LALA85] P. K. Lala, *Fault Tolerant and Fault Testable Hardware Design*, Prentice-Hall (1985).

[LALA01] P. K. Lala, *Self-Checking and Fault-Tolerant Digital Design*, Morgan Kaufmann (2001).

[LANG70] G. G. Langdon Jr. and C. K. Tang, ''Concurrent Error Detection for Group Look-Ahead Binary Adders,'' *IBM J. Res. Dev.*, 14 (September 1970): 563–573.

[LO87a] J. C. Lo and S. Thanwastien, ''The Design of Fast Totally Self-Checking Berger Codes Checkers Based on Berger Code Partitioning,'' Research Paper, University of Southwestern Louisiana, Lafayette (March 25, 1987). (Also in *Dig., 18th IEEE Int. Symp. on Fault-Tolerant Computing* [June 1988].)

[LO87b] J. C. Lo and S. Thanawastien, ''On the Design of Combinational Totally Self-Checking 1-out-of-3 Code Checkers,'' Research Paper, University of Southwestern Louisiana, Lafayette (May 25, 1987).

[MACC85] E. J. McCluskey, ''Built-In Self-Test Techniques, Built-In Self-Test Structures,'' *IEEE Design Test, 2* (April 1985): 21–28, 29–36.

[MAK82] G. P. Mak, J. A. Abraham, and E. S. Davidson, "The Design of PLAs with Concurrent Error Detection," *Dig., 12th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1982): 303–310.

[MAKI74] G. Maki and D. H. Sawain III, "Fault-Tolerant Asynchronous Sequential Machines," *IEEE Trans. Comput.*, C-23 (July 1974): 651–657.

[MAND72] D. Mandelbaum, "Error Correction in Residue Arithmetic," *IEEE Trans. Comput.*, C-21 (June 1972): 538–545.

[MARO78a] M. A. Marouf and A. D. Friedman, "Design of Self-Checking Checkers for Berger Codes," *Dig., 8th IEEE Int. Symp. on Fault-Tolerant Computing* (1978): 179–184.

[MARO78b] M. A. Marouf and A. D. Friedman, "Efficient Design of Self-Checking Checker for Any *m*-out-of-*n* Code," *IEEE Trans. Comput.*, C-27 (June 1978): 482–490.

[MCIN05] H. McIntyre, D. Wendell, K. J. Lin, P. Kaushik, S. Seshadri, A. Wang, V. Sundararaman, P. Wang, S. Kim, W-J. Hsu, H-C. Park, G. Levinsky, J. Lu, M. Chirania, R. Heald, P. Lazar, and S. Dharmasena, "A 4-MB On-Chip L2 Cache for a 90-nm 1.6-GHz 64-Bit Microprocessor," *IEEE J. Solid-State Circ.*, 40 (January 2005): 52–59.

[MONT72] P. M. Monteiro and T. R. N. Rao, "A Residue Checker for Arithmetic and Logical Operations," *Dig., 12th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1972): 8–13.

[MOUR86a] S. Mourad, J. L. A. Hughes, and E. J. McCluskey, "Multiple Fault Detection in Parity Trees," *Proc. IEEE Compcon Spring* (1986): 441–444.

[MOUR86b] S. Mourad, J. L. A. Hughes, and E. J. McCluskey, "Stuck-at Fault Detection in Parity Trees," *Proc. IEEE Fall Joint Computer Conf.* (September 1986): 836–840.

[MUEL99] M. Mueller, L. C. Alves, W. Fisher, M. L. Fair, and I. Modi, "RAS Strategy for IBM S/390 G5 and G6," *IBM J. Res. Dev.*, 43 (September–November 1999): 875–888.

[MUKA74] Y. Mukai and Y. Tohma, "A Method for the Realization of Fail-Safe Asynchronous Sequential Circuits," *IEEE Trans. Comput.*, C-23 (July 1974): 736–739.

[MUKA76] Y. Mukai and Y. Tohma, "A Masked-Fault Free Realization of Fail-Safe Asynchronous Sequential Circuits," *Dig., 6th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1976): 69–74.

[NAKA05] M. Nakai, S. Akui, K. Seno, T. Meguro, T. Seki, T. Kondo, A. Hashiguchi, H. Kawahara, K. Kumano, and M. Shimura, "Dynamic Voltage and Frequency Management for a Low-Power Embedded Microprocessor," *IEEE J. Solid-State Circ.*, 40 (January 2005): 28–35.

[NANY79] T. Nanya and Y. Tohma, "Universal Multicode STT State Assignments for Asynchronous Sequential Machines," *IEEE Trans. Comput.*, C-28 (November 1979): 811–818.

[NANY83] T. Nanya and Y. Tohma, "A 3-Level Realization of Totally Self-Checking Checkers for *m*-out-of-*n* Codes," *Dig., 13th IEEE Int. Symp. on Fault-Tolerant Computing* (1983): 173–176.

[NANY85] T. Nanya and T. Hamamatsu, "Totally Self-Checking Checkers for Subsets of *m*-out-of-2*m* Codes" (in Japanese), *Trans. IECE Japan*, J68-D (March 1985): 229–236.

[NANY87] T. Nanya and T. Kawamura, "A Note on Strongly Fault Secure Sequential Circuits," *IEEE Trans. Comput.*, C-36 (September 1987): 1121–1123.

[NANY88] T. Nanya and T. Kawamura, "Error Secure / Error Propagating Concept and Its Application to Design of Strongly Fault Secure Processors," *IEEE Trans. Comput.*, C-37 (January 1988): 14–24.

[NEUM75] P. G. Neumann and T. R. N. Rao, "Error-Correction Codes for Byte-Organized Arithmetic Processors," *IEEE Trans. Comput.*, C-24 (March 1975): 226–232.

[NICO84] M. Nicolaidis, I. Jansch, and B. Courtois, "Strongly Code-Disjoint Checkers," *Dig., 14th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1984): 16–21.

[NICO85] M. Nicolaidis, "Evaluation of a Self-Checking Version of the MC68000 Microprocessor," *Dig., 15th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1985): 350–356.

[OBER79] R. M. M. Oberman, *Digital Circuits for Binary Arithmetic*, Macmillian (1979).

[OZGU77] F. Özgüner, "Design of Totally Self-Checking Asynchronous and Synchronous Sequential Machines," *Dig., 7th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1977): 124–129.

[PASC88] A. M. Paschalis, D. Nikolos, and C. Halatsis, "Efficient Modular Design of TSC Checkers for *m*-out-of-2*m* Codes," *IEEE Trans. Comput.*, C-37 (March 1988): 301–309.

[PATE83] J. H. Patel and L. Y. Fung, "Concurrent Error Detection in Multiply and Divide Arrays," *IEEE Trans. Computers*, C-32 (April 1983): 417–422.

[PETE58] W. W. Peterson, "On Checking Adder," *IBM J.* (April 1958): 166–168.

[PETE59] W. W. Peterson and M. O. Rabin, "On Codes for Checking Logical Operations," *IBM J. Res. Dev.*, 3 (1959): 163–168.

[PIER65] W. H. Pierce, "Interconnection Structure for Redundant Logic (The Theory of Interwoven Redundant Logic)," in *Failure-Tolerant Computer Design*, Academic Press (1965), ch. 5.

[PIES83] S. J. Pristrak, "Design Method of Totally Self-Checking Checkers for *m*-out-of-*n* Codes," *Dig., 13th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1983): 162–168.

[PIES85] S. J. Piestrak, "PLA Implementations of Totally Self-Checking Circuits Using M-out-of-N Codes," *Proc. IEEE Int. Conf. on Computer Design* (October 1985): 777–781.

[PIES87] S. J. Piestrak, "Design of Fast Self-Testing Checkers for a Class of Berger Codes," *IEEE Trans. Comput.*, C-36 (May 1987): 629–634.

[PRAD72a] D. K. Pradhan and S. M. Reddy, "A Design Technique for Synthesis of Fault-Tolerant Adders," *Dig., 12th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1972): 20–24.

[PRAD72b] D. K. Pradhan and S. M. Reddy, "Error Control Techniques for Logic Processors," *IEEE Trans. Comput.*, C-21 (December 1972): 1331–1337.

[PRAD73] D. K. Pradhan and S. M. Reddy, "Fault-Tolerant Asynchronous Networks," *IEEE Trans. Comput.*, C-22 (July 1973): 662–669.

[PRAD74] D. K. Pradhan, "Fault-Tolerant Carry-Save Adders," *IEEE Trans. Comput.*, C-23 (December 1974): 1320–1322.

[PRAD76] D. K. Pradhan and S. M. Reddy, "Techniques to Construct (2, 1) Separating Systems from Linear Error-Correcting Codes," *IEEE Trans. Comput.*, C-25 (September 1976): 945–949.

[PRAD78] D. K. Pradhan, "Asynchronous State Assignments with Unateness Properities and Fault-Secure Design," *IEEE Trans. Comput.*, C-27 (May 1978): 396–404.

[RAO68b] T. R. N. Rao, "Error-Checking Logic for Arithmetic-Type Operations of a Processor," *IEEE Trans. Comput.*, C-17 (September 1968): 845–849.

[RAO70] T. R. N. Rao, "Biresidue Error-Correcting Codes for Computer Arithmetic," *IEEE Trans. Comput.*, C-19 (May 1970): 398–402.

[RAO71] T. R. N. Rao and O. N. Garcia, "Cyclic and Multiresidue Codes for Arithmetic Operations," *IEEE Trans. Info. Theory*, IT-17 (January 1971): 85–91.

[RAO72] T. R. N. Rao, "Error Correction in Adders Using Systemtic Subcodes," *IEEE Trans. Comput.*, C-21 (March 1972): 254–259.

[RAO74] T. R. N. Rao, *Error Coding for Arithmetic Processors*, Academic Press (1974).

[RAO77] T. R. N. Rao and H. J. Reinheimer, "Fault-Tolerant Modularized Arithmetic Logic Unites," *Proc. Nat. Computer Conf.*, AFIPS (1977): 703–710.

[REED70] I. S. Reed, "Error Tolerant Sequential Circuits," US Patent 3529141 (September 15, 1970).

[REDD72a] S. M. Reddy, "Easily Testable Realization for Logic Functions," *IEEE Trans. Comput.*, C-21 (November 1972): 1183–1188.

[REDD72b] S. M. Reddy, "Easily Testable Relization for Logic Functions," *IEEE Trans. Comput.*, C-21 (December 1972): 1421–1426.

[REDD72c] S. M. Reddy, "A Design Procedure for Fault-Locatable Switching Circuits," *IEEE Trans. Comput.*, C-21 (December 1972): 1421–1426.

[REDD74a] S. M. Reddy and J. R. Wilson, "Easily Testable Cellular Realization for the (Exactly *p*)-out-of-*n* and (*p* or More)-out-of-*n* Logic Functions," *IEEE Trans. Comput.*, C-23 (January 1974): 98–100.

[REDD74b] S. M. Reddy, "A Note on Self-Checking Chechers," *IEEE Trans. Comput.*, C-23 (October 1974): 1100–1102.

[REDD85] S. M. Reddy, K. K. Saluja, and M. Karpovsky, "A Data Compression Technique for Built-in Self-Test," *Dig., 15th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1985): 294–299.

[RENN84] D. A. Rennels, "Fault-Tolerant Computing—Concepts and Examples," *IEEE Trans. Comput.*, C-33 (December 1984): 1116–1129.

[RUSU03] S. Rusu, J. Stinson, S. Tam, J. Leung, H. Muljono, and B. Cherkauer, "A 1.5-GHz 130-nm Itanium® 2 Processor with 6-MB On-Die L3 Cache," *IEEE J. Solid-State Circ.*, 38 (November 2003): 1887–1895.

[SAWI74] D. H. Sawin III, and G. K. Maki, "Asynchronous Sequential Machines Designed for Fault Detection," *IEEE Trans. Comput.*, C-23 (July 1974): 239–249.

[SEDM79] R. M. Sedmak, "Design for Self-Verfication: An Approach for Dealing with Testability Problems in VLSI-Based Designs," *Proc. IEEE Int. Test Conf.* (1979): 112–120.

[SEDM80a] R. M. Sedmak, "Implementation Techniques for Self-Verification," *Proc. IEEE Int. Test Conf.* (1980): 267–278.

[SEDM80b] R. M. Sedmak and H. L. Liebergot, "Fault Tolerant of a General Purpose Computer Implemented by Very Large Scale Integration," *IEEE Trans. Comput.*, C-29 (June 1980): 492–500.

[SELL68] F. F. Sellers Jr., M. Y., Hsiao, and L. W. Bearnson, *Error Detecting Logic for Digital Computers*, McGraw-Hill (1968).

[SHED78] J. J. Shedletsky, "Error Correction by Alternate-Data Retry," *IEEE Trans. Comput.*, C-27 (February 1978): 106–112.

[SHIN05] J. L. Shin, B. Petride, M. Singh, and A. S. Leon, "Design and Implementation of an Embedded 512-KB Level-2 Cache Subsystem," IEEE Journal of Solid-State Circuit, 40 (September 2005): 1815–1820.

[SIEW82] D. P. Siewiorek and R. Swarz, *The Theory and Practice of Reliability Systems*, Digital Press (1982).

[SMIT76] J. E. Smith, "The Design of Totally Self-Checking Combinational Circuits," Report of Coordinated Science Labs, University of Illinois, R-737 (February 1976).

[SMIT77] J. E. Smith, "The Design of Totally Self-Checking Check Circuits for a Class of Unordered Codes," *J. Design Autom. Fault-Tolerant Comput.*, 1 (August 1977): 321–342.

[SMIT78] J. E. Smith and G. Mertze, "Strongly Fault Secure Logic Networks," *IEEE Trans. Comput.*, C-27 (June 1978): 495–499.

[SMIT83] J. E. Smith and P. Lam, "A Theory of Totally Self-Checking System Design," *IEEE Trans. Comput.*, C-32 (September 1983): 831–844.

[SPAI99] L. Spainhower, and T. A. Gregg, "IBM S/390 Parallel Enterprise Server G5 Fault Tolerance : A Historical Perspective," *IBM J. Res. Dev.*, 43 (September–November 1999): 863–873.

[STIN03] J. Stinson, and S. Rusu, "A 1.5GHz Third Generation Itanium® Processor," *2003 IEEE Int. Solid-State Circuits Conf.*, 14.4 (February 2003).

[TAKA05] T. Takayanagi, J. L. Shin, B. Petrick, J. Y. Su, H. Levy, H. Pham, J. Son, N. Moon, D. Bistry, U. Nair, M. Singh, V. Mathur, and A. S. Leon, "A Dual-Core 64-Bit UltraSPARC Microprocessor for Defence Server Applications," *IEEE J. Solid-State Cir.*, 40 (January 2005): 7–17.

[TAKE80] K. Takeda and Y. Tohma, "Logic Design of Fault-Tolerant Arithmetic Units Based on the Data Complementation Strategy," *Dig., 10th IEEE Int. Symp. on Fault-Tolerant Computing* (October 1980): 348–350.

[TAMI84] Y. Tamir and C. H. Sequin, ''Design and Application of Self-Testing Comparators Implemented with MOS PLA's,'' *IEEE Trans. Comput.* C-33 (June 1984): 494–506.

[TANG83] D. T. Tang and L. S. Woo, ''Exhaustive Test Pattern Generation with Constant Weight Vectors,'' *IEEE Trans. Comput.*, C-32 (December 1983): 1145–1150.

[TANG84] D. T. Tang and C. L. Chen, ''Logic Test Pattern Generation Using Linear Codes,'' *IEEE Trans. Comput.*, C-33 (September 1984): 845–850.

[TAO86] D. L. Tao, P. K. Lala, and C. R. P. Hartmann, ''A Concurrent Strategy for PLAs,'' *Proc. IEEE Int. Test Conf.* (1986): 705–709.

[TAO87] D. L. Tao and P. K. Lala, ''Three Level Totally Self-Checking Checker for 1-out-of-*n* Code,'' *Dig., 17th IEEE Symp. on Fault-Tolerant Computing* (July 1987): 108–113.

[TEND02] J. M. Tendler, J. S. Dodson, J. S. Field Jr., H. Le, and B. Sinharoy, ''POWER 4 System Microarchitecture,'' *IBM J. Res. Dev.*, 46 (January 2002): 5–25.

[TOHM71] Y. Tohma, Y. Ohyama, and R. Sakai, ''Realization of Fail-Safe Sequential Machines by Using *k*-out-of-*n* Code,'' *IEEE Trans. Comput.*, C-20 (November 1971): 1270–1275.

[TOHM74] Y. Tohma, ''Design Techique of Fail-Safe Circuits Using Flip-Flops for Internal Memory,'' *IEEE Trans. Comput.*, C-23 (November 1974): 1149–1154.

[TOHM86] Y. Tohma, ''Coding Techniques in Fault-Tolerant, Self-Checking and Fail-Safe Circuits,'' in D. K. Pradhan (ed.), *Fault-Tolerant Computing, Theory and Techniques*, Prentice-Hall (1986), ch. 5.

[TOY78] W. N. Toy, ''Fault-Tolerant Design of Local ESS Processor,'' *Proc. IEEE*, 66 (October 1978): 1126–1145.

[TSAO82] M. M. Tsao, A. W. Wilson, R. C. McGarity, C. T. Tseng, and D. P. Siewiorek, ''The Design of C. Fast: Single Chip Fault Tolerant Microprocessor,'' *Dig., 12th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1982): 63–69.

[VIAU80] J. Viaud and R. David, ''Sequentially Self-Checking Circuits,'' *Dig., 10th IEEE Int. Symp. on Fault-Tolerant Computing* (October 1980): 263–268.

[WAKE74] J. F. Wakerly, ''Partially Self-Checking Circuits and Their Use in Performing Logical Operations,'' *IEEE Trans. Comput.*, C-23 (July 1974): 658–666.

[WAKE76] J. F. Wakerly, ''Checked Binary Addition with Checksum Codes,'' *J. Design Autom. Fault-Tolerant Comput.*, 1 (October 1976): 18–27.

[WAKE78] J. F. Wakerly, *Error Detecting Codes, Self-Checking Circuits, and Applications*, North-Holland (1978).

[WANG79] S. L. Wang and A. Avizienis, ''The Design of Totally Self-Checking Circuits Using Programmable Logic Arrays,'' *Dig., 9th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1979): 173–180.

[WILL82] T. W. Williams and K. P. Parker, ''Design for Testability—A Survey,'' *IEEE Trans. Comput.*, C-31 (January 1982): 2–5.

[WONG83] C. Y. Wong, W. K. Fuchs, J. A. Abraham, and E. S. Davidson, ''The Design of a Microprogram Control Unit with Concurrent Error Detection,'' *Dig., 13th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1983): 476–483.

[YEN87] M. M. Yen, W. K. Fuchs, and J. A. Abraham, ''Designing for Concurrent Error Detection in VLSI: Application to a Microprogram Control Unit,'' *IEEE J. Solid-State Cir.*, SC-22 (August 1987): 595–605.

CONTENTS

13

Codes for Data Entry Systems

Nonbinary M-ary words processed by data entry systems often suffer from *asymmetric errors*. In character recognition systems, for example, two symbols a_i and a_j with similar shapes have a high probability of being mistaken for one another. Among the many types of data processed by data entry systems, M-ary words selected from a specified codebook, such as postal codes and product numbers, should be strongly protected from asymmetric errors because these words are often used for indexing a database. This chapter presents two types of M-ary asymmetric error correcting codes, that is, systematic codes and nonsystematic codes, that can be utilized to generate these codebooks.

In the data entry systems such as keyboard input systems and character recognition systems, some types of *human-made errors* may add to the asymmetric errors. These errors are, for example, *symbol deletion / insertion errors* in the keyboard input systems, or *adjacent-symbol-transposition errors* in the keyboard input systems or the handwritten character recognition systems. This chapter also presents another class of M-ary asymmetric error correcting codes capable of correcting single deletion / insertion / adjacent-symbol-transposition errors as well as correcting single asymmetric errors.

As another data entry systems, new types of bar codes (i.e., two-dimensional matrix symbols) have been popularly used in various sales items and products. This chapter discusses quick response codes (i.e., QR codes) and two-dimensional unidirectional clustered error correcting codes for high-density two-dimensional matrix symbols.

13.1 *M*-ARY ASYMMETRIC ERRORS IN DATA ENTRY SYSTEMS

Improved reliability is strongly required for data entry systems, for example, keyboard input systems and character recognition systems, because they often suffer from errors such as mis-typing and mis-identification. Among the many types of data processed by

Code Design for Dependable Systems: Theory and Practical Applications, by Eiji Fujiwara

data entry systems, M-ary words selected from a specified codebook, such as postal codes, product numbers, bank account numbers, and driver's license numbers, should be strongly protected from errors because these words are often used for indexing a database. Errors in these M-ary words can be corrected or detected by applying M-ary error control codes [GALL96, TANG70].

Generally, the number M of symbols used in data entry systems is neither prime nor the power of a prime, for instance, $M = 10$ for numerals and $M = 10 + 26 + 26 = 62$ for alphanumeric symbols using upper-case and lower-case letters as well as numerals. Therefore conventional error control codes defined over a Galois field cannot be directly applied to M-ary words in most cases. To overcome this problem, systematic M-ary error correcting codes have been designed based on the prime factorization of M-ary symbols [BROW73]. Recently a class of systematic M-ary single-symbol error correcting codes has been designed using a prime field and an integer residue ring [NAMB01]. As an extension to this class of codes, systematic M-ary single-symbol error correcting codes capable of correcting adjacent-symbol transposition errors have been proposed [SUZU98].

The aforementioned M-ary error control codes are designed based on the assumption that the error is *symmetric*, which means that each symbol in a codeword may be erroneously changed to another symbol with equal probability. In data entry systems, however, errors are generally *asymmetric*; that is to say, the probability of a symbol a_i being mistaken for another symbol a_j, denoted by $p(a_j|a_i)$, is generally not equal to $p(a_k|a_i)$, where $a_j \neq a_k$. For example, in character recognition systems, the probability of a 7 being mistaken for a 9 is much higher than that of a 7 being mistaken for a 4, or $p(9|7) \gg p(4|7)$, because the numerals 7 and 9 are similar in shape whereas 7 and 4 are dissimilar in shape. Likewise, in keyboard input systems, symbols located on adjacent keys are mistaken for one another with high probability. Based on this observation, systematic M-ary asymmetric single-symbol error locating codes [SAOW01] have been designed. Several classes of asymmetric symbol error control codes for M-ary channels have been proposed in [VARS73], [SAIT90a]. These codes correct either additive errors, where transmitted integer u $(u \in \{0, 1, \cdots, M-1\})$ is changed to $u + \varepsilon$ $(\varepsilon > 0, u + \varepsilon \leq M - 1)$, or subtractive errors, where u is changed to $u - \varepsilon$ $(\varepsilon > 0, u - \varepsilon \geq 0)$. On the other hand, errors in data entry systems cannot be expressed as additive or subtractive errors. Therefore the codes for M-ary channels are not applicable to these systems. An asymmetric symbol error correction scheme for character recognition systems have been proposed in [INAB94]. This scheme employs concatenated codes to correct asymmetric symbol errors.

13.2 *M*-ARY ASYMMETRIC SYMBOL ERROR CORRECTING CODES

This section presents a new class of M-ary single asymmetric symbol error correcting codes that are far more efficient than the existing single symmetric symbol error correcting codes [KANE04a].

13.2.1 Systematic Codes

A new class of systematic M-ary single asymmetric symbol error correcting codes is presented in this subsection. This class of codes is designed based on a new class of rings, and injective and surjective mappings.

1. Preliminaries

$\mathcal{E}$-Asymmetric Symbol Error

Definition 13.1 Let $\mathbf{A} = \{a_0, a_1, \cdots, a_{M-1}\}$ be a set of M-ary symbols. An *asymmetric symbol error set* $\mathcal{E}$ is defined as follows:

$$\mathcal{E} = \{(a_i \rightarrow a_j) \mid a_i, a_j \in \mathbf{A}, \Pr(a_j|a_i) > T, 0 \leq i \neq j \leq M-1\},$$

where $\Pr(a_j|a_i)$ is the probability of an error such that a_i is changed into a_j, and T is a threshold error probability given in advance. □

Definition 13.2 Let $(a_i \rightarrow a_j) \in \mathcal{E}$. An error where a_i is changed into a_j is called an *$\mathcal{E}$-asymmetric symbol error.* □

Definition 13.3 An *error directionality graph* G corresponding to the asymmetric symbol error set $\mathcal{E}$ is defined as $G = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V} = \mathbf{A} = \{a_0, a_1, \cdots, a_{M-1}\}$ is a set of vertices and $\mathbf{E} = \mathcal{E} = \{(a_i \rightarrow a_j) \mid \Pr(a_j|a_i) > T,\ 0 \leq i \neq j \leq M-1\}$ is a set of edges. □

An edge $(a_i \rightarrow a_j) \in \mathbf{E}$ of the error directionality graph G indicates that the probability of a_i being changed into a_j is larger than T.

Example 13.1 [KANE04a]

As an example of asymmetric symbol errors in data entry devices, Table 13.1 shows the confusion matrix of handwritten numeral recognition systems [NOUM93]. The value at the intersection of the i-th row and the j-th column in the table indicates a probability where handwritten numeral i is recognized as numeral j. These values are calculated

TABLE 13.1 Confusion Matrix of Handwritten Numeral Recognition Systems

Handwritten numeral i \ Recognized numeral j	0	1	2	3	4	5	6	7	8	9
0	0.9957	0.0011	0.0002	0.0004	0	0	0.0013	0.0011	0	0.0002
1	0	0.9991	0.0009	0	0	0	0	0	0	0
2	0	0	0.9983	0.0002	0	0	0	0.0007	0.0007	0
3	0	0	0.0009	0.9985	0	0	0	0.0004	0	0.0002
4	0	0.0002	0	0	0.9988	0	0.0008	0	0	0.0002
5	0	0.0007	0	0.0007	0.0007	0.9972	0.0002	0	0.0005	0
6	0.0012	0.0003	0	0	0.0005	0	0.9978	0	0.0003	0
7	0	0.0003	0.0005	0.0010	0	0	0	0.9948	0	0.0035
8	0	0	0	0	0	0.0007	0.0002	0	0.9990	0
9	0.0002	0.0010	0.0002	0.0002	0.0005	0	0	0.0014	0.0005	0.9959

Source: *[KANE04a].* © *2004 IEEE.*

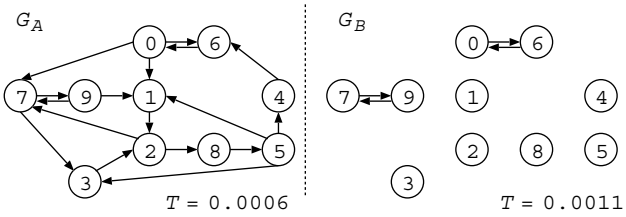


Figure 13.1 *Example of error directionality graphs for handwritten numeral recognition systems.* Source: *[KANE04a]. © 2004 IEEE.*

from statistical data [NOUM93]. The asymmetric symbol error set $\mathcal{E}_A$ in these systems is presented as follows:

$$\begin{aligned}\mathcal{E}_A = \{&(0 \to 1), (0 \to 6), (0 \to 7), (1 \to 2), (2 \to 7), (2 \to 8),\\ &(3 \to 2), (4 \to 6), (5 \to 1), (5 \to 3), (5 \to 4), (6 \to 0),\\ &(7 \to 3), (7 \to 9), (8 \to 5), (9 \to 1), (9 \to 7)\},\end{aligned}$$

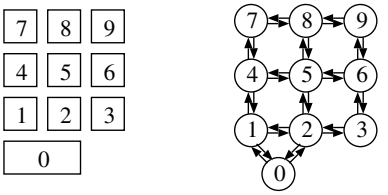
where the threshold error probability is $T = 0.0006$. Another asymmetric symbol error set $\mathcal{E}_B$ with $T = 0.0011$ is given by

$$\mathcal{E}_B = \{(0 \to 6), (6 \to 0), (7 \to 9), (9 \to 7)\}.$$

The error directionality graphs G_A and G_B based, respectively, on $\mathcal{E}_A$ and $\mathcal{E}_B$ are shown in Figure 13.1.

As another data entry device, the symbols of two adjacent keys in the numeric keypads have high error probabilities because adjacent keys are sometimes mis-tapped. Figure 13.2(a) illustrates the typical layout of a numeric keypad. Although the value of T is not explicitly indicated, the following asymmetric symbol error set $\mathcal{E}_C$ can be presented:

$$\begin{aligned}\mathcal{E}_C = \{&(0 \to 1), (0 \to 2), (1 \to 0), (1 \to 2), (1 \to 4), (2 \to 0),\\ &(2 \to 1), (2 \to 3), (2 \to 5), (3 \to 2), (3 \to 6), (4 \to 1),\\ &(4 \to 5), (4 \to 7), (5 \to 2), (5 \to 4), (5 \to 6), (5 \to 8),\\ &(6 \to 3), (6 \to 5), (6 \to 9), (7 \to 4), (7 \to 8), (8 \to 5),\\ &(8 \to 7), (8 \to 9), (9 \to 6), (9 \to 8)\}.\end{aligned}$$



(a) Typical layout of numeric keypad

(b) Error directionality graph G_C

Figure 13.2 *Asymmetric errors in numeric keypad.* Source*: [KANE04a]. © 2004 IEEE.*

This error set includes all pairs of symbols corresponding to adjacent keys in the numeric keypad. Figure 13.2(b) shows the error directionality graph G_C based on $\mathcal{E}_C$.

Definition 13.4 Let $\mathbf{u} = (u_0, u_1, \cdots, u_{N-1})$ be a codeword of code $\mathbf{C}$ over the set of M-ary symbols $\mathbf{A} = \{a_0, a_1, \cdots, a_{M-1}\}$, i.e., $u_i \in \mathbf{A}$ $(0 \le i \le N-1)$. If the code $\mathbf{C}$ can correct every single-symbol error $(u_i \to u_i') \in \mathcal{E}$, then $\mathbf{C}$ is an M-ary single $\mathcal{E}$-asymmetric symbol error correcting code. □

Bound for Single $\mathcal{E}$-Asymmetric Symbol Error Correcting Codes Here we observe the systematic M-ary codes that are capable of correcting single $\mathcal{E}$-asymmetric symbol errors that occur in the check part as well as those in the information part. However, in order to derive the upper bound on the information symbol length of the codes, Lemma 13.1 deals with the codes capable of correcting single $\mathcal{E}$-asymmetric symbol errors only in the information part. Define the following functions:

$$\delta(a_i) = \Big|\{(a_j \to a_i) \,|\, (a_j \to a_i) \in \mathbf{E} = \mathcal{E}\}\Big|,$$
$$\Delta(G) = \max_{i \in \{0,1,\cdots,M-1\}} \delta(a_i),$$

where $|\mathbf{X}|$ denotes the cardinality of $\mathbf{X}$. In other words, $\delta(a_i)$ is the indegree of a_i, and $\Delta(G)$ is the maximum indegree of vertices in G.

Lemma 13.1 *A systematic code that corrects single $\mathcal{E}$-asymmetric symbol errors in the information part exists only if*

$$k \le \left\lfloor \frac{M^r - 1}{\Delta(G)} \right\rfloor,$$

where $\lfloor x \rfloor$ shows the maximum integer less than or equal to x, k is the information-symbol length, r the check-symbol length, and G the error directionality graph based on $\mathcal{E}$.

Proof Let $\mathbf{u}_1$ and $\mathbf{u}_2$ be any two distinct codewords expressed by

$$\mathbf{u}_1 = (\underbrace{\widetilde{a}, \cdots, \widetilde{a}}_{s}, \underbrace{a', \widetilde{a}, \cdots, \widetilde{a}}_{k-s}, \underbrace{p_{1,0}, \cdots, p_{1,r-1}}_{r}),$$
$$\mathbf{u}_2 = (\underbrace{\widetilde{a}, \cdots, \widetilde{a}}_{t}, \underbrace{a'', \widetilde{a}, \cdots, \widetilde{a}}_{k-t}, \underbrace{p_{2,0}, \cdots, p_{2,r-1}}_{r}),$$

where

$$\widetilde{a}, a', a'', p_{1,j}, p_{2,j} \in V = A \ (0 \le j \le r-1),$$
$$\delta(\widetilde{a}) = \Delta(G), 0 \le s, t \le k-1,$$
$$(a' \to \widetilde{a}) \in \mathcal{E},$$

and $(a'' \to \widetilde{a}) \in \mathcal{E}$. If $\mathbf{u}_1$ and $\mathbf{u}_2$ have identical check parts (i.e., $p_{1,j} = p_{2,j} = p_j$ for $0 \le j \le r-1$), then both $\mathbf{u}_1$ and $\mathbf{u}_2$ may be changed into the following identical word by a single $\mathcal{E}$-asymmetric symbol error in the information part:

$$\mathbf{u}' = (\underbrace{\widetilde{a}, \widetilde{a}, \cdots, \widetilde{a}}_{k}, \underbrace{p_0, \cdots, p_{r-1}}_{r}).$$

In this case errors in $\mathbf{u}_1$ and $\mathbf{u}_2$ cannot be corrected. Therefore all codewords with the following information part should not have an identical check part:

$$\begin{array}{cccccccc} (\widetilde{a} & \cdots & \widetilde{a} & a' & \widetilde{a} & \cdots & \widetilde{a}), \\ 0 & \cdots & i-1 & i & i+1 & \cdots & k-1 \end{array}$$

where $(a' \rightarrow \widetilde{a}) \in \mathcal{E}$ or $a' = \widetilde{a}$, and $0 \leq i \leq k-1$. So the inequality

$$k\Delta(G) + 1 \leq M^r$$

is satisfied because there are $k\Delta(G) + 1$ codewords with this property. Subsequently the inequality in Lemma 13.1 holds. Q.E.D.

This lemma indicates that any code capable of correcting single $\mathcal{E}$-asymmetric symbol errors in the information part has at most $\lfloor (M^r - 1)/\Delta(G) \rfloor$ information symbols. Since M-ary single $\mathcal{E}$-asymmetric symbol error correcting codes include this error correction capability, the information symbol length of these codes never exceeds $\lfloor (M^r - 1)/\Delta(G) \rfloor$. The following theorem is obvious.

Theorem 13.1 *A systematic M-ary single $\mathcal{E}$-asymmetric symbol error correcting code exists only if*

$$k \leq \left\lfloor \frac{M^r - 1}{\Delta(G)} \right\rfloor.$$

Rings We define a new class of rings on which the M-ary single $\mathcal{E}$-asymmetric symbol error correcting codes are designed.

Definition 13.5 Let c be a positive integer, and also q_i be a prime number or a power of a prime number, where $1 \leq i \leq c$. A set $R(q_1,\ q_2,\ \cdots,\ q_c)$ is defined by

$$R(q_1,\ q_2,\ \cdots,\ q_c) = \{\langle x_1,\ x_2,\ \cdots,\ x_c\rangle \,|\, x_i \in GF(q_i),\ 1 \leq i \leq c\}.$$

Let $x = \langle x_1,\ x_2,\ \cdots,\ x_c\rangle$ and $y = \langle y_1,\ y_2,\ \cdots,\ y_c\rangle$ be elements in $R(q_1,\ q_2,\ \cdots, q_c)$. Addition $(+)$ and multiplication $(\times)$ in $R(q_1,\ q_2,\ \cdots,\ q_c)$ are defined as follows:

$$\begin{aligned} x + y &= \langle x_1,\ x_2,\ \cdots,\ x_c\rangle + \langle y_1,\ y_2,\ \cdots,\ y_c\rangle \\ &= \langle (x_1 +_1 y_1),\ (x_2 +_2 y_2),\ \cdots,\ (x_c +_c y_c)\rangle, \\ x \times y &= \langle x_1,\ x_2,\ \cdots,\ x_c\rangle \times \langle y_1,\ y_2,\ \cdots,\ y_c\rangle \\ &= \langle (x_1 \times_1 y_1),\ (x_2 \times_2 y_2),\ \cdots,\ (x_c \times_c y_c)\rangle, \end{aligned}$$

where $+_i$ and $\times_i$ $(1 \leq i \leq c)$ are additive and multiplicative operators in $GF(q_i)$, respectively. □

Theorem 13.2 *The set $R(q_1,\ q_2,\ \cdots,\ q_c)$ is a ring.*

This theorem can easily be proved by showing that the set $R(q_1,\ q_2,\ \cdots,\ q_c)$ with the operators $+$ and $\times$ above defined satisfies the ring axioms. Therefore the proof is omitted.

A column vector over $R(q_1,\ q_2,\ \cdots,\ q_c)$ having length r is denoted by

$$\langle\langle \mathbf{x}_1,\ \mathbf{x}_2,\ \cdots,\ \mathbf{x}_c\rangle\rangle = \begin{pmatrix} \langle x_{1,1},\ x_{1,2},\ \cdots,\ x_{1,c}\rangle \\ \langle x_{2,1},\ x_{2,2},\ \cdots,\ x_{2,c}\rangle \\ \vdots \\ \langle x_{r,1},\ x_{r,2},\ \cdots,\ x_{r,c}\rangle \end{pmatrix},$$

where $x_{i,j} \in GF(q_j)$ and $\mathbf{x}_j$ is the transpose of $(x_{1,j},\ x_{2,j},\ \cdots,\ x_{r,j})$. Let $y = \langle y_1,\ y_2,\ \cdots, y_c\rangle$ be an element in $R(q_1,\ q_2,\ \cdots,\ q_c)$. The product of y and $\langle\langle \mathbf{x}_1,\ \mathbf{x}_2, \cdots,\ \mathbf{x}_c\rangle\rangle$ is defined as

$$y\langle\langle \mathbf{x}_1,\ \mathbf{x}_2,\ \cdots,\ \mathbf{x}_c\rangle\rangle = \begin{pmatrix} \langle y_1,\ y_2,\ \cdots,\ y_c\rangle \times \langle x_{1,1},\ x_{1,2},\ \cdots,\ x_{1,c}\rangle \\ \langle y_1,\ y_2,\ \cdots,\ y_c\rangle \times \langle x_{2,1},\ x_{2,2},\ \cdots,\ x_{2,c}\rangle \\ \vdots \\ \langle y_1,\ y_2,\ \cdots,\ y_c\rangle \times \langle x_{r,1},\ x_{r,2},\ \cdots,\ x_{r,c}\rangle \end{pmatrix}.$$

Similarly the product of a matrix and a vector over $R(q_1,\ q_2,\ \cdots,\ q_c)$ is simply defined as follows:

$$\begin{bmatrix} m_{0,0} & m_{0,1} & \cdots & m_{0,N-1} \\ m_{1,0} & m_{1,1} & \cdots & m_{1,N-1} \\ \vdots & \vdots & \cdots & \vdots \\ m_{r-1,0} & m_{r-1,1} & \cdots & m_{r-1,N-1} \end{bmatrix} \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{N-1} \end{pmatrix}$$

$$= \begin{pmatrix} (m_{0,0} \times v_0) + (m_{0,1} \times v_1) + \cdots + (m_{0,N-1} \times v_{N-1}) \\ (m_{1,0} \times v_0) + (m_{1,1} \times v_1) + \cdots + (m_{1,N-1} \times v_{N-1}) \\ \vdots \\ (m_{r-1,0} \times v_0) + (m_{r-1,1} \times v_1) + \cdots + (m_{r-1,N-1} \times v_{N-1}) \end{pmatrix},$$

where $m_{i,j}, v_j \in R(q_1,\ q_2,\ \cdots,\ q_c)$, $0 \le i \le r-1$, and $0 \le j \le N-1$.

Example 13.2 [KANE04a]

The ring $R(3,3)$ is defined by the following:

$$\begin{aligned} R(3,3) &= \{\langle x_1, x_2\rangle \,|\, x_1, x_2 \in GF(3)\} \\ &= \{\langle 0,\ 0\rangle, \langle 0,\ 1\rangle, \langle 0,\ 2\rangle, \langle 1,\ 0\rangle, \langle 1,\ 1\rangle, \langle 1,\ 2\rangle, \langle 2,\ 0\rangle, \langle 2,\ 1\rangle, \langle 2,\ 2\rangle\}. \end{aligned}$$

The following shows examples of addition and multiplication in $R(3,\ 3)$:

$$\begin{aligned} \langle 2,\ 0\rangle + \langle 2,\ 2\rangle &= \langle (2+2 \bmod 3), (0+2 \bmod 3)\rangle = \langle 1,\ 2\rangle, \\ \langle 2,\ 0\rangle \times \langle 2,\ 1\rangle &= \langle (2 \times 2 \bmod 3), (0 \times 1 \bmod 3)\rangle = \langle 1,\ 0\rangle. \end{aligned}$$

Product of $\langle 1,\ 2\rangle$ and $\begin{pmatrix}\langle 0,\ 1\rangle\\ \langle 2,\ 1\rangle\end{pmatrix}$ is performed as follows:

$$\langle 1,\ 2\rangle\begin{pmatrix}\langle 0,\ 1\rangle\\ \langle 2,\ 1\rangle\end{pmatrix} = \begin{pmatrix}\langle 1,\ 2\rangle \times \langle 0,\ 1\rangle\\ \langle 1,\ 2\rangle \times \langle 2,\ 1\rangle\end{pmatrix} = \begin{pmatrix}\langle 0,\ 2\rangle\\ \langle 2,\ 2\rangle\end{pmatrix}.$$

Definition 13.6 A set of *regular elements* in $R(q_1,\ q_2,\ \cdots,\ q_c)$ is denoted by $R'(q_1,\ q_2,\ \cdots,\ q_c)$:

$$R'(q_1,\ q_2,\ \cdots,\ q_c) = \{\langle x_1,\ x_2,\ \cdots,\ x_c\rangle \,|\, x_i \in GF(q_i) - \{0\},\ 1 \le i \le c\},$$

where regular element is defined as an element having its multiplicative inverse. □

2. Code Design

Parity-Check Matrix H over Rings Let $R(q_1, q_2, \cdots, q_c)$ be the ring defined by Definition 13.5, and also let $\mathbf{H}_i$ be an $r \times n_i$ parity-check matrix of a systematic single-symbol error correcting code over $GF(q_i)$ expressed by

$$\mathbf{H}_i = \begin{bmatrix} | & | & & | \\ h_{i,0} & h_{i,1} & \cdots & h_{i,n_i-1} \\ | & | & & | \end{bmatrix},$$

where $\begin{vmatrix} \\ h_{i,j} \\ \end{vmatrix}$, $0 \le j \le n_i - 1$, is a column vector with r elements in $GF(q_i)$, n_i is code length, and $1 \le i \le c$. Here the Hamming codes are applied to $\mathbf{H}_i$'s because they have the maximum code length $n_i = (q_i^r - 1)/(q_i - 1)$ for single-symbol error correction. Let $N = \prod_{i=1}^{c} n_i$. A function J_i is defined as an arbitrary mapping from $\mathbf{Z}_N = \{0,\ 1,\ \cdots,\ N-1\}$ into $\mathbf{Z}_{n_i} = \{0,\ 1,\ \cdots,\ n_i - 1\}$, satisfying

$$(J_1(s),\ J_2(s),\ \cdots,\ J_c(s)) \neq (J_1(t),\ J_2(t),\ \cdots,\ J_c(t)), \tag{13.1}$$

where $s, t \in \mathbf{Z_N},\ J_i(s), J_i(t) \in \mathbf{Z_{n_i}}$, and $s \neq t$. By using column vectors in $\mathbf{H}_1,\ \mathbf{H}_2,\ \cdots,\ \mathbf{H}_c$, we define the s-th column vector $\boldsymbol{h}_s$ of a parity-check matrix $\mathbf{H}$ over $R(q_1,\ q_2,\ \cdots,\ q_c)$ as

$$\boldsymbol{h}_s = \langle\langle \begin{matrix} | \\ h_{1,J_1(s)}, \\ | \end{matrix} \begin{matrix} | \\ h_{2,J_2(s)}, \\ | \end{matrix} \cdots, \begin{matrix} | \\ h_{c,J_c(s)} \\ | \end{matrix} \rangle\rangle,$$

where $0 \le s \le N - 1$. The parity-check matrix $\mathbf{H}$ over $R(q_1,\ q_2,\ \cdots,\ q_c)$, having check symbol length r, is given by the following:

$$\mathbf{H} = \left[\, \boldsymbol{h}_0 \quad \boldsymbol{h}_1 \quad \cdots \quad \boldsymbol{h}_{N-r-1} \,\middle|\, \boldsymbol{h}_{N-r} \quad \cdots \quad \boldsymbol{h}_{N-1} \,\right],$$

where $[\,\boldsymbol{h}_{N-r} \quad \cdots \quad \boldsymbol{h}_{N-1}\,]$ is the $r \times r$ identity matrix over $R(q_1,\ q_2,\ \cdots,\ q_c)$. The last r columns of $\mathbf{H}$ indicate the check part of $\mathbf{H}$.

TABLE 13.2 Example of Mapping J_i

s	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$J_1(s)$	0	0	0	1	1	1	2	2	2	2	3	3	3	3	1	0
$J_2(s)$	1	2	3	0	2	3	0	1	2	3	0	1	2	3	1	0

Source: *[KANE04a]. © 2004 IEEE.*

Example 13.3 [KANE04a]

Let $\mathbf{H}_1$ and $\mathbf{H}_2$ be identical 2×4 parity-check matrices of a single-symbol error correcting Hamming code over $GF(3)$ expressed by

$$\mathbf{H}_1 = \begin{bmatrix} | & | & | & | \\ h_{1,0} & h_{1,1} & h_{1,2} & h_{1,3} \\ | & | & | & | \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 \end{bmatrix},$$

$$\mathbf{H}_2 = \begin{bmatrix} | & | & | & | \\ h_{2,0} & h_{2,1} & h_{2,2} & h_{2,3} \\ | & | & | & | \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 \end{bmatrix}.$$

A column vector $\boldsymbol{h}_s$ over $R(3,3)$ is given by the following:

$$\boldsymbol{h}_s = \left\langle\left\langle \begin{matrix} | & | \\ h_{1,J_1(s)} & h_{2,J_2(s)} \\ | & | \end{matrix} \right\rangle\right\rangle,$$

where $0 \le s \le N-1 = 4 \times 4 - 1 = 15$, and J_1 and J_2 are mappings given in Table 13.2. The parity-check matrix $\mathbf{H}$ over $R(3,\ 3)$ having $r = 2$ check symbols is given by

$$\mathbf{H} = \left[\begin{array}{cccccccccccccc|cc} \langle 0,1\rangle & \langle 0,1\rangle & \langle 0,1\rangle & \langle 1,0\rangle & \langle 1,1\rangle & \langle 1,1\rangle & \langle 1,0\rangle & \langle 1,1\rangle & \langle 1,1\rangle & \langle 1,1\rangle & \langle 1,0\rangle & \langle 1,1\rangle & \langle 1,1\rangle & \langle 1,1\rangle & \langle 1,1\rangle & \langle 0,0\rangle \\ \langle 1,0\rangle & \langle 1,1\rangle & \langle 1,2\rangle & \langle 0,1\rangle & \langle 0,1\rangle & \langle 0,2\rangle & \langle 1,1\rangle & \langle 1,0\rangle & \langle 1,1\rangle & \langle 1,2\rangle & \langle 2,1\rangle & \langle 2,0\rangle & \langle 2,1\rangle & \langle 2,2\rangle & \langle 0,0\rangle & \langle 1,1\rangle \end{array}\right].$$

The last two columns indicate the check part.

Mapping Functions As mentioned previously, the M-ary single $\mathcal{E}$-asymmetric symbol error correcting codes should correct any single-symbol error indicated by an edge $(a_x \rightarrow a_y)$ of error directionality graph. In order to correct the error $(a_x \rightarrow a_y)$ by using the parity-check matrix $\mathbf{H}$ over $R(q_1,\ q_2,\ \cdots,\ q_c)$, the corresponding error value over $R(q_1,\ q_2,\ \cdots,\ q_c)$ should be a regular element, meaning f should satisfy $f(a_y) - f(a_x) \in R'(q_1,\ q_2,\ \cdots,\ q_c)$ for all $(a_x \rightarrow a_y) \in \mathcal{E}$. Further, since f is not an injective mapping, the original value a_x can be obtained from $f(a_x)$ only if f satisfies $f(a_x) \neq f(a_z)$ for all $(a_z \rightarrow a_y) \in \mathcal{E}$. Therefore the function f should satisfy two classes of conditions related to graph coloring problems of error directionality graph G, as illustrated in Figure 13.3

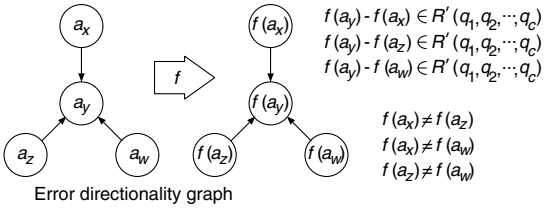


Figure 13.3 *Conditions for function f.* Source: *[KANE04a]. © 2004 IEEE.*

Based on the considerations above, the function f is defined as follows.

Definition 13.7 Let $G = (\mathbf{V}, \mathbf{E})$ be an error directionality graph based on asymmetric symbol error set $\mathcal{E}$. Function f is a mapping from $\mathbf{V} = \mathbf{A} = \{a_0,\ a_1, \cdots,\ a_{M-1}\}$ into $R(q_1,\ q_2,\ \cdots,\ q_c)$ satisfying the following conditions:

$$\begin{aligned}[(a_x \rightarrow a_y) \in \mathbf{E} \wedge f(a_x) &= \langle x_1,\ x_2,\ \cdots,\ x_c\rangle \\ \wedge f(a_y) &= \langle y_1,\ y_2,\ \cdots,\ y_c\rangle] \\ &\rightarrow [x_i \neq y_i,\ \text{where } 1 \leq i \leq c], \end{aligned} \tag{13.2}$$

$$\begin{aligned}&[(a_x \rightarrow a_y) \in \mathbf{E} \wedge (a_z \rightarrow a_y) \in \mathbf{E} \wedge a_x \neq a_z] \\ &\qquad \rightarrow [f(a_x) \neq f(a_z)], \end{aligned} \tag{13.3}$$

where $\langle x_1,\ x_2,\ \cdots,\ x_c\rangle$, $\langle y_1,\ y_2,\ \cdots,\ y_c\rangle \in R(q_1,\ q_2,\ \cdots,\ q_c)$, and $R(q_1,\ q_2, \cdots,\ q_c)$ satisfies

$$\prod_{i=1}^{c} q_i \leq M. \tag{13.4}$$

□

In order to design M-ary single $\mathcal{E}$-asymmetric symbol error correcting code, ring $R(q_1,\ q_2,\ \cdots,\ q_c)$ and function f that satisfy the conditions above must be determined. However, $R(q_1,\ q_2,\ \cdots,\ q_c)$ and f cannot be obtained systematically because the conditions shown by Eqs. (13.2) and (13.3) are related to graph coloring problems. Thus $R(q_1,\ q_2,\ \cdots,\ q_c)$ and f are determined by brute force computer search over a set of parameters $\{(q_1,\ q_2,\ \cdots,\ q_c)\,|\,q_i \geq \mathrm{CN}(G), \prod_{i=1}^{c} q_i \leq M\}$, where $\mathrm{CN}(G)$ is the *chromatic number* of G, meaning the number of colors of G, and q_i is prime or power of prime. If $R(q_1,\ q_2,\ \cdots,\ q_c)$ and f satisfying the conditions do not exist, then the code cannot be designed. In this case the conventional M-ary single symmetric symbol error correcting codes [NAMB01] should be used.

Definition 13.8 Function g is an arbitrary *surjective mapping* from A onto $R(q_1,\ q_2,\ \cdots,\ q_c)$. Inverse function g^{-1} is a mapping from $R(q_1,\ q_2,\ \cdots,\ q_c)$ into A that satisfies $g(g^{-1}(x)) = x$, where $x \in R(q_1,\ q_2,\ \cdots,\ q_c)$. □

The function g can be easily determined, e.g., $g(a_i) = \mathrm{W}_{i \bmod Q}$, where $Q = \prod_{i=1}^{c} q_i$ and $R(q_1,\ q_2,\ \cdots,\ q_c) = \{\mathrm{W}_0,\ \mathrm{W}_1,\ \cdots,\ \mathrm{W}_{Q-1}\}$.

Although both functions f and g map A into $R(q_1,\ q_2,\ \cdots,\ q_c)$, there exist differences in their necessary conditions. That is, f has some constraints related to the coloring problems of the error directionality graph G , whereas g does not have such constraints but has a condition of surjection.

Example 13.4 [KANE04a]

Let $\mathbf{A} = \{0,\ 1,\ \cdots,\ 9\}$ ba a set of 10-ary symbols. We derive the functions $f : \mathbf{A} \rightarrow R(3,\ 3)$, $g : \mathbf{A} \rightarrow R(3,\ 3)$, and $g^{-1} : R(3,\ 3) \rightarrow \mathbf{A}$ for the error directionality graph G_A shown in Figure 13.1. The functions f and g label each vertex of G_A with an

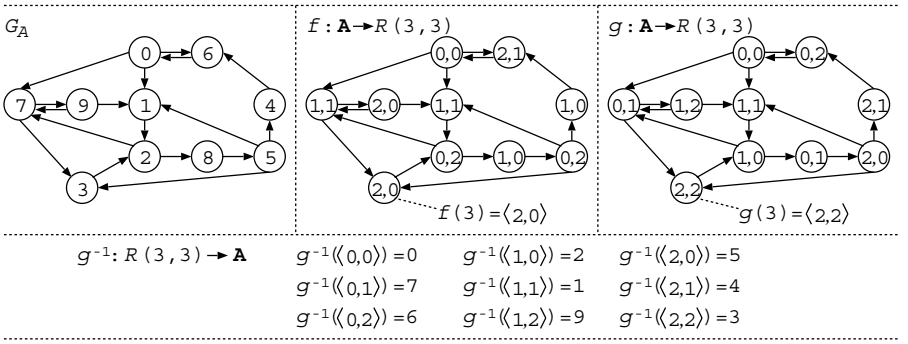


Figure 13.4 *Examples of functions f, g, and g^{-1} for G_A.* Source*: [KANE04a]. © 2004 IEEE.*

element in $R(3,\ 3)$. Figure 13.4 expresses the functions f and g by means of the graph G_A labeled with elements in $R(3,\ 3)$. The function f satisfying Eqs. (13.2) and (13.3) is obtained by a brute force search through all possible labeling patterns of G_A. The function g is obtained by labeling each vertex with an element in $R(3,\ 3)$ in such a way that every element in $R(3,\ 3)$ is used at least once. The function g^{-1} is determined as $g^{-1}(\langle x_1,\ x_2 \rangle) = a_i$, where $g(a_i) = \langle x_1,\ x_2 \rangle$. Note that $g^{-1}(\langle 0, 1 \rangle)$, which is determined as 7 in Figure 13.4, can also be determined as 8 because $g(7) = g(8) = \langle 0, 1 \rangle$.

Figure 13.5 shows an example of the functions $f : \mathbf{A} \to R(9)$, $g : \mathbf{A} \to R(9)$, and $g^{-1} : R(9) \to \mathbf{A}$ for the error directionality graph G_C. These functions are derived in the same way as the case of G_A. In this case the functions f and g are identical.

Code Design

Theorem 13.3 *Let* $\boldsymbol{H}$ *be an* $r \times N$ *parity-check matrix over* $R(q_1,\ q_2, \cdots,\ q_c)$*, and also let* f *and* g *be mapping functions defined in Definitions 13.7 and 13.8, respectively. Code*

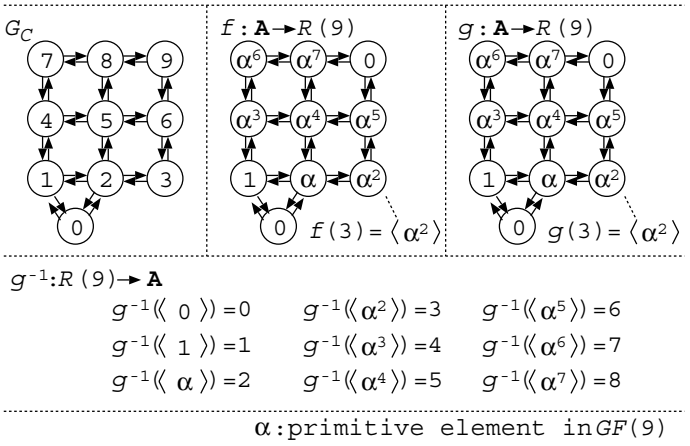


Figure 13.5 *Examples of functions f, g, and g^{-1} for G_C.* Source*: [KANE04a]. © 2004 IEEE.*

C *defined by the following is a systematic M-ary single $\mathcal{E}$-asymmetric symbol error correcting code:*

$$\boldsymbol{C} = \left\{ \boldsymbol{u} = (d_0 \ \cdots \ d_{k-1} \ p_0 \ \cdots \ p_{r-1}) \ \middle| \ \begin{array}{r} \boldsymbol{H} \cdot (f(d_0) \ \cdots \ f(d_{k-1}) \ g(p_0) \ \cdots \ g(p_{r-1}))^T \\ = \langle (\overbrace{\boldsymbol{0}, \cdots, \boldsymbol{0}}^{c}) \rangle \end{array} \right\},$$

where $d_i \in \boldsymbol{A},\ 0 \le i \le k-1$, is an information symbol, $p_j \in \boldsymbol{A},\ 0 \le j \le r-1$, is a check symbol, $\boldsymbol{0}$ is a zero column vector with length r, $k = N - r$, and

$$N = \prod_{i=1}^{c} \frac{q_i^r - 1}{q_i - 1}. \tag{13.5}$$

Proof Let $\mathbf{u} = (d_0 \ \cdots \ d_{k-1} \ p_0 \ \cdots \ p_{r-1})$ be the original codeword, and let $\mathbf{u}' = (d_0' \ \cdots \ d_{k-1}' \ p_0' \ \cdots \ p_{r-1}')$ be the corresponding received word. The received word $\mathbf{u}'$ having single $\mathcal{E}$-asymmetric symbol errors that occur in the following cases can be corrected.

Case 1. *Single $\mathcal{E}$-asymmetric symbol errors in the information part*. Assume that there is an error in the l-th information symbol, meaning $d_i' = d_i \ (0 \le i \le k-1, i \ne l)$ and $(d_l \to d_l') \in \mathbf{E}$, where $0 \le l \le k-1$. Since $(f(d_l') - f(d_l)) \in R'(q_1, q_2, \cdots, q_c)$ from Eq. (13.2), and two column vectors in $\mathbf{H}_i$ are linearly independent, the syndrome defined by

$$\begin{aligned} S &= \langle\langle s_1, \ s_2, \ \cdots, \ s_c \rangle\rangle \\ &= \mathbf{H} \cdot (f(d_0') \ \cdots \ f(d_{k-1}') \ g(p_0') \ \cdots \ g(p_{r-1}'))^T \end{aligned}$$

can be factorized as follows without any ambiguity:

$$S = (f(d_l') - f(d_l))\boldsymbol{h}_l.$$

Thus the error location l can be determined from the syndrome S. Since f is not a one-to-one mapping, an original M-ary symbol d_l cannot always be uniquely determined only by $f(d_l)$. The original symbol d_l, however, can be obtained from a pair of values $(f(d_l), d_l')$ because Eq. (13.5) indicates that if

$$[a_x \ne a_y] \wedge [(a_x \to d_l') \in \mathbf{E}] \wedge [(a_y \to d_l') \in \mathbf{E}],$$

then $f(a_x) \ne f(a_y)$. In other words, an M-ary symbol a_x that satisfies $f(a_x) = f(d_l)$ and $(a_x \to d_l') \in E$ gives the original symbol d_l. Hence the error in d_l' can be corrected.

Case 2. *Single $\mathcal{E}$-asymmetric symbol errors in the check part*. Assume that there is an error in the l'-th check symbol, meaning $p_i' = p_i \ (0 \le i \le r-1, i \ne l')$ and $(p_{l'} \to p_{l'}') \in E$, where $0 \le l' \le r-1$.

- If $(g(p'_{l'}) - g(p_{l'})) \in R'(q_1, q_2, \cdots, q_c)$, then the error location $l = l' + k \geq k$, which indicates that an error exists in the check part, is obtained from the syndrome in the same way as the case 1. Then the correct check symbols can be recovered by re-encoding the received error-free information part.
- If $(g(p'_{l'}) - g(p_{l'})) \notin R'(q_1, q_2, \cdots, q_c)$, then there exists i that satisfies $s_i = o$, where $1 \leq i \leq c$. This means that the error is discriminated from the errors in the information part because the syndromes caused by errors in the information part do not satisfy $s_i = o$ for any i. Thus correct check symbols can be recovered by re-encoding the received error-free information part.

Clearly, $\boldsymbol{C}$ can correct single $\mathcal{E}$-asymmetric symbol errors. Further it is obvious from the definition of $\mathbf{H}$ that the code is systematic and the code length N is given by Eq. (13.5). Q.E.D.

3. Decoding Procedure

Let the received word be denoted as

$$\mathbf{u}' = (d'_0\ d'_1\ \cdots\ d'_{k-1}\ p'_0\ p'_1\ \cdots\ p'_{r-1})$$

and also the output of the decoder be denoted as $\hat{\mathbf{u}} = (\hat{d}_0\ \hat{d}_1\ \cdots\ \hat{d}_{k-1}\ \hat{p}_0\ \hat{p}_1\ \cdots\ \hat{p}_{r-1})$. The word $\mathbf{u}'$ is transformed into elements in $R(q_1, q_2, \cdots, q_c)$ by the functions f and g:

$$\begin{aligned} U' &= (D'_0\ D'_1\ \cdots\ D'_{k-1}\ P'_0\ P'_1\ \cdots\ P'_{r-1}) \\ &= (f(d'_0)\ f(d'_1)\ \cdots\ f(d'_{k-1})\ g(p'_0)\ g(p'_1)\ \cdots\ g(p'_{r-1})). \end{aligned}$$

The syndrome of U' is calculated as follows:

$$S = \langle\langle s_1,\ s_2,\ \cdots,\ s_c\rangle\rangle = \mathbf{H} \cdot U'^T.$$

If there exists i that satisfies $s_i = \boldsymbol{0}$, then the information part of $\mathbf{u}'$ has no error. In this case the output $\hat{\mathbf{u}}$ is obtained by re-encoding the error-free information part, meaning $(d'_0\ d'_1\ \cdots\ d'_{k-1})$. If $s_i \neq \boldsymbol{0}$ for all i's, the syndrome S is factorized as $S = \mathbf{e}\langle\langle v_1,\ v_2,\ \cdots,\ v_c\rangle\rangle$, where $\mathbf{e} \in R'(q_1,\ q_2,\ \cdots,\ q_c)$, v_i is a column vector in $\mathbf{H}_i$, and $1 \leq i \leq c$. The error location is given by l, which satisfies the following equation:

$$\langle\langle h_{1,J_1(l)},\ h_{2,J_2(l)},\ \cdots,\ h_{c,J_c(l)}\rangle\rangle = \langle\langle v_1,\ v_2,\ \cdots,\ v_c\rangle\rangle.$$

If this is not satisfied for any l, then uncorrectable errors exist in $\mathbf{u}'$. If $l \geq k$, then an error exists in the check part of $\mathbf{u}'$. In this case, $\hat{\mathbf{u}}$ is obtained by re-encoding $(d'_0\ d'_1\ \cdots\ d'_{k-1})$. If $l < k$, $\hat{\mathbf{u}}$ is given by the following:

$$\hat{\mathbf{u}} = (d'_0\ d'_1\ \cdots\ d'_{l-1}\ \ \hat{d}_l\ d'_{l+1}\ \cdots\ d'_{k-1}\ p'_0\ p'_1\ \cdots\ p'_{r-1}),$$

where $f(\hat{d}_l) = D'_l - \mathbf{e}$ and $(\hat{d}_l \rightarrow d'_l) \in \mathbf{E}$.

Figure 13.6 illustrates the procedure above.

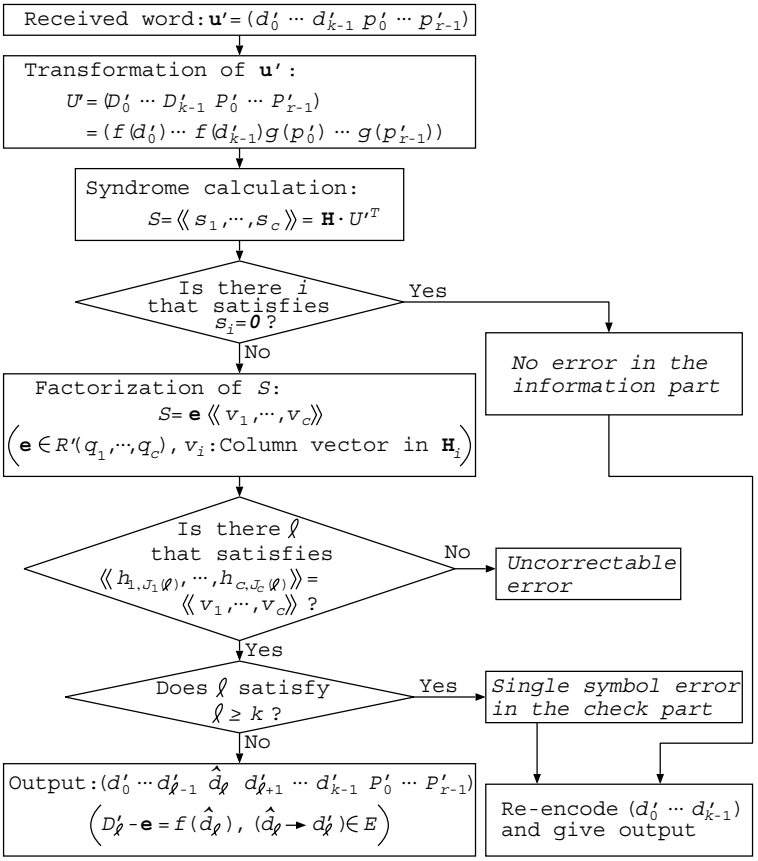


Figure 13.6 *Decoding procedure.* Source: *[KANE04a]. © 2004 IEEE.*

Example 13.5 [KANE04a]

Let $\mathbf{H}'$ be the parity-check matrix over $R(3, 3)$ defined by the following:

$$\begin{aligned}\mathbf{H}' &= [\boldsymbol{h}_0\ \ \boldsymbol{h}_1\ \ \cdots\ \ \boldsymbol{h}_8] \\ &= \left[\begin{array}{ccccccccc|cc} \langle 0,1\rangle & \langle 0,1\rangle & \langle 0,1\rangle & \langle 1,0\rangle & \langle 1,1\rangle & \langle 1,1\rangle & \langle 1,0\rangle & \langle 1,1\rangle & \langle 0,0\rangle \\ \langle 1,0\rangle & \langle 1,1\rangle & \langle 1,2\rangle & \langle 0,1\rangle & \langle 0,1\rangle & \langle 0,2\rangle & \langle 1,1\rangle & \langle 0,0\rangle & \langle 1,1\rangle \end{array}\right].\end{aligned}$$

This code is obtained by deleting the last seven columns in the information part of $\mathbf{H}$ of Example 13.3. Let the codeword of 10-ary single $\mathcal{E}_A$-asymmetric symbol error correcting code be $\mathbf{u} = (d_0\ d_1\ d_2\ d_3\ d_4\ d_5\ d_6\ p_0\ p_1) = (3\ 8\ 9\ 0\ 1\ 2\ 5\ 5\ 9)$ over $\mathbf{A} = \{0, 1, \cdots, 9\}$, which satisfies

$$\mathbf{H}' \cdot (f(3)\, f(8)\, f(9)\, f(0)\, f(1)\, f(2)\, f(5)\ g(5)\ g(9))^T = \langle\langle \boldsymbol{0}, \boldsymbol{0} \rangle\rangle,$$

where $f : \mathbf{A} \to R(3, 3)$ and $g : \mathbf{A} \to R(3, 3)$ are the functions given in Figure 13.4.

Assume that an error exists in d_2; that is, $d_2 = 9$ is changed to $d_2' = 1$:

$$\mathbf{u}' = (d_0'\ d_1'\ d_2'\ d_3'\ d_4'\ d_5'\ d_6'\ p_0'\ p_1') = (3\ 8\ \underline{\mathbf{1}}\ 0\ 1\ 2\ 5\ 5\ 9).$$

Decoding is performed by the following procedure:

The received word $\mathbf{u}'$ is transformed into elements in $R(3,3)$ by the functions f and g,

$$\begin{aligned} U' &= (D_0'\ D_1'\ D_2'\ D_3'\ D_4'\ D_5'\ D_6'\ P_0'\ P_1') \\ &= (f(3)\ f(8)\ f(1)\ f(0)\ f(1)\ f(2)\ f(5)\ g(5)\ g(9)) \\ &= (\langle 2,0\rangle\ \langle 1,0\rangle\ \langle 1,1\rangle\ \langle 0,0\rangle\ \langle 1,1\rangle\ \langle 0,2\rangle\ \langle 0,2\rangle\ \langle 2,0\rangle\ \langle 1,2\rangle). \end{aligned}$$

The syndrome S is calculated as

$$\begin{aligned} S &= \mathbf{H}' \cdot (\langle 2,0\rangle\ \langle 1,0\rangle\ \langle 1,1\rangle\ \langle 0,0\rangle\ \langle 1,1\rangle\ \langle 0,2\rangle\ \langle 0,2\rangle\ \langle 2,0\rangle\ \langle 1,2\rangle)^T \\ &= \begin{pmatrix} \langle 0,1\rangle \\ \langle 2,2\rangle \end{pmatrix}. \end{aligned}$$

Then this syndrome is factorized as

$$S = \langle\langle 2\begin{pmatrix} 0 \\ 1 \end{pmatrix}, 1\begin{pmatrix} 1 \\ 2 \end{pmatrix}\rangle\rangle = \langle 2,1\rangle\langle\langle \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \end{pmatrix}\rangle\rangle = \mathbf{e}\boldsymbol{h}_2,$$

where $\mathbf{e} = \langle 2,1\rangle$. Therefore the error location $l = 2$ is obtained. Since $D_2' - \mathbf{e} = \langle 1,1\rangle - \langle 2,1\rangle = \langle 2,0\rangle = f(9)$ and $(9 \to d_2') = (9 \to 1) \in \mathbf{E}$, the symbol $\hat{d}_2$ is determined as 9.

The corrected output is $\hat{\mathbf{u}} = (3\ 8\ \underline{\mathbf{9}}\ 0\ 1\ 2\ 5\ 5\ 9)$.

Assume that the check symbol $p_0 = 5$ in $\mathbf{u}$ is changed to $p_0' = 4$. Then the syndrome S is calculated as

$$S = \begin{pmatrix} \langle 0,1\rangle \\ \langle 0,0\rangle \end{pmatrix}.$$

In this case the error is discriminated from errors in the information part because $s_1 = (0,0)^T$. Hence the output $\hat{\mathbf{u}}$ is obtained by re-encoding $(3\ 8\ 9\ 0\ 1\ 2\ 5)$.

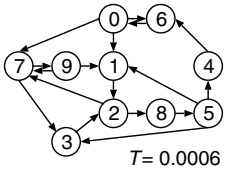
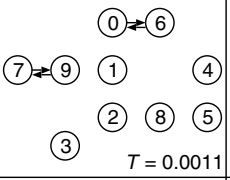
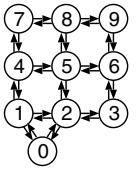
4. Evaluation

From Eq. (13.5) the information-symbol length of the code over $R(q_1, q_2, \cdots, q_c)$ is given by

$$k = N - r = \prod_{i=1}^{c} \frac{q_i^r - 1}{q_i - 1} - r,$$

where r is the check-symbol length. This means that the information-symbol length k depends on $R(q_1, q_2, \cdots, q_c)$ and r. In order to obtain $R(q_1, q_2, \cdots, q_c)$, which gives the greatest information-symbol length k for given error directionality graph G, brute force computer search is performed as follows: (1) Enumerate the rings that satisfy Eq. (13.4),

TABLE 13.3 Check-Symbol Lengths and Information Symbol Lengths of Codes

	10-Ary codes for handwritten numeral recognition systems				10-Ary codes for numeric keypads		Existing 10-ary single symmetric symbol error correcting codes [NAMB01]
Error directionality graph	 $T = 0.0006$		 $T = 0.0011$				
Ring	$R(3,3)$		$R(2,2,2)$		$R(9)$		—
r	k	Bound	k	Bound	k	Bound	k
2	14	33	25	99	8	24	—
3	166	333	340	999	88	249	46
4	1,596	3,333	3,371	9,999	816	2,499	498
5	14,636	33,333	29,786	99,999	7,376	24,999	4,998

Source : *[KANE04a].* © *2004 IEEE.*

Note: r: check-symbol length, k: information-symbol length, bound: upper bound on the information-symbol length given in Theorem 13.1.

(2) calculate the information-symbol length k for each ring, and (3) find a ring $R(q_1, q_2, \cdots, q_c)$ that gives the maximal value of k and for which the function f satisfying Eqs. (13.2) and (13.3) exists.

Table 13.3 shows the information-symbol length k of the codes for two types of data entry devices: the handwritten numeral recognition system and the numeric keypad. Note that the upper bound on the information-symbol lengths of the M-ary single $\mathcal{E}$-asymmetric symbol error correcting codes are obtained from Theorem 13.1. For handwritten numeral recognition systems, two classes of codes are evaluated: one with threshold error probability $T = 0.0006$ and the other with $T = 0.0011$. The last column of the table shows the information-symbol lengths of the existing excellent M-ary single symmetric symbol error correcting codes [NAMB01]. The indicated asymmetric codes have greater information-symbol length than the existing symmetric codes. Also the codes require two check symbols, whereas the existing symmetric codes require at least three check symbols.

13.2.2 Nonsystematic Codes

From a practical standpoint, M-ary asymmetric error control codes for data entry systems are not necessarily systematic because in many cases, there is no significance to distinguishing between information symbols and check symbols. Furthermore M-ary asymmetric error control codes can be flexibly designed by using nonsystematic codes. In this subsection we look at a new class of nonsystematic M-ary asymmetric error correcting codes [KANE04b] that are based on a *multilevel coding method* and a *set partitioning algorithm* originally developed for M-ary communication channels [IMAI77].

1. Preliminaries

This subsection presents some mathematical definitions used for code design.

Rooted Tree $T(q_1, q_2, \ldots, q_c)$ Let the *level* of a node v in a rooted tree be the length of the path from the node v to the root node, where each edge has a length of 1, and let the level of the root node be 0. A *rooted tree* $T(q_1, q_2, \ldots, q_c)$ is defined as follows:

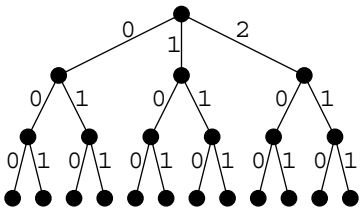


Figure 13.7 *Tree T(3, 2, 2).* Source: *[KANE04b]. © 2004 IEEE.*

- Each node in level i has q_{i+1} child nodes, for $0 \leq i \leq c-1$.
- Each node in level c has no child node (i.e., every node in level c is a leaf node).
- All edges connecting a node in level i with its q_{i+1} child nodes are labeled, each with a distinct element $x \in GF(q_{i+1})$ for $0 \leq i \leq c-1$.

Example 13.6 [KANE04b]

Figure 13.7 gives an example of a tree $T(3, 2, 2)$, where $GF(2) = \{0, 1\}$ and $GF(3) = \{0, 1, 2\}$.

Confusion Matrix P Let $A = \{a_0, a_1, \ldots, a_{M-1}\}$ be a set of M-ary symbols. Nonsystematic M-ary asymmetric error correcting codes are constructed based on the following *confusion matrix* $\mathbf{P}$:

$$\mathbf{P} = \begin{bmatrix} p(a_0|a_0) & p(a_1|a_0) & \ldots & p(a_{M-1}|a_0) \\ p(a_0|a_1) & p(a_1|a_1) & \ldots & p(a_{M-1}|a_1) \\ \vdots & \vdots & & \vdots \\ p(a_0|a_{M-1}) & p(a_1|a_{M-1}) & \ldots & p(a_{M-1}|a_{M-1}) \end{bmatrix},$$

where $p(a_j|a_i)$ are the transition probabilities for a transmitted symbol a_i being received as a symbol a_j, which are given a priori.

Recall the example in Table 13.1 of the confusion matrix $\mathbf{P}$ for a class of handwritten postal code recognition systems, where $\mathbf{A} = \{0, 1, \ldots, 9\}$ and $a_i = i$ for $0 \leq i \leq 9$. The matrix $\mathbf{P}$ is derived from statistical data given in [NOUM93].

2. Code Design

The code design method of the nonsystematic M-ary asymmetric error correcting codes $\mathbf{C}$ is based on a *multilevel coding method* [IMAI77]. That is, $\mathbf{C}$ is determined by a function F that maps a set $\mathbf{A}$ of M-ary symbols to a ring $R(q_1, q_2, \cdots, q_c)$ mentioned in Definition 13.5 and Theorem 13.2 of Subsection 13.2.1, and by a series of block codes $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_c$, where $\mathbf{C}_i$ is defined over $GF(q_i)$ for $1 \leq i \leq c$. Figure 13.8 shows the relation between a codeword $U = (u_0\ u_1\ \ldots\ u_{n-1})$ of the M-ary code $\mathbf{C}$ and a codeword $W_i = (w_{i,0}\ w_{i,1}\ \ldots\ w_{i,n-1})$ of the block code $\mathbf{C}_i$. Note in this figure that $U = (u_0\ u_1\ \ldots\ u_{n-1})$ is a codeword of $\mathbf{C}$ only if $W_i = (w_{i,0}\ w_{i,1}\ \ldots\ w_{i,n-1})$ is a codeword of $\mathbf{C}_i$ for all $i \in \{1, 2, \ldots, c\}$, where $F(u_j) = \langle w_{1,j}, w_{2,j}, \ldots, w_{c,j} \rangle$ for $0 \leq j \leq n-1$.

The following shows a construction of the function F based on a newly defined set partitioning algorithm, and then defines a new class of nonsystematic M-ary asymmetric error correcting codes using the function F.

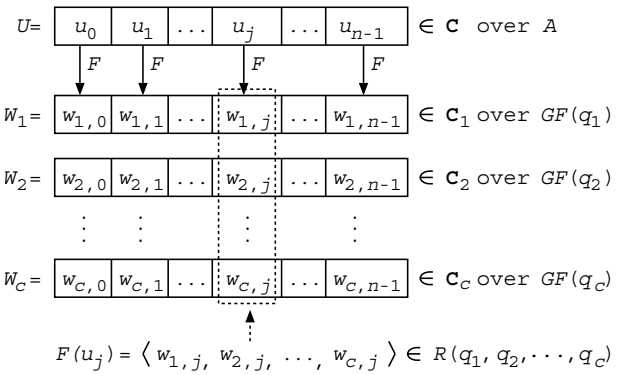


Figure 13.8 *Relation between codeword U of nonsystematic M-ary asymmetric error correcting code C and codeword W_i of block code C_i over $GF(q_i)$ for $1 \leq i \leq c$.* Source*: [KANE04b]. © 2004 IEEE.*

(1) Set-Partitioning Algorithm

In order to define the function F that maps a set $\mathbf{A}$ of M-ary symbols to a ring $R(q_1,\ q_2,\ \ldots,\ q_c)$, the set $\mathbf{A}$ is iteratively partitioned into q_i subsets for each i, $1 \leq i \leq c$. This is achieved by using the new algorithm indicated here, which is based on the *set-partitioning algorithm* described in [IMAI77]. Similar to the conventional set-partitioning algorithm that divides a set of M-ary symbols into subsets based on Euclidean distance in the signal constellation, the new algorithm defined here divides the set $\mathbf{A}$ into subsets based on the following probability:

$$p(a_s, a_t) = \max_{a_k \in \mathbf{A}} \{\min\{p(a_k|a_s), p(a_k|a_t)\}\},$$

where $a_s, a_t \in \mathbf{A}$ and $a_s \neq a_t$. Figure 13.9 shows examples of transition probabilities corresponding to small and large values for $p(a_s, a_t)$, respectively, where the width of each arrow indicates the transition probability, a_s and a_t are transmitted symbols, and a_k is a received symbol. In case 13.9(a), the transmitted symbol, either $a_s = 2$ or $a_t = 4$, can be reliably estimated from the received symbol a_k because either $p(a_k|a_s)$ or $p(a_k|a_t)$ is

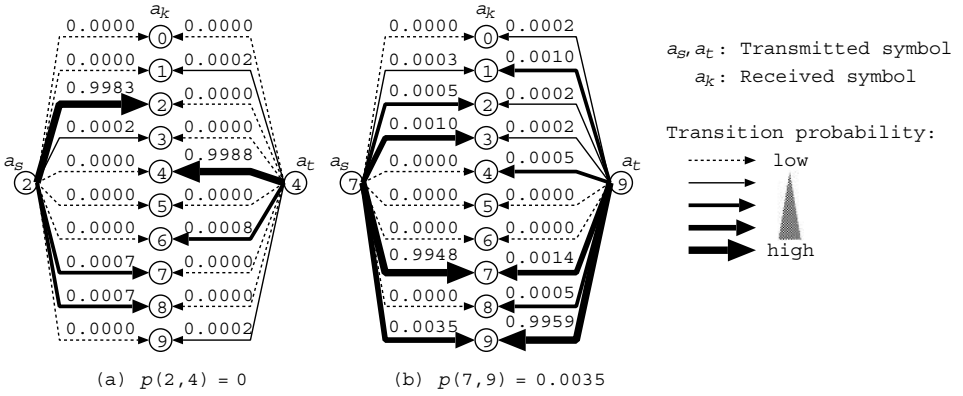


Figure 13.9 *Examples of transition probabilities: (a) For a small value of $p(a_s, a_t)$ and (b) for a large value of $p(a_s, a_t)$.* Source*: [KANE04b]. © 2004 IEEE.*

sufficiently small for all $a_k \in \mathbf{A}$. In case 13.9(b), however, the transmitted symbol, either $a_s = 7$ or $a_t = 9$, cannot always be reliably estimated from the received symbol a_k because there exists $a_k \in \mathbf{A}$ such that both $p(a_k|a_s)$ and $p(a_k|a_t)$ have a large value, such as $a_k = 7$ or 9. This implies that a small value for $p(a_s, a_t)$ should correspond to a large Euclidean distance between the two transmitted symbols because a_s and a_t are easily distinguished from each other regardless of the received symbol; conversely, a large value for $p(a_s, a_t)$ should correspond to a small Euclidean distance between the two symbols.

Based on the probability $p(a_s, a_t)$ as defined above, a new set-partitioning algorithm is derived for a given confusion matrix $\mathbf{P}$ and a tree $T(q_1, q_2, \ldots, q_c)$, where

$$\prod_{i=1}^{c} q_i \geq M.$$

In the following set-partitioning algorithm, a subset $\mathbf{A}(v)$ of $\mathbf{A}$ is assigned to each node v in $T(q_1, q_2, \ldots, q_c)$.

New Set-Partitioning Algorithm

Step 1. *Assign set* $\boldsymbol{A}$ *to the root node* v_R *in the rooted tree* $T(q_1, q_2, \ldots, q_c)$*, that is,* $\boldsymbol{A}(v_R) := \boldsymbol{A}$.

Step 2. *Set* $i := 0$.

Step 3. *For each node* v *in level* i*, divide* $\boldsymbol{A}(v)$ *into* q_{i+1} *disjoint subsets* $\boldsymbol{A}_1(v)$, $\boldsymbol{A}_2(v)$, $\ldots$, $\boldsymbol{A}_{q_{i+1}}(v)$*, in such a way that the following value is minimized:*

$$\max_{1 \leq j \leq q_{i+1}} \left\{ \max_{\substack{a_s, a_t \in \boldsymbol{A}_j(v) \\ a_s \neq a_t}} \{p(a_s, a_t)\} \right\},$$

where the cardinality of $\boldsymbol{A}_j(v)$ *satisfies the condition*

$$|\boldsymbol{A}_j(v)| \leq \frac{\prod_{k=1}^{c} q_k}{\prod_{k=1}^{i+1} q_k} \qquad \text{for all } 1 \leq j \leq q_{i+1}.$$

Each of the obtained subsets $\boldsymbol{A}_1(v), \boldsymbol{A}_2(v), \ldots, \boldsymbol{A}_{q_{i+1}}(v)$ *is then assigned to a distinct offspring node* v'_j *of* v*, namely* $\boldsymbol{A}(v'_j) := \boldsymbol{A}_j(v)$ *for* $1 \leq j \leq q_{i+1}$.

Step 4. *If* $i < c - 1$*, then set* $i := i + 1$ *and go to step 3; otherwise, terminate the algorithm.*

Figure 13.10 shows an overview of the algorithm. The tree obtained by this algorithm, with a subset of $\mathbf{A}$ assigned to each node, will be denoted by $\mathfrak{T}(q_1, q_2, \ldots, q_c)$. As the algorithm is performed, each M-ary symbol is assigned to a distinct leaf node in $\mathfrak{T}(q_1, q_2, \ldots, q_c)$.

Definition 13.9 For a given $\mathfrak{T}(q_1, q_2, \ldots, q_c)$, the function $F : \mathbf{A} \rightarrow R(q_1, q_2, \ldots, q_c)$ is defined as follows:

$$F(a_j) = \langle f_1(a_j), f_2(a_j), \ldots, f_c(a_j) \rangle = \langle x_1, x_2, \ldots, x_c \rangle,$$

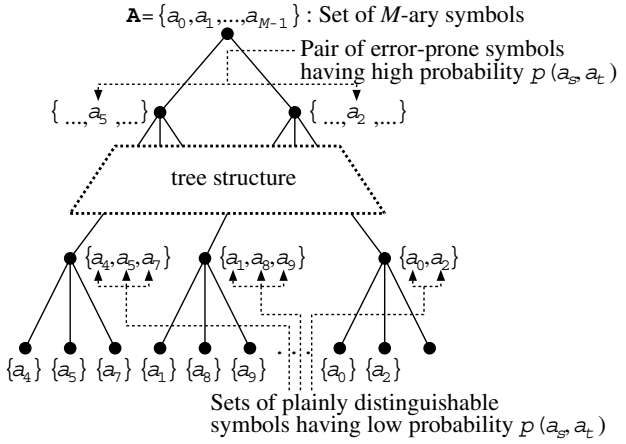


Figure 13.10 *Overview of new set-partitioning algorithm.* Source: *[KANE04b]. © 2004 IEEE.*

where f_i is a function that maps $\mathbf{A}$ to $GF(q_i)$ for $1 \leq i \leq c$ and $\langle x_1, x_2, \ldots, x_c \rangle \in R(q_1, q_2, \ldots, q_c)$ represents the sequence of edge labels for a path from the root node v_R to the leaf node v for which $\mathbf{A}(v) = \{a_j\}$. □

Example 13.7 [KANE04b]

Figure 13.11 shows a $\mathcal{T}(3, 2, 2)$ tree for the confusion matrix $\mathbf{P}$ given in Table 13.1. In this case function $F : \mathbf{A} \rightarrow R(3, 2, 2)$ is determined as follows:

$$
\begin{aligned}
&F(0) = \langle 1, 0, 0 \rangle, \quad F(1) = \langle 2, 0, 0 \rangle, \quad F(2) = \langle 1, 1, 1 \rangle,\\
&F(3) = \langle 0, 0, 0 \rangle, \quad F(4) = \langle 2, 0, 1 \rangle, \quad F(5) = \langle 2, 1, 0 \rangle,\\
&F(6) = \langle 0, 0, 1 \rangle, \quad F(7) = \langle 2, 1, 1 \rangle, \quad F(8) = \langle 0, 1, 0 \rangle,\\
&F(9) = \langle 0, 1, 1 \rangle.
\end{aligned}
$$

(2) Code Design

Definition 13.10 Let $\mathcal{T}(q_1, q_2, \ldots, q_c)$ be a tree obtained by the new set-partitioning algorithm, and let $\mathbf{C}_i$ be a block code over $GF(q_i)$ having length n and minimum

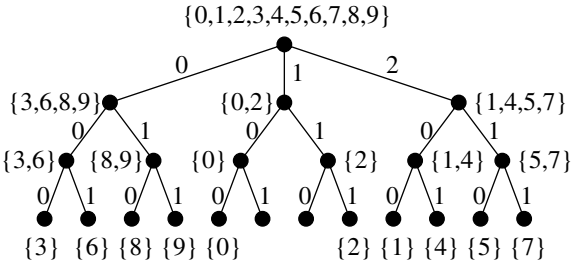


Figure 13.11 *Example of a $\mathcal{T}$ (3, 2, 2) tree.* Source: *[KANE04b]. © 2004 IEEE.*

Hamming distance d_i, where $1 \leq i \leq c$ and $d_1 \geq d_2 \geq \cdots \geq d_c$. A nonsystematic M-ary code $\mathbf{C}$ is then defined as follows:

$$\mathbf{C} = \{(u_0 \ \ldots \ u_{n-1}) | (f_i(u_0) \ \ldots \ f_i(u_{n-1})) \in \mathbf{C}_i \quad \textit{for all } i \in \{1, \ldots, c\}, u_j \in \mathbf{A} \text{ and } \quad \textit{for all } j \in \{0, \ldots, n-1\}\},$$

where $\langle f_1(u_j), f_2(u_j), \ldots, f_c(u_j)\rangle = F(u_j)$ is determined by $\mathfrak{T}(q_1, q_2, \ldots, q_c)$. □

By the Hamming distance property of codewords in $\mathbf{C}$, the following theorem shows that $\mathbf{C}$ has the desired asymmetric error correction capability.

Theorem 13.4 *Let $U^A = (u_0^A \ u_1^A \ \ldots \ u_{n-1}^A)$ and $U^B = (u_0^B \ u_1^B \ \ldots \ u_{n-1}^B)$ be codewords of $\mathbf{C}$, and let I_j be the minimum value that satisfies $f_{I_j}(u_j^A) \neq f_{I_j}(u_j^B)$, where $j \in \{0, 1, \ldots, n-1\}$. Then the Hamming distance between U^A and U^B satisfies the following inequality:*

$$d(U^A, U^B) \geq d_I,$$

where $I = \min_{0 \leq j \leq n-1}\{I_j\}$ and d_I is the minimum Hamming distance of $\mathbf{C}_I$.

Proof By the definition of I, there exists $j \in \{0, 1, \ldots, n-1\}$ that satisfies $f_I(u_j^A) \neq f_I(u_j^B)$. Then the Hamming distance between

$$(f_I(u_0^A) \ f_I(u_1^A) \ \ldots \ f_I(u_{n-1}^A))$$

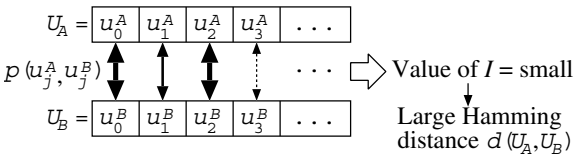
and

$$(f_I(u_0^B) \ f_I(u_1^B) \ \ldots \ f_I(u_{n-1}^B))$$

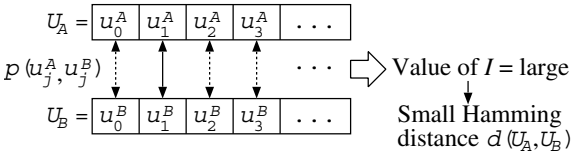
is greater than or equal to d_I. Therefore $d(U^A, U^B) \geq d_I$ holds. Q.E.D.

Figure 13.12 illustrates the Hamming distance between two distinct codewords of the code. On the one hand, as shown in Figure 13.12(a), if the above two codewords $U^A = (u_0^A \ u_1^A \ \ldots \ u_{n-1}^A)$ and $U^B = (u_0^B \ u_1^B \ \ldots \ u_{n-1}^B)$ have at least one pair of error-prone symbols (u_j^A, u_j^B) whose value of $p(u_j^A, u_j^B)$ is large, where $u_j^A \neq u_j^B$, then there exists a small I_j that satisfies $f_{I_j}(u_j^A) \neq f_{I_j}(u_j^B)$ because the new set-partitioning algorithm preferentially divides such error-prone symbols u_j^A and u_j^B into distinct nodes in $\mathfrak{T}(q_1, q_2, \ldots, q_c)$, as shown in Figure 13.10. Consequently $d(U^A, U^B)$ is guaranteed to be relatively large in this case because $d_1 \geq d_2 \geq \cdots \geq d_c$. On the other hand, as shown in Figure 13.12(b), if U_A and U_B do not have any pairs of error-prone symbols; that is, $p(u_j^A, u_j^B)$ is small for all $j \in \{0, 1, \ldots, n-1\}$, then $I = \min_{0 \leq j \leq n-1}\{I_j\}$ may have a large value. In this case $d(U_A, U_B)$ is small compared to the former case in Figure 13.12(a). Therefore the code $\mathbf{C}$ has a stronger error correction capability for pairs of error-prone codewords; that is, $\mathbf{C}$ is an asymmetric error correcting code.

In practice, the set $\mathbf{C}$ of codewords is generated by computer search. The exact number of codewords in $\mathbf{C}$ cannot be systematically determined because of the enumerative generation of $\mathbf{C}$. However, the number of codewords in $\mathbf{C}$ can be approximated by the following theorem.



(a) U_A and U_B: having at least one pair of error-prone symbols



(b) U_A and U_B: not having any pairs of error-prone symbols

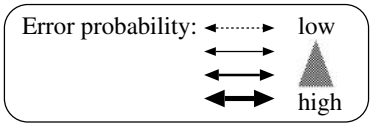


Figure 13.12 *Hamming distance between two distinct codewords U_A and U_B.* Source: *[KANE04b]. © 2004 IEEE.*

Theorem 13.5 *If every symbol in $GF(q_i)$ appears with the same probability $1/q_i$ in all codewords of $\mathbf{C}_i$ for $i \in \{1, 2, \ldots, c\}$, then the number of codewords in $\mathbf{C}$ defined by $\mathcal{T}(q_1, q_2, \ldots, q_c)$ is approximated as follows:*

$$|\boldsymbol{C}| \simeq \prod_{i=1}^{c} |\boldsymbol{C}_i| \times \left(\frac{M}{\prod_{i=1}^{c} q_i} \right)^n.$$

Proof Let $\mathbf{X}$ be a set of vectors over $R(q_1, q_2, \ldots, q_c)$ defined as

$$\mathbf{X} = \{(\mathbf{x}_0 \; \mathbf{x}_1 \; \ldots \; \mathbf{x}_{n-1}) \,|\, \mathbf{x}_j = \langle x_{1,j}, x_{2,j}, \ldots, x_{c,j} \rangle \in R(q_1, q_2, \ldots, q_c), \\ (x_{i,0} \; x_{i,1} \; \ldots \; x_{i,n-1}) \in \mathbf{C}_i, \; 1 \le i \le c, \; 0 \le j \le n-1\}.$$

By using the set $\mathbf{X}$, we can redefine the code $\mathbf{C}$ as

$$\mathbf{C} = \{(F^{-1}(\mathbf{x}_0) \; F^{-1}(\mathbf{x}_1) \; \ldots \; F^{-1}(\mathbf{x}_{n-1})) \,|\, (\mathbf{x}_0 \; \mathbf{x}_1 \; \ldots \; \mathbf{x}_{n-1}) \in \mathbf{X}, \\ \mathbf{x}_j \in \Phi(F), \quad \forall j \in \{0, 1, \ldots, n-1\}\},$$

where F^{-1} is the inverse function of F, and

$$\Phi(F) = \{\mathbf{x} \,|\, \mathbf{x} = F(u), u \in A\}$$

TABLE 13.4 Number of Codewords $|\mathbf{C}|$, Code Rate, and Decoded SER for $n = 7$

Code	$\mathcal{T}$	$\mathbf{C}_1$	d_1	$\mathbf{C}_2$	d_2	Approximation of $\lvert\mathbf{C}\rvert$	$\lvert\mathbf{C}\rvert$	Code rate	Decoded SER
I	$\mathcal{T}(11)$	RS	5	—	—	683	685	0.405	$\leq 10^{-9}$
II	$\mathcal{T}(11)$	RS	4	—	—	7,513	7,513	0.554	2.1×10^{-6}
III	$\mathcal{T}(11)$	HM	3	—	—	82,644	82,644	0.702	2.7×10^{-5}
IV	$\mathcal{T}(11)$	PC	2	—	—	909,090	909,091	0.851	2.3×10^{-3}
V	$\mathcal{T}(7,2)$	ERS	5	PC	2	2,082	2,086	0.474	3.3×10^{-8}
VI	$\mathcal{T}(7,2)$	ERS	4	PC	2	14,577	14,644	0.595	5.0×10^{-6}
VII	$\mathcal{T}(7,2)$	HM	3	PC	2	102,040	102,232	0.716	2.8×10^{-5}
VIII	$\mathcal{T}(2,5)$	HM	3	PC	2	250,000	250,000	0.771	1.3×10^{-3}
Noncoded case							10^7	1.000	2.3×10^{-3}
7-Digital postal code							142,705	0.736	1.4×10^{-3}

Source: *[KANE04b]. © 2004 IEEE*

Note: RS: Reed-Solomon code, ERS: Extended Reed-Solomon code, HM: Hamming code; PC: Simple parity-check code. The approximation of $|\mathbf{C}|$ is derived from Theorem 13.5.

denotes the range of F. Given the equiprobability condition for $\mathbf{C}_i$, the probability of every symbol in $(\mathbf{x}_0\ \mathbf{x}_1\ \ldots\ \mathbf{x}_{n-1}) \in \mathbf{X}$ being included in $\Phi(F)$ is

$$\left(\frac{|\Phi(F)|}{|R(q_1, q_2, \ldots, q_c)|}\right)^n = \left(\frac{M}{\prod_{i=1}^{c} q_i}\right)^n.$$

Therefore the number of codewords in $\mathbf{C}$ is approximated by

$$|\mathbf{C}| \simeq |\mathbf{X}| \times \left(\frac{M}{\prod_{i=1}^{c} q_i}\right)^n = \prod_{i=1}^{c} |\mathbf{C}_i| \times \left(\frac{M}{\prod_{i=1}^{c} q_i}\right)^n.$$

Q.E.D.

Table 13.4, shown later, indicates that this approximation is highly accurate for the class of codes indicated.

3. Decoding Procedure

A maximum likelihood decoding, in general, gives a low probability of erroneous decoding. A received word $U' = (u'_0\ u'_1\ \ldots\ u'_{n-1})$ can be decoded by a brute force search through the set $\mathbf{C}$ of codewords to find a codeword $(\hat{u}_0\ \hat{u}_1\ \ldots\ \hat{u}_{n-1}) \in \mathbf{C}$ that maximizes the probability

$$\prod_{i=0}^{n-1} p(u'_i|\hat{u}_i).$$

In the conventional block codes used in communication and memory systems, a brute force search, in general, requires a prohibitively long time because there exists a huge number of codewords. In contrast, the indicated code is designed to generate a codebook for M-ary data, such as postal codes and product numbers, where the number of codewords is relatively small, and also the constraint on decoding delay is not so severe compared to the communication and memory systems. Therefore the maximum likelihood decoding using brute force search is feasible for the codes.

4. Evaluation

The codes are evaluated in terms of the number of codewords and the decoded SER (symbol error rate), which is defined as

$$\text{Decoded SER} = \frac{\text{Total number of erroneously decoded symbols}}{\text{Total number of decoded symbols}}.$$

Figure 13.13 illustrates the trees used in the following evaluation, which is based on the confusion matrix $\mathbf{P}$ for handwritten numeral recognition systems shown in Table 13.1. The trees $\mathcal{T}(11), \mathcal{T}(2,5)$, and $\mathcal{T}(7,2)$ are constructed using the new set-partitioning algorithm described previously. Table 13.4 shows the results of computer simulations, which indicate the relation between the number of codewords $|\mathbf{C}|$, the code rate $(\log_M|\mathbf{C}|)/n$, and the decoded SER for M-ary asymmetric error correcting codes having length $n = 7$, where $c \in \{1,2\}$. The code construction parameters used in the simulations are given by $\mathcal{T}, \mathbf{C}_1, d_1, \mathbf{C}_2$, and d_2. Also shown in the table are the approximations of $|\mathbf{C}|$ given by

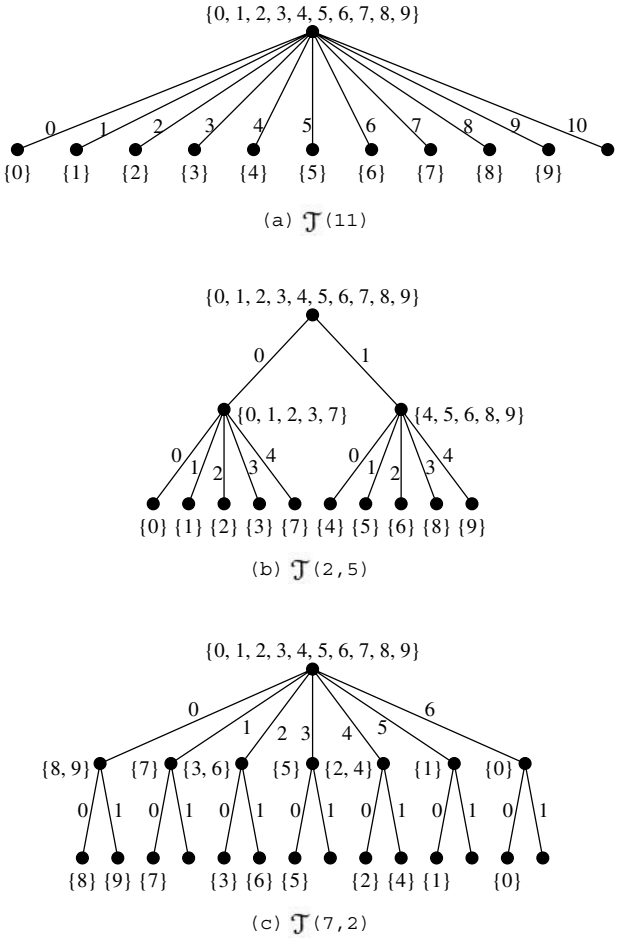


Figure 13.13 *Trees used for evaluation.* Source: *[KANE04b]. © 2004 IEEE.*

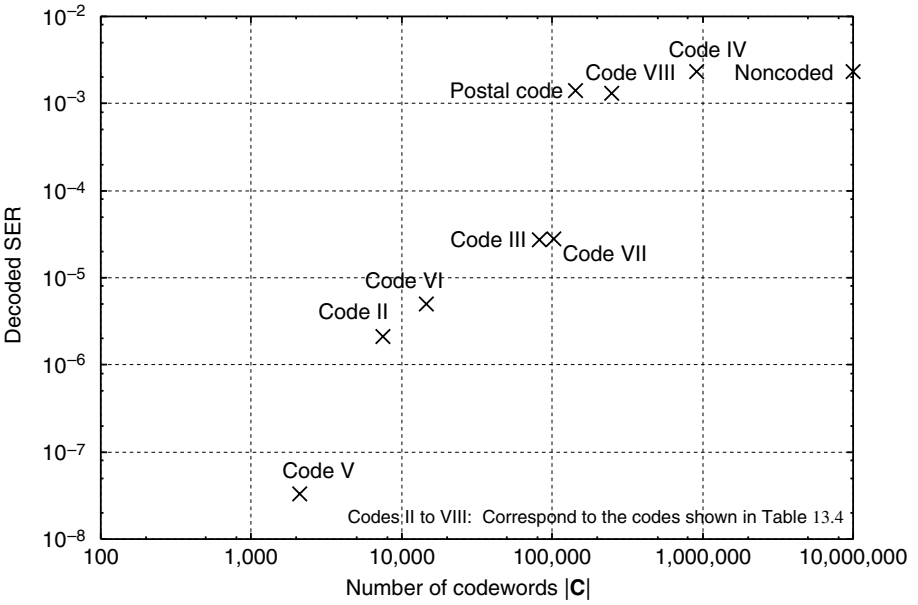


Figure 13.14 *Relation between number of codewords $|C|$ and decoded SER (symbol error rate) for $n = 7$.* Source: *[KANE04b]. © 2004 IEEE.*

Theorem 13.5. The last two rows in the table show the decoded SER for the noncoded case, where $|\mathbf{C}| = 10,000,000$, and for the 7-digit postal code, where $|\mathbf{C}| = 142,705$. Note that the postal code does not have explicit error correction capability. Figure 13.14 illustrates the relation between $|\mathbf{C}|$ and the decoded SER of the codes, where each label of II through VIII corresponds to the code shown in Table 13.4. Code IV has a high decoded SER because this code has a small minimum Hamming distance $d_1 = 2$, which results in many erroneous decodings. All of the other codes achieve a low decoded SER with a reasonable number of codewords. For example, if the total number of postal codes is reduced from $142,705$ to $102,232$, then the decoded SER drops from 1.4×10^{-3} to 2.8×10^{-5} while using the code VII.

13.3 NONSYSTEMATIC *M*-ARY ASYMMETRIC ERROR CORRECTING CODES WITH DELETION / INSERTION / ADJACENT-SYMBOL-TRANSPOSITION ERROR CORRECTION CAPABILITIES

Another new class of M-ary asymmetric error correcting codes is suitable for data entry systems, such as keyboard input systems and character recognition systems. The codes are capable of correcting single deletion / insertion / adjacent-symbol-transposition errors as well as correcting single asymmetric errors [KANE04c].

In order to correct these errors, single deletion / insertion error correcting codes [TENE84] and single adjacent-symbol-transposition error correcting codes [TANG70] have been proposed. These conventional codes, however, are designed so as to correct only one type of error—asymmetric errors, *deletion / insertion errors,* or *transposition errors*—and therefore these codes are not necessarily suitable for data entry systems where three or four error types sometimes occur.

The design for this class of codes is based on a combination of an M-ary single deletion / insertion error correcting code, a single adjacent-symbol-transposition error correcting code over $GF(q)$, and a mapping derived from *vertex coloring* for an error directionality graph. This section also presents a simple decoding procedure and an evaluation of the designed codes.

13.3.1 Preliminaries

The indicated M-ary codes are capable of correcting four types of errors shown below:

Asymmetric error. A symbol $u_i(u_j)$ in a codeword is changed to another symbol $u_i'(u_j')$ with probability $p(u_i'|u_i) = p_i$ $(p(u_j'|u_j) = p_j)$, and $0 \le p_i \ne p_j < \varepsilon,\ i \ne j$, where ε is the threshold error probability given in advance.

Adjacent-symbol-transposition error. The order of two adjacent symbols in a codeword is reversed.

Deletion error. A symbol in a codeword is deleted, and hence the word length is shortened by one symbol.

Insertion error. An extra symbol is inserted into a codeword, and hence the word length is lengthened by one symbol.

The following theorem gives a class of M-ary codes capable of correcting single insertion errors as well as correcting single deletion errors.

Theorem 13.6 [TENE84] *The following code* $\mathbf{C}_{DI}$ *is a single deletion / insertion error correcting code over* $\mathbf{Z}_M = \{0, 1, \ldots, M-1\}$:

$$\mathbf{C}_{DI} = \left\{ (v_0\ v_1\ \ldots\ v_{n-1}) \,\middle|\, \left(\sum_{i=0}^{n-2} (i+1)\overline{v_i} \right) \bmod n = 0,\ \left(\sum_{i=0}^{n-1} v_i \right) \bmod M = 0 \right\}.$$

Here $v_i \in \mathbf{Z}_M$ *for all* $i \in \{0, 1, \ldots, n-1\}$ *and* $(\overline{v_0}\ \overline{v_1}\ \ldots\ \overline{v_{n-2}})$ *is an associate vector for* $(v_0\ v_1\ \ldots\ v_{n-1})$ *defined as follows:*

$$\overline{v_i} = \begin{cases} 1 & (v_i \le v_{i+1}), \\ 0 & (v_i > v_{i+1}). \end{cases}$$

Note in the theorem that the associate vector is a codeword of the binary single deletion / insertion error correcting code proposed in [LEVE66].

Theorem 13.7 *Let* β *be a primitive element in* $GF(q^r)$*, where* q *is a prime or power of a prime. The null space of*

$$\boldsymbol{H} = \left[\beta^0\, \beta^1\, \ldots\, \beta^{n-1} \right]_{r \times n} \tag{13.6}$$

is a single-symbol error correcting code over $GF(q)$*, where* $n = (q^r - 1)/(q-1)$ *and* β^i *is a column vector having length* r.

This theorem can be proved by showing that two column vectors in $\mathbf{H}$ are linearly independent.

Theorem 13.8 *The null space of* $\boldsymbol{H}$ *given in Eq. (13.6) is a single adjacent-symbol-transposition error correcting code over* $GF(q)$.

Proof Let

$$X = (x_0\ x_1\ \ldots\ x_{i-2}\ \underline{x_{i-1}\ x_i}\ x_{i+1}\ \ldots\ x_{n-1})$$

be a transmitted codeword, and let

$$X' = (x_0\ x_1\ \ldots\ x_{i-2}\ \underline{x_i\ x_{i-1}}\ x_{i+1}\ \ldots\ x_{n-1})$$

be a received word having single adjacent-symbol-transposition error in x_{i-1} and x_i. Syndrome S for X' is given as follows:

$$\begin{aligned} S = X'\mathbf{H}^T &= (x_i\beta^{i-1} + x_{i-1}\beta^i) - (x_{i-1}\beta^{i-1} + x_i\beta^i) \\ &= (x_{i-1} - x_i)(\beta^i - \beta^{i-1}). \end{aligned}$$

Hence the adjacent-symbol-transposition errors can be corrected if the following condition is satisfied:

$$\begin{aligned} &(\beta^i - \beta^{i-1}) \neq a(\beta^j - \beta^{j-1}) \\ &\textit{for all } i, j \in \{1, 2, , n-1\},\ \textit{and for all } a \in GF(q) - \{0\}, \end{aligned} \tag{13.7}$$

where $i < j$. Suppose that there exist i, j, and a satisfying

$$(\beta^i - \beta^{i-1}) = a(\beta^j - \beta^{j-1}),$$

then the following equations hold:

$$\begin{aligned} \beta^{i-1}(\beta - \beta^0) &= a\beta^{j-1}(\beta - \beta^0), \\ \beta^{i-1} &= a\beta^{j-1}, \\ (\beta^{i-1})^{q-1} &= (a\beta^{j-1})^{q-1}, \\ \beta^{(i-1)(q-1)} &= \beta^{(j-1)(q-1)}. \quad (\because\ a^{q-1} = 1) \end{aligned}$$

The equations above imply $i = j$ because β is a primitive element in $GF(q^r)$ and $0 \leq (i-1)(q-1) < (j-1)(q-1) < q^r - 1$. This contradicts the hypothesis of $i < j$, and hence Eq. (13.7) holds. Q.E.D.

By Theorems 13.7 and 13.8, the null space of **H** can be used as either a single-symbol error correcting code or a single adjacent-symbol-transposition error correcting code. Note that the code defined by **H** cannot distinguish between single-symbol errors and single adjacent-symbol-transposition errors.

13.3.2 Code Design

The codes are defined over the set of M-ary symbols $\mathbf{A} = \{a_0, a_1, , a_{M-1}\}$. For given asymmetric error probabilities $p(a_j|a_i)$ and the threshold error probability ε, error directionality graph G is defined in Definition 13.3.

Mapping f is defined as a *vertex coloring function* for the error directionality graph G, shown below.

Definition 13.11 For the set of M-ary symbols $\mathbf{A}$ and Galois field $GF(q)$, f is defined as a mapping from $\mathbf{A}$ to $GF(q)$ satisfying the following condition:

$$[(a_i \rightarrow a_j) \in \mathbf{E}] \rightarrow [f(a_i) \neq f(a_j)]$$
$$\textit{for all } a_i, a_j \in \mathbf{A}, a_i \neq a_j,$$

where $\mathbf{E}$ is the set of edges in the error directionality graph $G = (\mathbf{V}, \mathbf{E})$. □

Definition 13.12 For the set of M-ary symbols $\mathbf{A}$ and the set of integers $\mathbf{Z}_M = \{0, 1, \ldots, M-1\}$, g is defined as an arbitrary *bijective mapping* from $\mathbf{A}$ to $\mathbf{Z}_M$. □

The following theorem gives a new class of nonsystematic M-ary codes capable of correcting single deletion / insertion errors and single adjacent-symbol-transposition errors as well as correcting single asymmetric errors [KANE04c].

Theorem 13.9 *Let $\boldsymbol{C}$ be a set of codewords $\boldsymbol{u} = (u_0\ u_1\ \ldots\ u_{n-1})$ satisfying the following three conditions:*

1. $\left(\sum_{i=0}^{n-1} g(u_i)\right) \bmod M = 0$,
2. $\left(\sum_{i=0}^{n-2} (i+1)\overline{g(u_i)}\right) \bmod n = 0$,
3. $(f(u_0), f(u_1), \ldots, f(u_{n-1}))\boldsymbol{H}^T = \boldsymbol{0}$,

where $u_i \in \boldsymbol{A} = \{a_0, a_1, \ldots, a_{M-1}\}$ for all $i \in \{0, 1, \ldots, n-1\}$, $(\overline{g(u_0)}\ \overline{g(u_1)}\ \ldots\ \overline{g(u_{n-2})})$ is the associate vector for $(g(u_0)\ g(u_1)\ \ldots\ g(u_{n-1}))$, $\boldsymbol{H}$ is the parity-check matrix given by Eq. (13.6), and $\boldsymbol{0}$ is a zero vector. The code $\boldsymbol{C}$ is a nonsystematic M-ary asymmetric error correcting code with deletion / insertion / adjacent-symbol-transposition error correction capabilities. In other words, a received word $\boldsymbol{v} = (v_0\ v_1\ \ldots\ v_{n'-1})$ is correctly decoded if $\boldsymbol{v}$ has no error or has one of the following errors:

- ***a.*** *Single asymmetric errors,*
- ***b.*** *Single deletion / insertion errors,*
- ***c.*** *Single adjacent-symbol-transposition errors existing in (v_i, v_{i+1}), where $f(v_i) \neq f(v_{i+1})$.*

Here a single asymmetric error is a single-symbol error indicated by an edge $(u_i \rightarrow u_j) \in \boldsymbol{E}$, that is, an error in which symbol u_i is changed to another symbol v_i with probability $p(v_i|u_i) > \varepsilon$.

Proof Let $\boldsymbol{u} = (u_0\ u_1\ \ldots\ u_{n-1})$ be a codeword having length n, and let $\boldsymbol{v} = (v_0\ v_1\ \ldots\ v_{n'-1})$ be a received word having length n'. Syndromes S_1 and S_2 for $\boldsymbol{v}$ is determined as follows:

$$S_1 = \left(\sum_{i=0}^{n'-1} g(v_i)\right) \bmod M,$$
$$S_2 = \left(\sum_{i=0}^{n'-2} (i+1)\overline{g(v_i)}\right) \bmod n,$$

where $(\overline{g(v_0)}\ \overline{g(v_1)}\ \ldots\ \overline{g(v_{n'-2})})$ is the associate vector for $G(\boldsymbol{v}) = (g(v_0)\ g(v_1)\ \ldots\ g(v_{n'-1}))$. For the case where $n' = n$, syndrome S_3 is determined as

$$S_3 = F(\boldsymbol{v})\mathbf{H}^T = (f(v_0)\ f(v_1)\ \ldots\ f(v_{n'-1}))\mathbf{H}^T,$$

where $F(\boldsymbol{v}) = (f(v_0)\ f(v_1)\ \ldots\ f(v_{n'-1}))$. The received word $\boldsymbol{v}$ is decoded as follows:

1. If $n' = n - 1$, then $\boldsymbol{v}$ has a single deletion error. This error is correctable because $G(\boldsymbol{u}) = (g(u_0)\ g(u_1)\ \ldots\ g(u_{n-1}))$ is a codeword of the single deletion / insertion error correcting code given by Theorem 13.6.
2. If $n' = n + 1$, then $\boldsymbol{v}$ has a single insertion error. This error is also correctable because $G(\boldsymbol{u}) = (g(u_0)\ g(u_1)\ \ldots\ g(u_{n-1}))$ is a codeword of the single deletion / insertion error correcting code given by Theorem 13.6.
3. If $n' = n, S_1 \neq 0$, and $S_3 \neq \boldsymbol{0}$, then $\boldsymbol{v}$ has a single asymmetric error. This error is correctable for the following reasons:

   - The mapping f satisfies $f(a_i) \neq f(a_j)$ for any pair of symbols a_i and a_j having error probability $p(a_j|a_i) > \varepsilon$.
   - The vector $F(\boldsymbol{u}) = (f(u_0)\, f(u_1)\ \ldots\ f(u_{n-1}))$ is a codeword of the single-symbol error correcting code defined by $\mathbf{H}$, and hence the error location l in $F(\boldsymbol{v}) = (f(v_0)\, f(v_1)\ \ldots\ f(v_{n'-1}))$ is determined by S_3.
   - The syndrome S_1 satisfies $S_1 = g(v_l) - g(u_l) \bmod M$, and hence the original correct symbol u_l is obtained from S_1.

4. If $n' = n, S_1 = 0$, and $S_3 \neq \boldsymbol{0}$, then $\boldsymbol{v}$ has a single adjacent-symbol-transposition error in (v_l, v_{l+1}), where $f(v_l) \neq f(v_{l+1})$ and $S_3 = a(\beta^l - \beta^{l+1})$ for $\exists a \in GF(q) - \{0\}$. This error is correctable because $F(\boldsymbol{u}) = (f(u_0)\, f(u_1)\ \ldots\ f(u_{n-1}))$ is a codeword of the single adjacent-symbol-transposition error correcting code defined by $\mathbf{H}$, and therefore error location is uniquely determined by S_3.
5. If $n' = n, S_1 = 0, S_2 = 0$, and $S_3 = \boldsymbol{0}$, then $\boldsymbol{v}$ has no error.
6. Otherwise, $\boldsymbol{v}$ has uncorrectable errors.

Therefore the code $\mathbf{C}$ has the required error correction capabilities. Q.E.D.

Figure 13.15 illustrates conditions 1, 2, and 3 for the codewords of the code.

Corollary 13.1 *If the mapping f used in Theorem 13.9 is injective, then $\mathbf{C}$ is capable of correcting the following three types of errors:*

a. *Single asymmetric errors, where $\varepsilon = 0$,*
b. *Single deletion / insertion errors,*
c. *Single adjacent-symbol-transposition errors.*

The proof of this corollary is obvious from Theorem 13.9.

The codebook $\mathbf{C}$ of the code, which corresponds to a codebook of, for example, postal codes and product numbers, is generated by a computer enumerating M-ary words $\boldsymbol{u} = (u_0\ u_1\ \ldots\ u_{n-1})$ that satisfy conditions 1, 2, and 3 of Theorem 13.9.

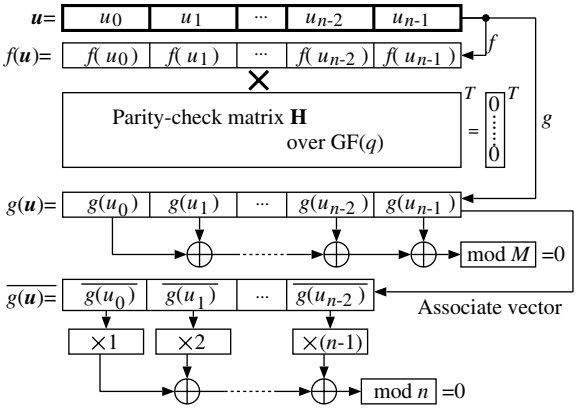


Figure 13.15 *Conditions for codeword* ***U****.* Source: *[KANE04c]. © 2004 IEEE.*

13.3.3 Numeric Keypad Code Example

This subsection presents an example of the code for numeric keypads, where $\mathbf{A} = \{0, 1, \ldots, 9\}$ [KANE04c]. Figure 13.16 shows a typical layout of numeric keypads and error directionality graph G for the keypads.

1. Code Design

Let $f : \mathbf{A} \rightarrow GF(7)$ be a mapping defined as follows:

$$f(0) = 0, \quad f(1) = 1, \quad f(2) = 2, \quad f(3) = 3, \quad f(4) = 4,$$
$$f(5) = 5, \quad f(6) = 6, \quad f(7) = 1, \quad f(8) = 2, \quad f(9) = 3.$$

The vertices in G are colored with elements in $GF(7)$ by the mapping f, as shown in Figure 13.16. Let $g : \mathbf{A} \rightarrow \mathbf{Z}_{10}$ be a mapping defined as $g(i) = i$. The parity-check matrix $\mathbf{H}$ over $GF(7)$ with code length $n = 7$ is given as follows:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 4 & 3 & 6 & 6 & 4 \\ 0 & 1 & 6 & 5 & 5 & 1 & 5 \end{bmatrix}. \tag{13.8}$$

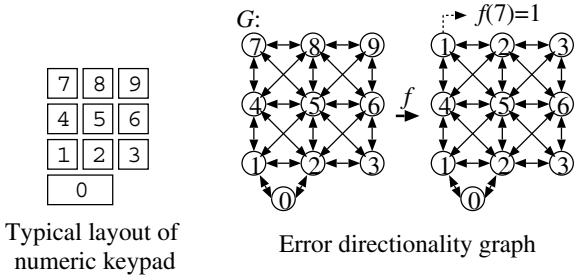


Figure 13.16 *Typical layout of a numeric keypad and error directionality graphs.* Source: *[KANE04c]. © 2004 IEEE.*

Consider the following word $\boldsymbol{u}$ over $\mathbf{A} = \{0, 1, \ldots, 9\}$:

$$\boldsymbol{u} = (u_0\ u_1\ u_2\ u_3\ u_4\ u_5\ u_6) = (4\ 8\ 8\ 3\ 5\ 2\ 0).$$

The following vectors are obtained for $\boldsymbol{u}$:

$$\begin{aligned} F(\boldsymbol{u}) &= (f(u_0)\, f(u_1)\ \ldots\ f(u_6)) = (4\ 2\ 2\ 3\ 5\ 2\ 0), \\ G(\boldsymbol{u}) &= (g(u_0)\ g(u_1)\ \ldots\ g(u_6)) = (4\ 8\ 8\ 3\ 5\ 2\ 0), \\ \overline{G(\boldsymbol{u})} &= (\overline{g(u_0)}\ \overline{g(u_1)}\ \ldots\ \overline{g(u_5)}) = (1\ 1\ 0\ 1\ 0\ 0). \end{aligned}$$

Hence $\boldsymbol{u} = (4, 8, 8, 3, 5, 2, 0)$ is a codeword because the following equations hold:

$$\begin{aligned} &\left(\sum_{i=0}^{6} g(u_i)\right) \bmod 10 = (4 + 8 + 8 + 3 + 5 + 2 + 0) \bmod 10 = 0, \\ &\left(\sum_{i=0}^{5} (i+1)\overline{g(u_i)}\right) \bmod 7 = (1 + 2 + 4) \bmod 7 = 0, \\ &F(\boldsymbol{u})\mathbf{H}^T = (4\ 2\ 2\ 3\ 5\ 2\ 0)\mathbf{H}^T = (0\ 0). \end{aligned}$$

2. Decoding Procedure

Errors that occur in the preceding codeword $\boldsymbol{u} = (4, 8, 8, 3, 5, 2, 0)$ are corrected as follows:

Asymmetric Error Correction. Let $\boldsymbol{v} = (v_0\ v_1\ \ldots\ v_6) = (4\ 8\ \mathbf{5}\ 3\ 5\ 2\ 0)$ be a received word having an asymmetric error in v_2, that is, $u_2 = 8$ is changed to $v_2 = 5$. The following vectors are determined for $\boldsymbol{v}$:

$$\begin{aligned} F(\boldsymbol{v}) &= (4\ 2\ 5\ 3\ 5\ 2\ 0), \\ G(\boldsymbol{v}) &= (4\ 8\ 5\ 3\ 5\ 2\ 0), \\ \overline{G(\boldsymbol{v})} &= (1\ 0\ 0\ 1\ 0\ 0). \end{aligned}$$

Syndromes S_1, S_2, and S_3 are calculated as follows:

$$\begin{aligned} S_1 &= (4 + 8 + 5 + 3 + 5 + 2 + 0) \bmod 10 = 7, \\ S_2 &= (1 + 4) \bmod 7 = 5, \\ S_3 &= (4\ 2\ 5\ 3\ 5\ 2\ 0) \begin{bmatrix} 1\ 0\ 4\ 3\ 6\ 6\ 4 \\ 0\ 1\ 6\ 5\ 5\ 1\ 5 \end{bmatrix}^T \\ &= (5,\ 4) = 3 \times (4,\ 6). \end{aligned}$$

Thus the received word $\boldsymbol{v}$ has an asymmetric error because $S_1 \neq 0$ and the length of $\boldsymbol{v}$ is $n' = 7$. The syndrome S_3 indicates that the error exists in v_2 because S_3 is a multiple of the column vector $(4,\ 6)^T$. The original correct symbol $\widetilde{v}_2$ is determined using S_1 as follows:

$$\widetilde{v}_2 = g^{-1}((g(v_2) - S_1) \mod M) = g^{-1}(-2 \bmod 10) = 8.$$

Finally, corrected word $(4\ 8\ \mathbf{8}\ 3\ 5\ 2\ 0)$ is obtained.

Adjacent-Symbol-Transposition Error Correction. Let $\boldsymbol{v} = (v_0\ v_1\ \ldots\ v_6) = (4\ 8\ \mathbf{3}\ \mathbf{8}\ 5\ 2\ 0)$ be a received word having an adjacent-symbol-transposition error in v_2 and v_3; that is, $(u_2, u_3) = (8, 3)$ is changed to $(v_2, v_3) = (u_3, u_2) = (3, 8)$. The following vectors are determined for $\boldsymbol{v}$:

$$F(\boldsymbol{v}) = (4\ 2\ 3\ 2\ 5\ 2\ 0),$$
$$G(\boldsymbol{v}) = (4\ 8\ 3\ 8\ 5\ 2\ 0),$$
$$\overline{G(\boldsymbol{v})} = (1\ 0\ 1\ 0\ 0\ 0).$$

Syndromes S_1, S_2, and S_3 are calculated as follows:

$$S_1 = (4 + 8 + 3 + 8 + 5 + 2 + 0) \bmod 10 = 0,$$
$$S_2 = (1 + 3) \bmod 7 = 4,$$
$$S_3 = (4\ 2\ 3\ 2\ 5\ 2\ 0)\begin{bmatrix} 1 & 0 & 4 & 3 & 6 & 6 & 4 \\ 0 & 1 & 6 & 5 & 5 & 1 & 5 \end{bmatrix}^T$$
$$= (1, 1) = 1 \times ((4, 6) - (3, 5)).$$

Thus the received word $\boldsymbol{v}$ has an adjacent-symbol-transposition error because $S_1 = 0$, $S_3 \neq (0, 0)$, and the length of $\boldsymbol{v}$ is $n' = 7$. The syndrome S_3 indicates that the error exists in (v_2, v_3) because S_3 is a multiple of $(4, 6) - (3, 5) = (1, 1)$. Hence the correct word $(4\ 8\ \mathbf{8}\ \mathbf{3}\ 5\ 2\ 0)$ can be obtained by reversing the order of the received symbols v_2 and v_3.

Deletion Error Correction. Let $\boldsymbol{v} = (v_0\ v_1\ \ldots\ v_5) = (4\ 8\ 8\ 5\ 2\ 0)$ be a received word having a deletion error in $u_3 = 3$. The following vectors are determined for $\boldsymbol{v}$:

$$F(\boldsymbol{v}) = (4\ 2\ 2\ 5\ 2\ 0),$$
$$G(\boldsymbol{v}) = (4\ 8\ 8\ 5\ 2\ 0),$$
$$\overline{G(\boldsymbol{v})} = (1\ 1\ 0\ 0\ 0).$$

Syndromes S_1 and S_2 are calculated as follows:

$$S_1 = (4 + 8 + 8 + 5 + 2 + 0) \bmod 10 = 7,$$
$$S_2 = (1 + 2) \bmod 7 = 3.$$

The received word $\boldsymbol{v}$ has a deletion error because the length of the received word is $n' = n - 1 = 6$. The deletion error in $G(\boldsymbol{v}) = (4\ 8\ 8\ 5\ 2\ 0)$ can be corrected using S_1 and S_2 because the original vector $G(\boldsymbol{u})$ for the transmitted word $\boldsymbol{u}$ is a codeword of the M-ary single deletion / insertion error correcting code. Therefore the following correct M-ary vector is obtained:

$$\widetilde{G}(\boldsymbol{v}) = (4\ 8\ 8\ 3\ 5\ 2\ 0).$$

Finally, corrected word is determined as follows:

$$(g^{-1}(4)\ g^{-1}(8)\ g^{-1}(8)\ g^{-1}(3)\ g^{-1}(5)\ g^{-1}(2)\ g^{-1}(0))$$
$$= (4\ 8\ 8\ \mathbf{3}\ 5\ 2\ 0).$$

Insertion Error Correction. Let $\boldsymbol{v} = (v_0\ v_1\ \ldots\ v_7) = (4\ 8\ 8\ 3\ 5\ 2\ \mathbf{9}\ 0)$ be a received word having an insertion error between $u_5 = 2$ and $u_6 = 0$. The following vectors are determined for $\boldsymbol{v}$:

$$F(\boldsymbol{v}) = (4\ 2\ 2\ 3\ 5\ 2\ 3\ 0),$$
$$G(\boldsymbol{v}) = (4\ 8\ 8\ 3\ 5\ 2\ 9\ 0),$$
$$\overline{G(\boldsymbol{v})} = (1\ 1\ 0\ 1\ 0\ 1\ 0).$$

Syndromes S_1 and S_2 are calculated as follows:

$$S_1 = (4 + 8 + 8 + 3 + 5 + 2 + 9 + 0) \bmod 10 = 9,$$
$$S_2 = (1 + 2 + 4 + 6) \bmod 7 = 6.$$

The received word $\boldsymbol{v}$ has an insertion error because the length of the received word is $n' = n + 1 = 8$. The insertion error in $G(\boldsymbol{v}) = (4\ 8\ 8\ 3\ 5\ 2\ 9\ 0)$ can be corrected using S_1 and S_2 because the original vector $G(\boldsymbol{u})$ for the transmitted word $\boldsymbol{u}$ is a codeword of the M-ary single deletion / insertion error correcting code. Therefore the following correct M-ary vector is obtained:

$$\widetilde{G}(\boldsymbol{v}) = (4\ 8\ 8\ 3\ 5\ 2\ 0).$$

Finally the following determines the correct word:

$$(g^{-1}(4)\ g^{-1}(8)\ g^{-1}(8)\ g^{-1}(3)\ g^{-1}(5)\ g^{-1}(2)\ g^{-1}(0))$$
$$= (4\ 8\ 8\ 3\ 5\ 2\ 0).$$

3. Evaluation

Here we evaluates the number of codewords $|\mathbf{C}|$ and the code rate $R = (\log_M|\mathbf{C}|)/n$ of the codes for numeric keypads. Table 13.5 shows the number of codewords and the code rate of the codes for the error directionality graphs G_A and G_B shown in Figure 13.17, where $\mathbf{A} = \{0, 1, \ldots, 9\}$ and the code length is $n = 7$. The matrix $\mathbf{H}_7$ is given by Eq. (13.8), and the matrix $\mathbf{H}_{11}$ is given as

$$\mathbf{H}_{11} = \begin{bmatrix} 1 & 0 & 4 & 7 & 9 & 8 & 6 \\ 0 & 1 & 10 & 5 & 2 & 7 & 1 \end{bmatrix}.$$

Also $f_7 : \mathbf{A} \to GF(7)$ is the mapping shown in the previous subsection, and $f_{11} : \mathbf{A} \to GF(11)$ is an injective mapping satisfying $f(i) = i$ for all $i \in \{0, 1, \ldots, 9\}$. Although the code rate for G_B is lower than that for G_A, the code for G_B has higher error

TABLE 13.5 Number of Codewords and Code Rates

Error directionality graph	q	Parity-check matrix $\mathbf{H}$	Mapping f	Number of codewords	Code Rate
G_A	7	$\mathbf{H}_7$	f_7	2,941	0.495
G_B	11	$\mathbf{H}_{11}$	f_{11}	1,171	0.438

Source: *[KANE04c]. © 2004 IEEE.*

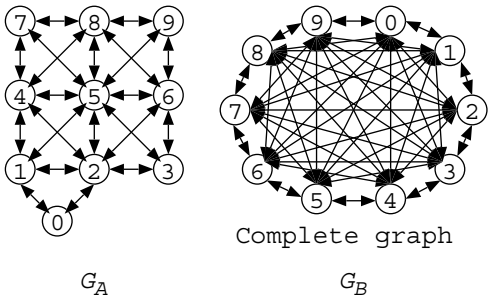


Figure 13.17 *Error directionality graphs used for evaluation.* Source: *[KANE04c]. © 2004 IEEE.*

correction capabilities. More precisely, the code for G_B is capable of correcting single deletion / insertion errors and single adjacent-symbol-transposition errors as well as correcting single-symbol errors, whereas the code for G_A corrects neither single-symbol errors $(a_i \rightarrow a_j) \notin \mathbf{E}$ nor single adjacent-symbol-transposition errors existing in (u_i, u_{i+1}), where $f(u_i) = f(u_{i+1})$.

13.4 CODES FOR TWO-DIMENSIONAL MATRIX SYMBOLS

Bar codes have wide applications, such as in point-of-sales (POS) systems, in mail delivery services, and in transport industries. Two-dimensional (2D) codes [PALV92], which meet a need to encode significantly larger data than the conventional bar codes, are the most popular in use today. There are two types of 2D codes: stacked-type codes and matrix-type codes. The former include stacked bar codes of CODE 49, CODE 16K, PDF417, and so forth. The latter include VERICODE, OP CODE, MAXI CODE [MATR], and QR code [JAPA02]. Some 2D codes are even being used for sales items and various other store products, for parts and components in factories, and for packages in shipping industries and transport industries. The recording density of conventional bar codes is low because the codes do not effectively utilize the recording space in the vertical direction, and the capacity is strictly limited by size. Two-dimensional codes express recording data by two-dimensional black-and-white cell patterns that capture a large volume of detailed information on the items. These codes include some redundancy so that they can restore symbols partially damaged by blots, scratches, and so on.

In this section, the codes for two-dimensional matrix symbols are presented for QR codes and two-dimensional unidirectional error correcting codes.

13.4.1 QR Codes

Quick response codes (i.e., QR codes) were developed by the Denso Corporation in Japan. As shown in Figure 13.18(a), the code contains information in both vertical and horizontal directions and is capable of expressing a maximum of 4,296 alphanumeric characters and restoring a maximum of 30% damage in the code symbol due to scratches, blots, and the like, as shown in Figure 13.18(b). The code contains three square-shaped marks at three corners so that the right position of the code symbol can be detected in a 360 degree (omni-directional) high-speed stable reading. This 2D code is standardized in accord with the ISO international standard (ISO/IEC 18004) and the JIS standard (Japanese Industrial Standards, JIS X 0510) [JAPA02].

(a) QR code symbol (b) Disturbances in QR code symbol

Figure 13.18 *Matrix symbol of the QR codes, and the disturbances in a QR code symbol due to blots or scratches.* Source: *[KANE03]. © 2003 IEEE.*

Formation of the QR Code Some control informations of character mode, input character length, and the like, are appended to the main body of the input data added by check information, all expressed in binary numbers. There are several character modes, such as numerical mode, alphanumeric mode, Kanji character mode, and mixed mode. For example, in the alphanumeric mode, there exist 45 characters each named by the integers shown in Table 13.6 [JAPA99].

An important feature of the code exists in the input data formation. That is, in almost all character modes, every two or three characters set is encoded in binary, which expresses the nonbinary input sequence efficiently, and finally compresses the input data. For example, in numeric mode, 8 numeral characters sequence ''01234567'' is assumed to be given as an input. Every three-character set is encoded in a 10-bit binary form, and the remaining two characters in 7 bits, that is,

$$012 = 0000001100,$$
$$345 = 0101011001,$$
$$67 = 1000011.$$

The total input binary data can be expressed as

$$0000001100 \quad 0101011001 \quad 1000011.$$

After some of the header information is appended, we get the input binary sequence of the code. In the numeric mode the total binary length of the input data, except for the header length, can be expressed by

$$Ln = 10 \times \lfloor L_D/3 \rfloor + Ld,$$

TABLE 13.6 Assignment of Alphanumeric Characters

0	0	6	6	C	12	I	18	O	24	U	30	SP	36	.	42
1	1	7	7	D	13	J	19	P	25	V	31	$	37	/	43
2	2	8	8	E	14	K	20	Q	26	W	32	%	38	:	44
3	3	9	9	F	15	L	21	R	27	X	33	*	39		
4	4	A	10	G	16	M	22	S	28	Y	34	+	40		
5	5	B	11	H	17	N	23	T	29	Z	35	−	41		

Source: *[JAPA02].*

where L_D is the input character length, $\lfloor x \rfloor$ denotes the largest integer smaller than or equal to x, and Ld is defined by

$$\begin{aligned} &\text{if } L_D \bmod 3 = 0, \text{ then } Ld = 0,\\ &\text{if } L_D \bmod 3 = 1, \text{ then } Ld = 4,\\ &\text{if } L_D \bmod 3 = 2, \text{ then } Ld = 7. \end{aligned}$$

If we encode each numeral in 4 bits, this example sequence would require total 32 bits. However, encoding with $L_D = 8$ requires 27 bits, that is, compressed and shortened by 5 bits.

In alphanumeric mode, every two characters set is encoded. For example, the five-character sequence "AC-42" is first encoded according to Table 13.6 to

$$10,\ 12,\ 41,\ 4,\ 2.$$

So every two-character set is expressed as $(10,\ 12)\ (41,\ 4)\ (2)$. Each set is encoded in binary form as

$$\begin{aligned} (10,\ 12) = 10 \times 45 + 12 = 462: &\quad 00111001110,\\ (41,\ 4) = 41 \times 45 + 4 = 1849: &\quad 11100111001,\\ (2) = 2: &\quad 000010. \end{aligned}$$

The total binary data can be expressed as

$$00111001110 \quad 1100111001 \quad 000010\,.$$

The total binary length of the input data in alphanumeric mode can be expressed by

$$La = 11 \times \lfloor L_D/2 \rfloor + 6 \times (L_D \bmod .\ 2).$$

Next the binary expressed data are divided by 8 bits, each of which is called a *byte*, and then the last byte with less than 8 bits is appended by some 0's to satisfy the last byte by having 8 bits length.

In the case above of 8 numerals encoded "01234567," the binary information of the numeric mode expressed by "0001" and the binary information of the input character code length expressed by "0000001000" are appended ahead to the binary input data "00000011000101011001100011." The 27-bit sequence is then also added by the binary terminal information of "0000" to the end of the sequence. This results in

$$\text{“}\underbrace{0001}_{}\underbrace{0000,\ 00100000}_{012},\ \underbrace{00001100}_{},\ \underbrace{01010110}_{345},\ \underbrace{01100001}_{67},\ \underbrace{10000000}_{}\text{”} \tag{13.9}$$

having six 8-bit bytes. In this case the last underlined three 0's are appended to the last byte in order to satisfy the condition of this byte having 8 bits of length.

Check Information Generation—Encoding— The QR codes have four restoration levels—the L, M, Q, and H levels whereby level L restores around 7% damage in a code symbol, level M around 15%, level Q around 25%, and level H around 30%. The input information is encoded by selecting the RS codes with the appropriate code

TABLE 13.7 RS Codes with Different Restoration Levels

Level	Code length n (bytes)	Check length r (bytes)	RS code (n, k, t)	Code function tc-md
L	26	7	(26,19,2)	2c-5d
M	26	10	(26,16,4)	4c-6d
Q	26	13	(26,13,6)	6c-7d
H	26	17	(26,9,8)	8c-9d
H	196	130	4× (39,13,13), and (40,14,13)	13 × 5c (5 interleaved)

Source: *[JAPA02].*
Note: t: correction length in bytes; tc-md-t-byte error correction and m-byte error detection.

parameters determined according to these restoration levels. If we apply the code in an industrial environment, level Q or H is recommended, whereas level L is adequate in a clean environment. Level M is most frequently applied because it is suited to various environments.

Some simple examples of the QR codes are provided in Table 13.7 for four RS codes, all having code length n of 26 bytes with four different levels. The $(n,\ k,\ t)$ of the RS codes means that the code has a length of $n\ (= 26)$ bytes, an input information length of k bytes, and a correction length of t bytes. The code function tc-md means that the code corrects t-byte errors and detects m-byte errors. It can be easily checked that the value of t/n represents the restoration ratio at each level, since $2/26 = 0.077$ at level L, $4/26 = 0.154$ at level M, $6/26 = 0.231$ at level Q, and $8/26 = 0.307$ at level H.

Another interesting case is that of the code with length $n = 196$ bytes and with level H, shown in the last row of Table 13.7. The code has five (n, k) code blocks, which are composed of four $(39, 13)$ code blocks and one $(40, 14)$ code block, each capable of correcting 13 bytes errors. Hence the total 196 bytes are organized by the following five blocks:

Input Information	Check Information
Block 1: $D_1\ D_2\ \ldots\ D_{13}$	$C_1\ C_2\ \ldots\ C_{26}$
Block 2: $D_{14}\ D_{15}\ \ldots\ D_{26}$	$C_{27}\ C_{28}\ \ldots\ C_{52}$
Block 3: $D_{27}\ D_{28}\ \ldots\ D_{39}$	$C_{53}\ C_{54}\ \ldots\ C_{78}$
Block 4: $D_{40}\ D_{41}\ \ldots\ D_{52}$	$C_{79}\ C_{80}\ \ldots\ C_{104}$
Block 5: $D_{53}\ D_{54}\ \ldots\ D_{65}\ D_{66}$	$C_{105}\ C_{106}\ \ldots\ C_{130}$

In this code organization the recording is performed in a serial form by $D_1\ D_{14}\ D_{27}\ D_{40}\ D_{53}\ D_2\ D_{15}\ D_{28}\ \ldots D_{13}\ D_{26}\ D_{39}\ D_{52}\ D_{65}\ D_{66}\ C_1\ C_{27}\ C_{53}\ C_{79}\ C_{105}\ C_2\ \ldots C_{26}\ C_{52}\ C_{78}\ C_{104}\ C_{130}$. The layout of this information in the code symbol is shown in Figure 13.19. As can be easily observed in the figure, the total 196 bytes encoded in the five code blocks are arranged in an *interleaved layout*. This way the two-dimensional clustered errors (i.e., maximum 5×13 bytes errors) can be corrected by the interleaved code. According to [JAPA02], many RS codes with four restoration levels and with code lengths of 26 to 3,706 bytes can be generated.

Likewise check bytes can be generated by using the RS codes over $GF(2^8)$. If we determine the restoration level, for example, to be H, then in the former case of 8 input numerals, three extra bytes (arbitrary nonzero bytes) should be added to the binary

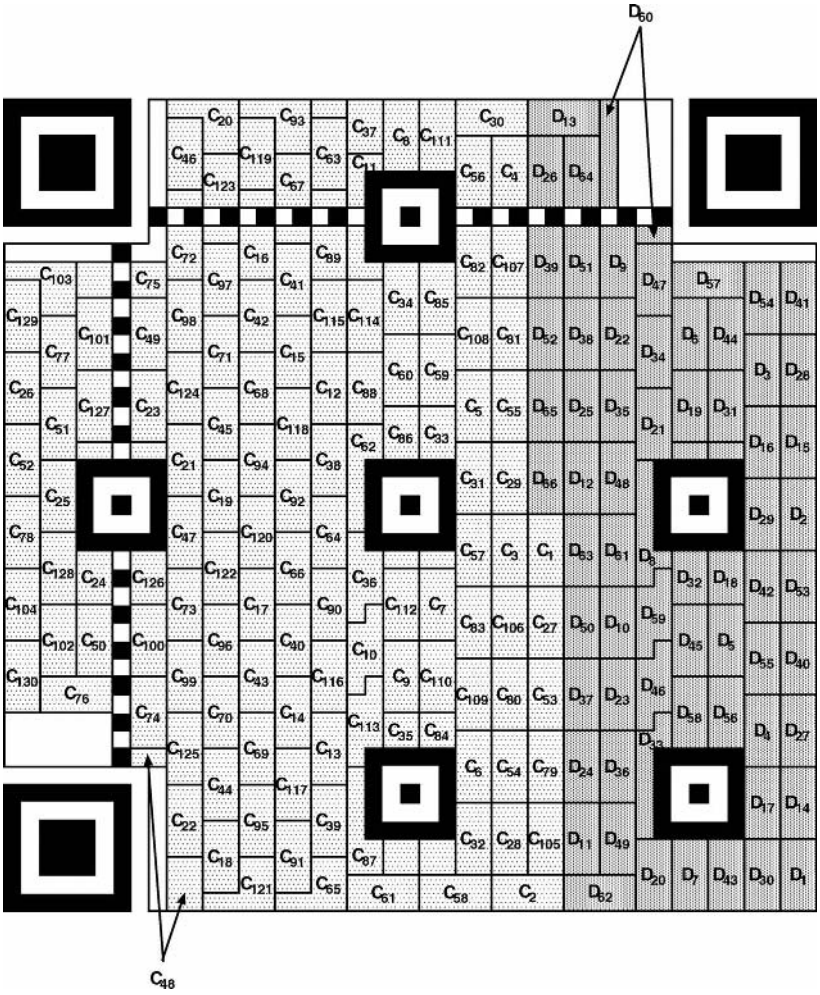


Figure 13.19 *Layout of the QR code interleaved by four (39, 13) RS codes and one (40, 14) RS code.* Source: *[JAPA02].*

sequence of input data having a six-byte length shown in (13.9). Then, in accordance with Table 13.7, the total 9 input information bytes are encoded in RS codes defined by the irreducible polynomial with a 17-th degree over $GF(2^8)$; that is, 17 check bytes are generated. The resulting (26, 9, 8) RS codeword with a 26-byte length is determined, and finally the binary expressed codeword is arranged in a two-dimensional matrix format.

Decoding QR Codes Multiple bytes error correction is generally performed in QR codes. That is, in QR codes, decoding of the RS codes with a minimum Hamming distance $d(= r + 1 = t + m + 1)$ is performed by applying the Berlekamp-Massey algorithm or the Euclid algorithm, as mentioned in Subsections 2.3.5 and 2.3.6. These algorithms determine the byte error locations and the byte error values (i.e., byte error patterns), each by solving t equations, and then these t bytes errors are corrected.

Modulation For the QR code symbols to be read correctly and quickly, it is important to randomize the two-dimensional recording pattern of the code symbol and also to remove, as much as possible, the specified binary pattern of the square-shaped marks from the recording data. To satisfy these, the recording data is modulo-2 added by some specific patterns, that is, modulation is performed. (For more details, the reader should refer to [JAPA02].)

13.4.2 Two-dimensional Unidirectional Clustered Error Correcting Codes

Two-dimensional unidirectional clustered error correcting codes are suitable for high-density two-dimensional matrix symbols [KANE03]. The codes are capable of correcting unidirectional errors confined to a rectangle having l_m rows and l_n columns.

Because these symbols are usually printed on surfaces such as paper, plastic, and metal, they can sometimes be damaged by blots or scratches, as was mentioned before. Further these errors are unidirectional because blotted white cells change into black cells while black cells remain unchanged, and scratched black cells change into white cells while white cells remain unchanged. These types of errors can be effectively corrected by two-dimensional unidirectional clustered error correcting codes.

Efficient one-dimensional unidirectional burst error correcting codes are presented in [PARK90] and [SAIT90b]. Also one-dimensional unidirectional byte error correcting codes have been constructed in [SAOW00]. Two-dimensional clustered error correcting codes, which are capable of correcting any errors confined in a rectangle with l_m rows and l_n columns, are presented in [BREI98]. No efficient two-dimensional unidirectional clustered error correcting codes, however, have been presented. This subsection covers this new class of array codes and shows them to be capable of correcting unidirectional $l_m \times l_n$-clustered errors.

1. One-dimensional Unidirectional Burst Error Correcting Codes

The existing one-dimensional unidirectional ρ-bit burst error correcting codes [PARK90] are abbreviated as 1D-U$_\rho$BEC codes. These codes can be used to construct the two-dimensional unidirectional $l_m \times l_n$-clustered error correcting codes, abbreviated as 2D-U$_{l_m \times l_n}$EC codes.

Let $D_i = (d_{i,\rho-1}\ d_{i,\rho-2}\ \cdots\ d_{i,0})$ be a binary information block having length ρ bits. The information part of the 1D-U$_\rho$BEC code having length $k \times \rho$ bits is expressed as $(D_{k-1}\ D_{k-2}\ \cdots\ D_0)$. The parity check PC_{1D} and arithmetic residue check ARC_{1D} for $(D_{k-1}\ D_{k-2}\ \cdots\ D_0)$ are performed as follows:

$$PC_{1D}^T = \begin{pmatrix} p_{\rho-1} \\ p_{\rho-2} \\ \vdots \\ p_0 \end{pmatrix} = \begin{pmatrix} d_{k-1,\rho-1} \\ d_{k-1,\rho-2} \\ \vdots \\ d_{k-1,0} \end{pmatrix} \oplus \begin{pmatrix} d_{k-2,\rho-1} \\ d_{k-2,\rho-2} \\ \vdots \\ d_{k-2,0} \end{pmatrix} \oplus \cdots \oplus \begin{pmatrix} d_{0,\rho-1} \\ d_{0,\rho-2} \\ \vdots \\ d_{0,0} \end{pmatrix},$$

$$ARC_{1D} = \left(\sum_{i=0}^{k-1} (i+1) w(D_i) \right) \bmod (2k\rho + 1),$$

where PC_{1D}^T is a transpose of PC_{1D}, $\oplus$ denotes addition over $GF(2)$, and $w(D_i)$ is the Hamming weight of D_i. Let $\mathbf{C}$ be a balanced code [PARK90] having $|\mathbf{C}| \geq 2k\rho + 1$

TABLE 13.8 Example of Function f for $n' = 7$

i	$f(i)$	i	$f(i)$	i	$f(i)$	i	$f(i)$	i	$f(i)$	i	$f(i)$	i	$f(i)$
0	0000111	5	0010101	10	0100011	15	0100100	20	1000011	25	1001100	30	1100001
1	0001011	6	0010110	11	0100101	16	0110001	21	1000101	26	1010001	31	1100010
2	0001101	7	0011001	12	0100110	17	0110010	22	1000110	27	1010010	32	1100100
3	0001110	8	0011010	13	0101001	18	0110100	23	1001001	28	1010100	33	1101000
4	0010011	9	0011100	14	0101010	19	0111000	24	1001010	29	1011000	34	1110000

Source: *[KANE03]. © 2003 IEEE.*

codewords. That is, the Hamming weight of every codeword with length n' in $\mathbf{C}$ is $\lfloor n'/2 \rfloor$, and n' is the minimum integer satisfying $2k\rho + 1 \leq {}_{n'}C_{\lfloor n'/2 \rfloor}$, where $\lfloor x \rfloor$ is the maximum integer smaller than or equal to x. Codeword V of the 1D-U$_\rho$BEC code is defined as follows:

$$V = (D_{k-1}\ D_{k-2}\ \cdots\ D_0\ PC_{1\mathrm{D}}\ f(ARC_{1\mathrm{D}})),$$

where f is an injective mapping from $\{0, 1, \cdots, 2k\rho\}$ to $\mathbf{C}$. Table 13.8 shows an example of f mapping for $n' = 7$.

We denote the check-bit length of the 1D-U$_\rho$BEC code by $R(K, \rho)$, where K is the required information-bit length and ρ the burst error length. If K is not a multiple of ρ, then $k = \lceil K/\rho \rceil$, and the leftmost $k\rho - K$ bits are filled with 0's, where $\lceil x \rceil$ is the minimum integer greater than or equal to x.

2. Two-dimensional Unidirectional Clustered Error Correcting Codes

(1) Codeword Structure The two-dimensional codeword $\mathbf{U}$ of the 2D-U$_{l_m \times l_n}$EC code is represented by the following binary $M \times N$ matrix:

$$\mathbf{U} = \left[\begin{array}{ccc|c|ccc} u_{0,0}^{0,0} & \cdots & u_{0,0}^{0,l_n-1} & & u_{0,n-1}^{0,0} & \cdots & u_{0,n-1}^{0,l_{n-1}} \\ \vdots & & \vdots & \cdots & \vdots & & \vdots \\ u_{0,0}^{l_m-1,0} & \cdots & u_{0,0}^{l_m-1,l_n-1} & & u_{0,n-1}^{l_m-1,0} & \cdots & u_{0,n-1}^{l_m-1,l_n-1} \\ \hline & \vdots & & & & \vdots & \\ \hline u_{m-1,0}^{0,0} & \cdots & u_{m-1,0}^{0,l_n-1} & & u_{m-1,n-1}^{0,0} & \cdots & u_{m-1,n-1}^{0,l_{n-1}} \\ \vdots & & \vdots & \cdots & \vdots & & \vdots \\ u_{m-1,0}^{l_m-1,0} & \cdots & u_{m-1,0}^{l_m-1,l_n-1} & & u_{m-1,n-1}^{l_m-1,0} & \cdots & u_{m-1,n-1}^{l_m-1,0} \end{array}\right] = \left[\begin{array}{c|c|c} U_{0,0} & \cdots & U_{0,n-1} \\ \hline \vdots & & \vdots \\ \hline U_{m-1,0} & \cdots & U_{m-1,n-1} \end{array}\right],$$

where $M = ml_m, N = nl_n$, and $U_{i,j}$ is a binary $l_m \times l_n$ submatrix expressed by

$$U_{i,j} = \left[\begin{array}{ccc} u_{i,j}^{0,0} & \cdots & u_{i,j}^{0,l_n-1} \\ \vdots & & \vdots \\ u_{i,j}^{l_m-1,0} & \cdots & u_{i,j}^{l_m-1,l_n-1} \end{array}\right]_{l_m \times l_n}.$$

Note that the 2D-U$_{l_m \times l_n}$EC code is designed as being capable of correcting single unidirectional $l_m \times l_n$-clustered errors in any place, that is, errors even existing in the boundaries of the submatrices $U_{s,t}, U_{s,t+1}, U_{s+1,t}$, and $U_{s+1,t+1}$, where $0 \leq s \leq m-1$ and $0 \leq t \leq n-1$. The codeword $\mathbf{U}$ is constituted by disjoint three parts, namely the

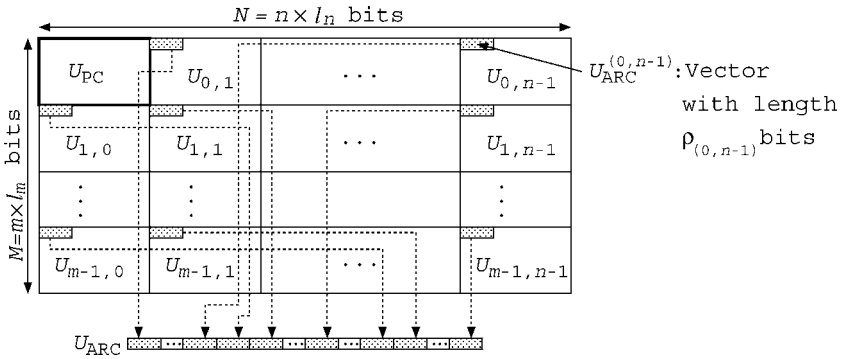


Figure 13.20 *Codeword structure.* Source: *[KANE03]. © 2003 IEEE.*

parity-check part U_{PC}, the arithmetic residue-check part U_{ARC}, and the information part U_{D}. The parity-check part U_{PC} is assigned to $U_{0,0}$, meaning $U_{\mathrm{PC}} = U_{0,0}$. The arithmetic residue-check part U_{ARC} consists of $mn - 1$ binary vectors, each of which is denoted by $U_{\mathrm{ARC}}^{(i,j)}$ where

$$(i, j) \in \Omega = \{(I, J) \,|\, 0 \le I \le m-1, 0 \le J \le n-1, (I, J) \ne (0, 0)\}.$$

As illustrated in Figure 13.20, the vector $U_{\mathrm{ARC}}^{(i,j)}$ is assigned to the left most $\rho_{(i,j)}$ bits in the first row in $U_{i,j}$. The length $\rho_{(i,j)}$ of $U_{\mathrm{ARC}}^{(i,j)}$ is specified in the following code design. The remaining bits in $\mathbf{U}$ constitute the information part U_{D}. Figure 13.20 illustrates the total codeword structure.

(2) Code Design The parity-check part U_{PC} is determined as

$$U_{\mathrm{PC}} = U_{0,0} = \begin{bmatrix} u_{0,0}^{0,0} & \cdots & u_{0,0}^{0,l_n-1} \\ \vdots & & \vdots \\ u_{0,0}^{l_m-1,0} & \cdots & u_{0,0}^{l_m-1,l_n-1} \end{bmatrix} = \sum_{(i,j)\in\Omega}^{\oplus} U_{i,j},$$

where $\sum^{\oplus}$ denotes summation of matrices over $GF(2)$ excluding $\rho_{(i,j)}$ check bits in $U_{\mathrm{ARC}}^{(i,j)}$.

The arithmetic residue-check part U_{ARC} is determined by the following procedure:

Step 1. Arithmetic residue checks ARC_V and ARC_H are written as

$$ARC_V = \left(\sum_{i=0}^{m-1} \left((i+1) \sum_{j=0}^{n-1} w(U_{i,j}) \right) \right) \bmod M_V,$$

$$ARC_H = \left(\sum_{j=0}^{n-1} \left(j \sum_{i=0}^{m-1} w(U_{i,j}) \right) \right) \bmod M_H,$$

where $M_V = 2ml_ml_n + 1, M_H = (n-1)l_ml_n + 1, U_{0,0} = U_{\mathrm{PC}}$, and $w(U_{i,j})$ is the number of 1's in $U_{i,j}$ excluding that in $U_{\mathrm{ARC}}^{(i,j)}$.

Step 2. Let $[ARC_V]_2$ and $[ARC_H]_2$ be binary representations of ARC_V and ARC_H, respectively. These binary vectors are concatenated in order to generate a vector $([ARC_V]_2, [ARC_H]_2)$ having length $K = \lceil \log_2 M_V \rceil + \lceil \log_2 M_H \rceil$ bits.

Step 3. Let $\rho (\leq l_n)$ be the minimum integer satisfying the inequality

$$\rho \times (mn - 1) \geq R(K, \rho) + K \tag{13.10}$$

Also let $\boldsymbol{0} = (0 \cdots 0)$ be a zero vector having length $k\rho - K$ bits, where $k = \lceil K/\rho \rceil$. The binary vector $(\boldsymbol{0}, [ARC_V]_2, [ARC_H]_2)$ is equally divided into k binary blocks each having length ρ bits,

$$(\underbrace{\boldsymbol{0}}_{k\rho - K}, \underbrace{[ARC_V]_2, [ARC_H]_2}_{K}) = (\underbrace{D_{k-1}, D_{k-2}, \cdots, D_0}_{k\rho}),$$

where D_i, $0 \leq i \leq k - 1$, is a binary block having length ρ bits.

Step 4. The binary vector $(D_{k-1}, D_{k-2}, \cdots, D_0)$ is encoded by 1D-U$_\rho$BEC code as shown previously, and the codeword obtained is

$$V = (D_{k-1}\ D_{k-2}\ \cdots\ D_0\ PC_{1D}\ f(ARC_{1D})).$$

Step 5. The leftmost $k\rho - K$ bits in V are removed, and the remaining part V' of V is divided into $mn - 1$ binary vectors, each of which corresponds to a vector $U_{\text{ARC}}^{(i,j)}$ where $(i, j) \in \Omega$. The resulting arithmetic residue-check part is

$$U_{\text{ARC}} = V' = (U_{\text{ARC}}^{(0,1)}, U_{\text{ARC}}^{(0,2)}, \cdots, U_{\text{ARC}}^{(0,n-1)}, U_{\text{ARC}}^{(1,0)}, U_{\text{ARC}}^{(1,1)}, \cdots, U_{\text{ARC}}^{(m-1,n-1)}),$$

where length $\rho_{(i,j)}$ of $U_{\text{ARC}}^{(i,j)}$ satisfies $\sum_{(i,j)\in\Omega} \rho_{(i,j)} = R(K, \rho) + K$ and $\rho - 1 \leq \rho_{(i,j)} \leq \rho$.

Figure 13.21 illustrates this residue-check procedure.

If the condition given by the inequality (13.10) is not satisfied for any $\rho \leq l_n$, the 2D-U$_{l_m \times l_n}$EC code cannot be designed, and then the conventional two-dimensional clustered error correcting codes [BREI98] are used.

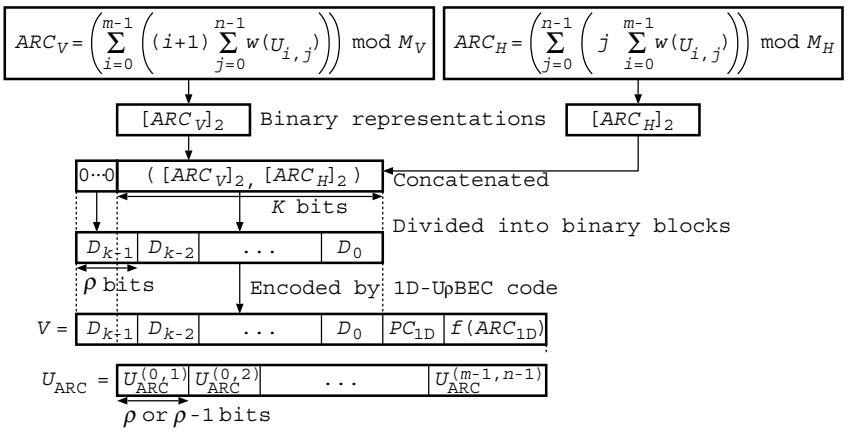


Figure 13.21 *Generation of* U_{ARC}. Source: *[KANE03]. © 2003 IEEE.*

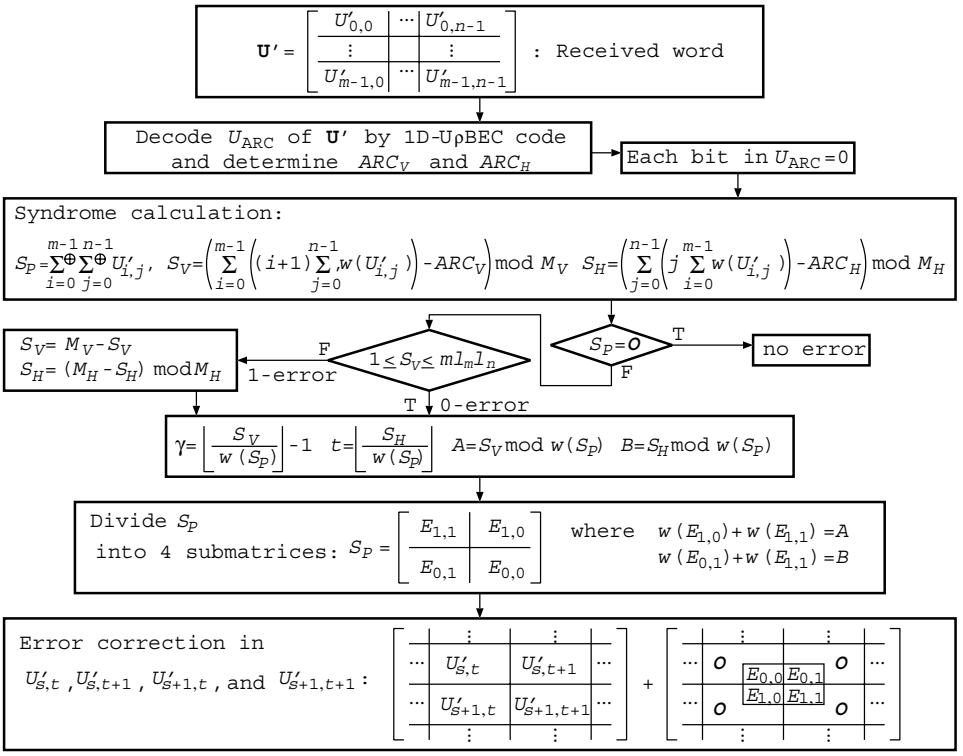


Figure 13.22 *Decoding procedure.* Source: *[KANE03]. © 2003 IEEE.*

Theorem 13.10 *Codeword* $\mathbf{U}$ *obtained by the above procedure is a codeword of 2D-*$U_{l_m \times l_n}$*EC code.*

(3) Decoding Procedure Figure 13.22 illustrates the decoding procedure of the 2D-$U_{l_m \times l_n}$EC code.

Example 13.8 [KANE03]

Encoding The objective is to design the 2D-$U_{3\times3}$EC code with code parameters $m = 3$ and $n = 4$. We have $M_V = 55, M_H = 28, K = \lceil \log_2 M_V \rceil + \lceil \log_2 M_H \rceil = 11, \rho = 2$, and $R(11, 2) + K = 9 + 11 = 20$. From the information part shown in Figure 13.23 (a), the parity-check part is determined as

$$U_{\mathrm{PC}} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

Then ARC_V and ARC_H are calculated as follows:

$$ARC_V = (1 \times 17 + 2 \times 12 + 3 \times 17) \bmod 55 = 37,$$
$$ARC_H = (0 \times 14 + 1 \times 8 + 2 \times 14 + 3 \times 10) \bmod 28 = 10.$$

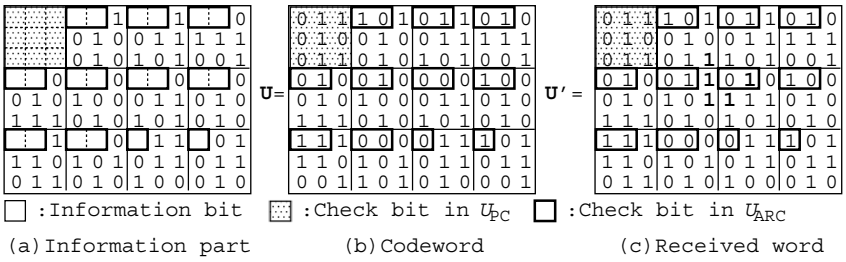


Figure 13.23 *Example code with parameters $l_m = l_m = 3$, $m = 3$, and $n = 4$.* Source: *[KANE03]. © 2003 IEEE.*

Thus U_{ARC} is determined as follows:

$$([ARC_V]_2, [ARC_H]_2) = (1\,0\,0\,1\,0\,1, 0\,1\,0\,1\,0)$$

$$(D_5, D_4, D_3, D_2, D_1, D_0) = (01, 00, 10, 10, 10, 10)$$

$$(D_5, \cdots, D_0, PC_{1\mathrm{D}}, f(ARC_{1\mathrm{D}})) = (01, 00, 10, 10, 10, 10, 01, 0110001)$$

$$U_{\mathrm{ARC}} = (10, 01, 01, 01, 01, 00, 10, 11, 00, 0, 1)$$

$$U_{\mathrm{ARC}}^{(0,1)}\ U_{\mathrm{ARC}}^{(0,2)}\ U_{\mathrm{ARC}}^{(0,3)}\ U_{\mathrm{ARC}}^{(1,0)}\ U_{\mathrm{ARC}}^{(1,1)}\ U_{\mathrm{ARC}}^{(1,2)}\ U_{\mathrm{ARC}}^{(1,3)}\ U_{\mathrm{ARC}}^{(2,0)}\ U_{\mathrm{ARC}}^{(2,1)}\ U_{\mathrm{ARC}}^{(2,2)}\ U_{\mathrm{ARC}}^{(2,3)}$$

where $(PC_{1\mathrm{D}}, f(ARC_{1\mathrm{D}}))$ is the check part of the 1D-U$_2$BEC code for the information part $(D_5\ D_4\ \cdots\ D_0)$, and $f(ARC_{1\mathrm{D}}) = f((1 \times 1 + 2 \times 1 + 3 \times 1 + 4 \times 1 + 5 \times 0 + 6 \times 1) \mod 25) = f(16) = (0110001)$ is a codeword of the balanced code given in Table 13.8. Finally, the codeword shown in Figure 13.23(b) is obtained.

Decoding Assume that a received word $\mathbf{U}'$ has a cluster of five unidirectional 0-errors, meaning five 0's are changed to 1's as shown in Figure 13.23 (c), where '**1**' denotes erroneous bit. The received word $\mathbf{U}'$ is decoded based on the decoding procedure shown in Figure 13.22. First, arithmetic residue-check part U_{ARC} in $\mathbf{U}'$ is decoded by using the 1D-U$_2$BEC code, and then an erroneous bit in U_{ARC} is corrected. From this, original $ARC_V = 37$ and $ARC_H = 10$ are obtained. Next, syndromes S_P, S_V, and S_H are determined as

$$S_P = \sum_{i=0}^{2} \oplus \sum_{j=0}^{3} \oplus U'_{i,j} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix},$$

$$S_V = \left(\sum_{i=0}^{2} \left((i+1) \sum_{j=0}^{3} w(U'_{i,j}) \right) - ARC_V \right) \bmod M_V = 7,$$

$$S_H = \left(\sum_{j=0}^{3} \left(j \sum_{i=0}^{2} w(U'_{i,j}) \right) - ARC_H \right) \bmod M_H = 5,$$

where $U'_{i,j}$ is a 3×3 submatrix of $\mathbf{U}'$:

$$\mathbf{U}' = \left[\begin{array}{c|c|c|c} U'_{0,0} & U'_{0,1} & U'_{0,2} & U'_{0,3} \\ \hline U'_{1,0} & U'_{1,1} & U'_{1,2} & U'_{1,3} \\ \hline U'_{2,0} & U'_{2,1} & U'_{2,2} & U'_{2,3} \end{array}\right].$$

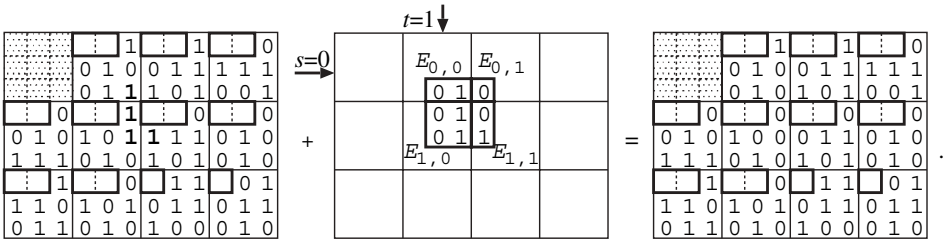
In the syndrome calculation above, all the bits in U_{ARC} are excluded. Since $1 \le S_V = 7 \le ml_m l_n = 27$, the received word $\mathbf{U}'$ has unidirectional 0-errors. Therefore s, t, A, and B are determined as follows:

$$\gamma = \left\lfloor \frac{S_V}{w(S_P)} \right\rfloor - 1 = 0,\ t = \left\lfloor \frac{S_H}{w(S_P)} \right\rfloor = 1,\ A = S_V \bmod w(S_P) = 3,\ B = S_H \bmod w(S_P) = 1.$$

The syndrome S_P is divided into four submatrices:

$$S_P = \left[\begin{array}{c|c} E_{1,1} & E_{1,0} \\ \hline E_{0,1} & E_{0,0} \end{array}\right] = \left[\begin{array}{c|cc} 0 & 0 & 1 \\ 1 & 0 & 1 \\ \hline 0 & 0 & 1 \end{array}\right].$$

Note that the submatrices of S_P satisfy $w(E_{1,0}) + w(E_{1,1}) = A = 3$ and $w(E_{0,1}) + w(E_{1,1}) = B = 1$. Finally, unidirectional 0-errors in the information part of $U'_{s,t}$, $U'_{s,t+1}$, $U'_{s+1,t}$, and $U'_{s+1,t+1}$, where $s = 0$ and $t = 1$, are corrected as follows:



(4) Evaluation The 2D-U$_{l_m \times l_n}$EC codes are evaluated in terms of the number of check bits given by $l_m \times l_n + R(K, \rho) + K$, where $R(K, \rho) + K$ is the length of U_{ARC} shown in **2**. Figure 13.24 shows the number of check bits for the 2D-U$_{l_m \times l_n}$EC codes with code parameters $M = N = 100$ and $M = N = 150$, where $l_m = l_n$. This figure also shows the cases of the existing two-dimensional $l_m \times l_n$-clustered error correcting codes [BREI98], and the QR code [JAPA02] using interleaved Reed-Solomon code. For the cases where l_m and l_n are not divisors of $M(= N)$, the 2D-U$_{l_m \times l_n}$EC code is designed by using parameters $m = n = \lceil M/l_m \rceil = \lceil N/l_n \rceil$. Also the bottom $ml_m - M$ rows and the rightmost $nl_n - N$ columns are deleted from the codeword. Note that the deleted rows and columns should not include any check bits. The number of check bits of the 2D-U$_{l_m \times l_n}$EC code is much smaller than that of the existing codes, especially for a large error size. For $M = N = 100$ and $l_m = l_n = 40$, the code requires 1,639 check bits, whereas the existing two-dimensional clustered error correcting code requires 4,800 bits and the QR code 3,812 bits.

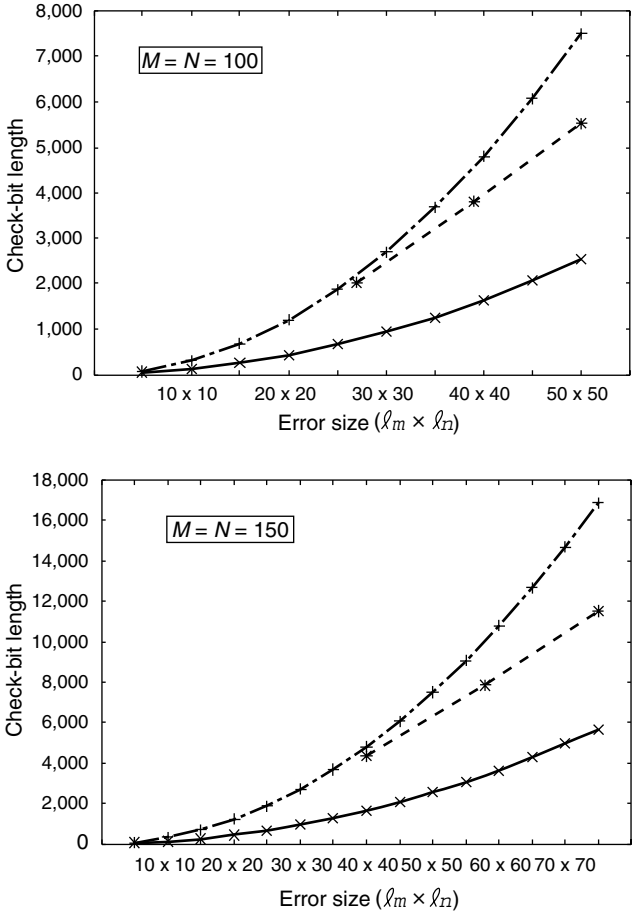


Figure 13.24 *Check-bit length of the 2D-U$_{l_m \times l_n}$EC codes.* Source: *[KANE03]. © 2003 IEEE.*

EXERCISES

13.1 Find the error directionality graph G of the confusion matrix shown in Table 13.1 with threshold error probability $T = 0.0008$.

13.2 Prove Theorem 13.2.

13.3 Find all regular elements in $R(3, 3)$.

13.4 Design the parity-check matrix $\mathbf{H}$ over $R(2, 5)$ of a systematic M-ary asymmetric symbol error correcting code with $r = 2$ check symbols.

13.5 Suppose that the confusion matrix for the set of 4-ary symbols $\mathbf{A} = \{\text{A, T, C, G}\}$ is given as follows:

	A	T	C	G
A	0.90	0.05	0	0.05
T	0.05	0.70	0.10	0.15
C	0	0.10	0.80	0.10
G	0.05	0.15	0.10	0.70

(a) With this matrix, perform the set-partitioning algorithm for $R(2,2)$, and find the mapping $F : \{\text{A, T, C, G}\} \longrightarrow R(2,2)$.

(b) Find a codeword of the 4-ary nonsystematic asymmetric symbol error correcting code having length $n = 5$ using the above-obtained mapping F, where $\mathbf{C}_1$ is a binary Hamming code with minimum distance-3 and $\mathbf{C}_2$ is a parity-check code.

(c) Find the number of codewords of the 4-ary nonsystematic asymmetric symbol error correcting code using Theorem 13.5.

13.6 Both codes VII and VIII of Table 13.4 are designed by using the Hamming code with minimum distance-3 and the parity-check code with minimum distance-2. Explain why code VII has a much lower decoded SER than code VIII.

13.7 Prove Theorem 13.6.

13.8 Design the parity-check matrix $\mathbf{H}$ over $GF(5)$ of a nonsystematic M-ary asymmetric error correcting code with deletion / insertion / adjacent-symbol-transposition error correction capabilities, whose code length $n = 5$. Prove that the obtained code has the function of deletion / insertion / adjacent-symbol-transposition error correction capabilities.

13.9 Given the error directionality graph shown in Figure 13.16, find a mapping $f : \{0, 1, \ldots, 9\} \longrightarrow GF(5)$ satisfying the condition of Definition 13.11.

13.10 Using the parity-check matrix $\mathbf{H}$ in Exercise 13.8 and the mapping f in Exercise 13.9, find the codeword of a nonsystematic 10-ary asymmetric error correcting code with deletion / insertion / adjacent-symbol-transposition error correction capabilities, where the code length $n = 5$, $\mathbf{A} = \{0, 1, \ldots, 9\}$, and $g(i) = i$ for all $i \in \mathbf{A}$.

13.11 Find the balanced code with a length of 6 bits.

13.12 Based on the specification of the QR codes, do the following:

(a) Convert "REED-SOLOMON" into a binary sequence in the alphanumeric mode.

(b) Convert "5299714" into a binary sequence in the numerical mode.

(c) Convert "5299714" into a binary sequence in the alphanumeric mode.

(d) Compare the lengths of the binary sequences obtained in (**b**) and (**c**), and give the reason why QR codes have several conversion modes.

13.13 In the numerical mode conversion of QR code, $p = 3$ digits are converted to $q = 10$ bits, and hence each numeral is expressed by $q/p = 10/3 = 3.33$ bits.

Complete the following table and confirm effectiveness of the numerical mode conversion with $p = 3$:

p	$X = 10^p$	$q = \lceil \log_2 X \rceil$	q/p
1	10	4	4
2			
3	1,000	10	3.33
4			
5			

(Answer: For $p = 2$: $X = 100$, $q = 7$, $q/p = 3.5$; for $p = 4$: $X = 10,000$, $q = 14$, $q/p = 3.5$; and for $p = 5$: $X = 100,000$, $q = 17$, $q/p = 3.4$.)

13.14 In the QR code shown in Figure 13.19, find the uncorrectable clustered error pattern.

13.15 Find the codeword of 2D-$U_{3\times3}$EC code using the function f shown in Table 13.8, where $m = n = 3$.

13.16 Given the codeword obtained in Exercise 13.15, find the decoding procedure of the received word having a unidirectional 3×3-clustered error.

13.17 Explain why 2D-$U_{l_m \times l_n}$EC codes cannot correct symmetric (bidirectional) $l_m \times l_n$-clustered errors.

REFERENCES

[BREI98] M. Breitbach, M. Bossert, V. Zyablov, and V. Sidorenko, "Array Codes Correcting a Two-Dimensional Cluster of Errors," *IEEE Trans. Info. Theory*, 44 (September 1998): 2025–2031.

[BROW73] D. A. H. Brown, "Construction of Block Error-Detection and Correction Codes to any Base," *Electron. Letts.*, 9 (June 1973): 290.

[GALL96] J. A. Gallian, "Error Detection Methods," *ACM Comput. Surveys*, 28 (September 1996): 504–517.

[IMAI77] H. Imai and S. Hirakawa, "A New Multilevel Coding Method Using Error-Correcting Codes," *IEEE Trans. Info. Theory*, IT-23 (May 1977): 371–377.

[INAB94] H. Inaba, F. Terashima, K. Wakasugi, and M. Kasahara, "A Note on Character Recognition System Using Error Correcting Codes"(in Japanese), *Trans. IEICE D-II*, J77-D-II (February 1994): 353–361.

[JAPA02] Japanese Industrial Standard, "Two Dimensional Symbol—QR Code—Basic specification," JIS X 0510:2002 Japanese Standard Association (November 2002).

[KANE03] H. Kaneko and E. Fujiwara, "Array Codes Correcting a Cluster of Unidirectional Errors for Two-Dimensional Matrix Symbols," *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems* (November 2003): 242–249.

[KANE04a] H. Kaneko and E. Fujiwara, "A Class of M-Ary Asymmetric Symbol Error Correcting Codes for Data Entry Devices," *IEEE Trans. Comput.*, C-53 (February 2004): 159–167.

[KANE04b] H. Kaneko, M. Numakami, and E. Fujiwara, "Nonsystematic M-Ary Asymmetric Error Correcting Codes Designed by Multilevel Coding Method," *Proc. IEEE Pacific Rim Int. Symp. on Dependable Computing* (March 2004): 219–226.

[KANE04c] H. Kaneko and E. Fujiwara, ''Nonsystematic *M*-Ary Asymmetric Error Correcting Codes with Deletion / Insertion / Adjacent-Symbol-Transposition Error Correction Capabilities,'' *Proc. 2004 IEEE Int. Symp. on Information Theory and Its Applications* (October 2004): 959–964.

[LEVE66] V. I. Levenstein, ''Binary Codes Capable of Correcting Deletion, Insertion, and Reversals,'' *Sov. Phys.-Dokl.*, 10 (February 1966): 707–710.

[MATR] ⟨http://www.aimglobal.org/aimstore/matrixsymbologies.htm⟩.

[NAMB01] K. Namba and E. Fujiwara, ''A Class of *m*-Ary Single-Symbol Error Correcting Codes,'' *Syst. Computs. in Japan*, 32 (June 2001): 21–28.

[NOUM93] T. Noumi, T. Matsui, I. Yamashita, T. Wakahara, and M. Yoshimuro, ''An Analysis of Substituted / Rejected Patterns in Handwritten Numeral Recognition'' (in Japanese), *Technical Report of IEICE*, PRU93-46 (September 1993): 25–32.

[PARK90] S. Park and B. Bose, ''Burst Asymmetric / Unidirectional Error Correcting / Detecting Codes,'' *Dig. 20th IEEE Int. Symp. on Fault-Tolerant Computing* (June 1990): 273–280.

[PAVL90] T. Pavlidis, J. Swartz, and Y. P. Wang, ''Fundamentals of Bar Code Information Theory,'' *IEEE Computer*, 23 (April 1990): 74–86.

[PAVL92] T. Pavlidis, J. Swartz, and Y. P. Wang, ''Information Encoding with Two-Dimensional Bar Codes,'' *IEEE Computer*, 25 (June 1992): 18–28.

[SAIT90a] Y. Saitoh and H. Imai, ''Constructions of Codes Correcting Burst Asymmetric Errors,'' *Lecture Notes in Computer Science, Applied Algebraic Algorithms and Error Correcting Codes*, 508 (1990): 59–70.

[SAIT90b] Y. Saitoh and H. Imai, ''Some Classes of Burst Asymmetric or Unidirectional Error Correcting Codes,'' *Electron. Letts.*, 26 (March 1990): 286–287.

[SAOW00] K. Saowapa, H. Kaneko, and E. Fujiwara, ''Unidirectional Byte Error Correcting Codes for *q*-Ary Data,'' *Proc. 2000 IEEE Int. Symp. on Information Theory* (June 2000): 8.

[SAOW01] K. Saowapa, H. Kaneko, and E. Fujiwara, ''*q*-Ary Asymmetric Error Locating Codes under Directional Error Model''(in Japanese), *Trans. IEICE*, J84-A (January 2001): 73–83.

[SUZU98] H. Suzuki and E. Fujiwara, ''*q*-Ary Single Substitution and Transposition Error Correcting Codes'' (in Japanese), IEICE Technical Report, FTS97-77 (February 1998).

[TANG70] D. T. Tang and V. Y. Lum, ''Error Control for Terminals with Human Operators,'' *IBM J. Res. Dev.*, 14 (July 1970): 409–416.

[TENE84] G. Tenengolts, ''Nonbinary Codes, Correcting Single Deletion or Insertion,'' *IEEE Trans. Info. Theory*, 30 (September 1984): 766–769.

[VARS73] R. R. Varshamov, ''A Class of Codes for Asymmetric Channels and a Problem from the Additive Theory of Numbers,'' *IEEE Trans. Info. Theory*, 19 (January 1973): 92–95.

CONTENTS

14

Codes for Multiple / Distributed Storage Systems

This chapter deals with the codes for multiple disk systems such as *RAID* (redundant arrays of independent disks) systems [GIBS89] [GIBS92] and for distributed storage subsystems connected by network.

Based on the discussion in Subsection 11.2.4, this chapter clarifies the requirements for the codes of RAID systems and then presents the MDS (maximum distance separable) array codes defined over certain polynomial rings that satisfy the requirements. Low-density *MDS array codes* tolerating two erased disk failures such as *EVENODD*, the *X-code*, and the *B-code* were presented in [BLAU93, 95], [XU99a, 99b]. With using circular permutation matrices, an MDS array code was designed to tolerate three erased disk failures [FENG05]. Here EVENODD and the X-code are explained more precisely. Another different coding technique that also tolerates multiple-disk failures in the disk arrays is called DATUM [ALVA02]. In this case user data and check data are de-clustered uniformly based on the layout function over the disk arrays in order to achieve high data throughput and small average response time especially for write accesses in disk failures.

The chapter also includes a disucssion of the scheme used for correcting erased data caused by multiple-disk failures in the distributed storage system, namely in the multiple-disk subsystems connected by network. The *BIBD (balanced incomplete block design) codes* [HELL94], whose parity-check matrices are designed based on the block design, are presented for tolerating erased data caused by three- or four-disk failures. The column vectors in the parity-check matrices of the codes are constructed by the *Steiner system* of block design. The extended codes provide simple and direct decoding that can recover the erased data only by simple parity calculations. The *additive codes* are also extended to enable the direct decoding for tolerating multiple-disk failures in the distributed storage systems.

Code Design for Dependable Systems: Theory and Practical Applications, by Eiji Fujiwara

14.1 MDS ARRAY CODES TOLERATING MULTIPLE-DISK FAILURES

As was mentioned in Subsection 11.2.4, single-disk failures in the RAID system are tolerated by simple parity-check codes, so theoretically a single erasure could be corrected by distance-2 codes. Recall from Subsection 2.2.4 that an erasure is an error whose location is indicated in advance by means of some error detection mechanism. For example, a pointer indicating disk failure is generated by the strong burst / byte error detecting codes in each disk memory subsystem, as discussed in Section 11.2. In this subsection, as we design practical erasure correcting codes for RAID systems, we will give careful consideration on four metrics: mean time to data loss, check disk overhead, update penalty, and group size [HELL94].

In our study of MDS array codes tolerating multiple-disk failures [BLAU93, 94, 96], [XU99a, 99b], an array code is used to express a codeword in two dimensions. That is, we consider $t \times n$ (t rows by n columns) arrays. In this model the column errors and column erasures can arise in the disk arrays. Further each disk contains m memory (striping) units, where the unit can be a bit, a byte, a sector, or the whole disk. We view such disk arrays as m layers of $t \times n$ arrays.

An optimal solution to correct τ column errors and ρ column erasures in the $t \times n$ arrays can be obtained by using $(n, n-r)$ RS codes over GF(2^t), where $r \geq 2\tau + \rho$. In this scheme each element of GF(2^t) is regarded as a t-bit column, thus transforming the (row-vector) codewords of the RS codes into $t \times n$ bits array. The optimal solution can exist because RS codes are maximum distance separable (MDS). Furthermore any pattern of τ errors and up to $r - 2\tau$ erasures can be decoded efficiently using the Berlekamp-Massey algorithm (i.e., error-erasure version; see Subsections 2.3.5 and 2.3.6), which requires τn operations of additions, multiplications, or divisions over GF(2^t) for syndrome calculation and also $O(r(\tau + \rho))$ ($\leq O(r^2)$) operations for determining the error value and the erased data.

In order to attain simple and high-speed decoding, the array codes are required to have the following properties:

1. The number of parity symbols must be one less than the minimum distance of the codes; that is, the codes are MDS.
2. The parity columns must be computable by simple XOR operations of the information columns.
3. Updating a single information bit requires updating minimum number of parity bits.

14.1.1 Theory for MDS Array Codes

1. MDS Array Codes with Code Length $n \leq p$

The new family of MDS codes presented here is defined over certain *polynomial rings* [BLAU93]. The decoding procedure is very simple compared to the RS codes with same code parameters. The simplified decoding scheme is accomplished by replacing the extension field multiplications with cyclic shifts and XOR operations of binary vectors. The codeword has a form of $t \times n$ arrays where $t = p - 1$ and $n \leq p$ for some prime p. The parity-check matrices are similar to those of RS codes; however, the field GF(2^t) is substituted by the ring of binary polynomials modulo $x^t + x^{t-1} + \cdots + x + 1$. The requirement that $t + 1$ is a prime guarantee that the resulting codes are MDS. These codes

will be explained in the following paragraphs. In the decoding of correcting multiple erasures and single errors, the only multiplications required during encoding and decoding involve ring elements of the form x^i, $i = 0,\ 1,\ 2,\ \cdots$, as one of the operands. Because of the fact that $(x-1)\ (\sum_{i=0}^{t} x^i) = x^{t+1} - 1 = x^p - 1$, the multiplication can be performed by cyclic shifts of binary vectors of length $p = t + 1$, and the implementation of any arithmetic ring operation requires only XOR operations.

The following shows the $(p-1) \times n$ array codeword of linear array code $\mathbf{C}_{p-1}$

$$\begin{aligned} \mathrm{C}(p-1,\ n,\ r) &= [\,\mathrm{C}_0 \quad \mathrm{C}_1 \quad \mathrm{C}_2 \quad \cdots \quad \mathrm{C}_{n-1}\,] \\ &= \begin{bmatrix} \mathrm{c}_{0,0} & \mathrm{c}_{0,1} & \mathrm{c}_{0,2} & \cdots & \mathrm{c}_{0,n-1} \\ \mathrm{c}_{1,0} & \mathrm{c}_{1,1} & \mathrm{c}_{1,2} & \cdots & \mathrm{c}_{1,n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ \mathrm{c}_{p-2,0} & \mathrm{c}_{p-2,1} & \mathrm{c}_{p-2,2} & \cdots & \mathrm{c}_{p-2,n-1} \end{bmatrix}, \end{aligned} \tag{14.1}$$

where C_i, $i = 0,\ 1,\ 2,\ \cdots,\ n-1$, is an i-th symbol expressed by $(p-1)$-th degree column vector, p is a prime, and $n \le p$.

For an integer a, let $|a|_p$ stand for the integer $b \in \{0,\ 1,\ 2,\ \cdots,\ p-1\}$ such that $b \equiv a \pmod{p}$. The linear array code $\mathbf{C}_{p-1}$ over $\mathrm{F} = \mathrm{GF}(q)$, where $\mathrm{GCD}(p,\ q) = 1$, is defined as a subspace of all arrays of the $\mathrm{C}(p-1,\ n,\ r)$ above that satisfy the following pr linear constraints:

$$\sum_{j=0}^{n-1} c_{|m-jz|_p,\,j} = 0, \tag{14.2}$$

where $\sum$ means summation modulo 2, $0 \le m \le p-2$, $0 \le z \le r-1$, and this has an extra all-zero row $(c_{p-1,0}\ c_{p-1,1}\ \cdots\ c_{p-1,n-1})$. Figure 14.1 shows $\mathrm{C}(p-1,\ n,\ r)$ illustrated for the case where $n = p = 5$ and $r = 3$. By this figure it is easy to verify that the array in Figure 14.2 is a codeword of array code $\mathrm{C}(4,\ 5,\ 3)$ over $\mathrm{GF}(2)$.

The $r \times n$ parity-check matrix $\mathbf{H}$ of this code over $\mathrm{R}_p(q)$, which is the ring of polynomials of degree less than $p-1$ over $\mathrm{F} = \mathrm{GF}(q)$ with multiplication taken modulo $\mathbf{M}_p(x)$, where

$$\begin{aligned} \mathbf{M}_p(x) &= (x^p - 1)/(x-1) \\ &= x^{p-1} + x^{p-2} + \cdots + x + 1 \end{aligned} \tag{14.3}$$

is defined by

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \alpha & \alpha^2 & \cdots & \alpha^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha^{r-1} & \alpha^{2(r-1)} & \cdots & \alpha^{(n-1)(r-1)} \end{bmatrix}, \tag{14.4}$$

where α is a root of $\mathbf{M}_p(x)$, and $r \le n \le p$. And also the linear code $\mathbf{C}$ of length n over $\mathrm{R}_p(q)$ is defined as

$$\begin{aligned} \mathbf{C}_{p-1} &= \{\mathrm{C} \in \big(R_p(q)\big)^n \,|\, \mathrm{C} \cdot \mathbf{H}^T = 0\}. \\ \mathrm{C} &= \mathrm{C}(p-1, n, r) = [\mathrm{C}_0\ \mathrm{C}_1\ \cdots\ \mathrm{C}_{n-1}] \end{aligned} \tag{14.5}$$

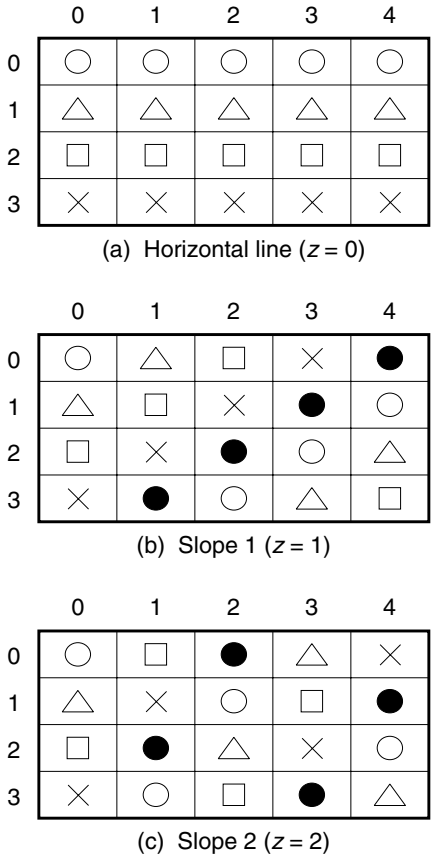


Figure 14.1 *Elements of the array code C(4, 5, 3) satisfying Eq. (14.2).* Source: *[BLAU93]. © 1993 IEEE.*

Since every r columns in $\mathbf{H}$ are linearly independent over $\mathrm{R}_p(q)$, $\mathbf{C}$ is a linear code of length n and minimum distance $r+1$, and it is MDS. The lengthened code with additional 2 columns is expressed as

$$\mathbf{H}' = \left[\begin{array}{c|cc} & 1 & 0 \\ & 0 & 0 \\ \mathbf{H} & \vdots & \vdots \\ & 0 & 0 \\ & 0 & 1 \end{array}\right]. \tag{14.6}$$

$$\mathrm{C} = \begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 2 & 0 & 1 & 1 & 1 & 1 \\ 3 & 1 & 0 & 0 & 1 & 0 \end{array}$$

Figure 14.2 *Example of a codeword of the array code C(4, 5, 3) over GF(2).* Source: *[BLAU93]. © 1993 IEEE.*

Example 14.1

The $\mathbf{H}$ matrix over $R_5(2)$ with $n = p = 5$, $r = 3$, and $q = 2$ is expressed by

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^8 \end{bmatrix},$$

where α is a root of $\mathbf{M}_5(x) = (x^5 - 1)/(x - 1) = x^4 + x^3 + x^2 + x + 1$ and is expressed by 4×4 binary companion matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

This is expressed in binary form as

$\mathbf{H} =$

I	**I**	**I**	**I**	**I**
I	. . . 1 / . . 1 . / . 1 . . / 1 . . .	. . 1 1 / . 1 . 1 / 1 . . 1 / 1 1 . .	1 1 . . / 1 . . 1 / 1 . 1 . / 1 . . .	. . . . / . . . . / . . . . / . . . .
I	. . 1 1 / . . 1 . / 1 . 1 . / . 1 1 .	1 1 . . / 1 . 1 . / 1 . . 1 / . 1 1 .	. . . 1 / . . 1 . / . 1 . . / 1 . . .	. 1 1 . / . 1 . 1 / . 1 . . / 1 1 . .

, (14.7)

where $\mathbf{I}$ is a 4×4 identity matrix. It can be verified that the word C shown in Figure 14.2 is a codeword, meaning $\mathrm{C} \cdot \mathbf{H}^T = 0$. □

2. Modified MDS Array Codes with Code Length n = p + r

A modified MDS array code with a code length larger than p, where p is a prime, that is, $n = p + r$, has been presented [BLAU96]. The codes are also defined over the ring of polynomials of degree less than or equal to $p - 2$ modulo $\mathbf{M}_p(x)$, shown in Eq. (14.3). In terms of the $(p - 1) \times n$ array $\mathrm{C} = [\mathrm{C}_0\ \mathrm{C}_1\ \cdots\ \mathrm{C}_{n-1}]$, each column C_i in the array is a binary coefficient vector of the polynomial modulo $\mathbf{M}_p(x)$. In this new model, the array has an *imaginary row* of zeros, which makes it a $p \times n$ array. A cyclic shift of a column in this array, that is, a multiplication by x modulo $x^p - 1$, can cause the bit corresponding to the last row to be nonzero. In this case, however, the arithmetic modulo $\mathbf{M}_p(x)$ forces to take the complement of the shifted column, restoring the zero in the last position. We can use the notation

$$\mathbf{C}_i(\alpha) = c_{p-2,\,i}\alpha^{p-2} + c_{p-3,\,i}\alpha^{p-3} + \cdots + c_{1,\,i}\alpha + c_{0,\,i} \tag{14.8}$$

to denote an i-th column polynomial modulo $\mathbf{M}_p(x)$. A codeword $\mathrm{C}(p, n, r)$ of a linear code $\mathbf{C}_p$ with length $n = p + r$ over the ring of binary polynomials of degree less than or

equal to $p-2$ modulo $\mathbf{M}_p(x)$ is defined by

$$\mathrm{C}(p,\ n,\ r) = \underbrace{\left[\mathrm{C}_0(\alpha)\ \mathrm{C}_1(\alpha)\ \mathrm{C}_2(\alpha)\ \cdots\ \mathrm{C}_{p-1}(\alpha)\right.}_{p}\ \underbrace{\left.\mathrm{C}_p(\alpha)\ \mathrm{C}_{p+1}(\alpha)\ \cdots\ \mathrm{C}_{p+r-1}(\alpha)\right]}_{r},$$

where

$$\mathrm{C}_{p+j}(\alpha) = \sum_{i=0}^{p-1} \alpha^{ji}\mathrm{C}_i(\alpha) \quad \text{for} \quad j = 0,\ 1,\ \cdots,\ r-1. \tag{14.9}$$

Here $\sum$ means summation modulo 2. The first p columns, $\mathrm{C}_0(\alpha)$ to $\mathrm{C}_{p-1}(\alpha)$, are information symbols, and the last r columns, $\mathrm{C}_p(\alpha)$ to $\mathrm{C}_{p+r-1}(\alpha)$, are parity symbols. Equation (14.9) specifies how the parity columns should be computed from the information columns. Note that the parity symbols depend on the information symbols, but not on each other, that is, independent. This is the major difference between the modified code and the former code defined by Eqs. (14.2) through (14.4). This means that updating a single bit in the information part would in most cases require updating a single bit in each of the parity symbols.

The parity-check matrix of the code $\mathbf{C}_p$ has a systematic form such as

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 & 0 & \cdots & 0 \\ 1 & \alpha & \cdots & \alpha^{p-1} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{r-1} & \cdots & \alpha^{(r-1)(p-1)} & 0 & 0 & \cdots & 1 \end{bmatrix}. \tag{14.10}$$

It can be proved that the codes defined by the above $\mathbf{H}$ are MDS, and the minimum distance of $\mathbf{C}_p$ is equal to $r+1$ for $r \leq 3$. The latter can be proved by using the fact that

$$\mathrm{GCD}(x^i,\ \mathbf{M}_p(x)) = \mathrm{GCD}(x^i + x^j,\ \mathbf{M}_p(x)) = 1 \qquad \text{for } i \not\equiv j \pmod{p}. \tag{14.11}$$

An example code with $p = 5$ and $r = 3$ is given by

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & 0 & 1 & 0 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^8 & 0 & 0 & 1 \end{bmatrix}.$$

This is extended by $r = 3$ check columns, compared to the previous code shown in Example 14.1. According to [BLAU96], the MDS property of the code $\mathbf{C}_p$ is clarified such that if 2 is a primitive element in GF(p), then:

1. The codes with $r = 4$ and 5 are MDS for all $p \neq 3$.
2. The codes with $r = 6$ are MDS for all $p \neq 3,\ 5,\ 13$.
3. The codes with $r = 7$ are MDS for all $p \neq 3,\ 5,\ 11,\ 13$.
4. The codes with $r = 8$ are MDS for all $p \neq 3, 5, 11, 13, 19, 29$.

Encoding and decoding of the codes, $\mathbf{C}_{p-1}$ and $\mathbf{C}_p$, are performed by simple cyclic shifts and XOR operations on the columns of the code array [BLAU93, 96].

14.1.2 Low-Density MDS Array Codes Tolerating Two Erased Disks

Based on the aforementioned background and theoretical basis of the MDS array codes, the following shows two efficient and practical schemes for tolerating double-disk failures in RAID architectures. Another efficient coding scheme, called *B-code*, not introduced here, is also a class of low-density MDS array codes expressed by a new graphic description, and constructed by using a combinatorial problem known as perfect one-factorization of complete graphs [XU99b]

1. EVENODD

The coding scheme of EVENODD requires two redundant disks, and its decoding is accomplished by simple XOR computation with independent parities [BLAU94]. The codes are $(p-1)\times(p+2)$ MDS array codes, where p is prime, with minimum distance 3 that can recover two erased disks or correct one disk failure.

Encoding Procedure There are two types of parity calculations, horizontal and diagonal parity calculations. For horizontal redundancy,

$$c_{k,p} = \sum_{i=0}^{p-1} c_{k,i} \qquad \text{for } k = 0,\ 1,\ \cdots,\ p-2. \tag{14.12}$$

For diagonal redundancy,

$$c_{k,p+1} = Q + \sum_{i=0}^{p-1} c_{|k-i|_p,i} \qquad \text{for } k = 0,\ 1,\ \cdots,\ p-2, \tag{14.13}$$

where

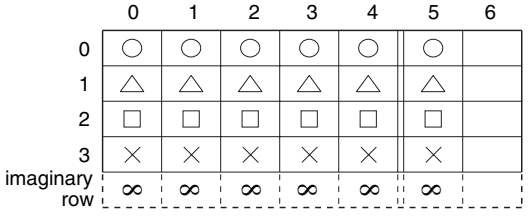
$$Q = \sum_{j=1}^{p-1} c_{p-1-j,\,j}. \tag{14.14}$$

In the equations above, $\sum$ means summation modulo 2.

The $(p-1)\times(p+2)$ array defined above can recover the information lost in any two columns; that is, the minimum distance of the code is 3. This means that any nonzero array has at least 3 nonzero columns. The key condition exists in the diagonal calculation of Q. We will see in the following example that without this diagonal calculation Q, the resulting code does not have minimum distance 3.

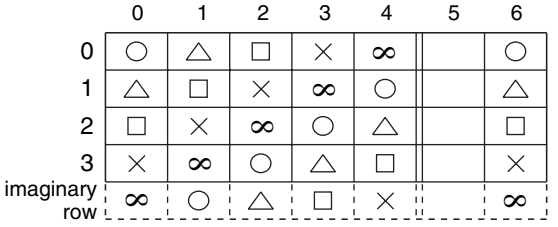
Example 14.2 [BLAU94]

Let $p = 5$, and let the symbols be denoted by $c_{i,j}$, $0 \le i \le 3$, $0 \le j \le 6$. The redundant symbols are in columns 5 and 6. The sets of symbols associated with horizontal parity are illustrated as follows:



Horizontal parity: $c_{k,5} = \sum_{i=0}^{4} c_{k,i}$, $k = 0, 1, 2, 3$

Similarly the sets of symbols associated with diagonal parity are illustrated as follows. Note that ∞ is associated with the special diagonal corresponding to Q that determines whether the diagonal parity is EVEN or ODD.



$$\text{Diagonal parity: } c_{k,6} = Q + \sum_{i=0}^{4} c_{|k-i|_5, i}\ ,\ Q = \sum_{j=0}^{4} c_{4-j, j}\ ,\ k = 0, 1, 2, 3$$

Assume that we encode the following five columns based on the Eqs. (14.12) through (14.14), where diagonal parity Q = 1. Then we obtain the two parity columns 5 and 6:

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

$Q = 1$

If we do not make the assumption that the diagonal carry either even or odd parity, the code does not have minimum distance 3. Assume that the encoding is given only by Eqs. (14.12) and (14.13), where the parameter Q is ignored in Eq. (14.13). Then the following is a codeword of weight 2:

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

If columns 1 and 5 are erased in the array above, we cannot retrieve them because the all-zero array is also a codeword. From this, the parameter Q is the key to the MDS property of the code.

Decoding Two Erasures The algorithm for correcting two erasures is presented in the following example in order to provide a rough overview of the decoding procedure.

Example 14.3 [BLAU94]

Suppose that $p = 5$, and the data in columns 0 and 2 have been erased, that is, lost in the following array:

	0	1	2	3	4	5	6
0	?	0	?	1	0	1	1
1	?	1	?	0	0	0	1
2	?	1	?	0	0	1	1
3	?	1	?	1	1	0	0

The first step is to find the parity of the diagonals. This parity is given by exclusive-OR of the bits of the two parity columns. If this is 0, then the diagonals have even parity; otherwise, they have odd parity. This is not difficult to see, and hence the reader is encouraged to undertake the proof. In the array we can see that the exclusive-OR of the bits in the two redundant columns is 1. Therefore the diagonals have odd parity, that is, $Q = 1$.

Next, the algorithm starts a recursion to retrieve the missing bits $c_{i,0}$ and $c_{i,2}$, $0 \leq i \leq 3$. The diagonal entries (3, 1), (2, 2), (1, 3), (0, 4) intersect column 2 in entry (2, 2) only. From $Q = 1$, we conclude that $c_{2,2} = 0$. So we retrieve bit (2, 0), using the horizontal parity, which is always even. We obtain $c_{2,0} = 0$. Then we consider the diagonal going through entry (2, 0), which consists of the entries (2, 0), (1, 1), (0, 2), (3, 4), (2, 6). The only bit missing is in entry (0, 2), and we can conclude that $c_{0,2} = 0$. Again, using the horizontal parity, we find that $c_{0,0} = 0$. Now, using the diagonal through (0, 0), we obtain $c_{3,2} = 0$, which implies, by the horizontal parity, that $c_{3,0} = 1$. Using the diagonal through (3, 0), we obtain $c_{1,2} = 0$, which finally implies that $c_{1,0} = 1$. The final reconstructed array is illustrated below:

	0	1	2	3	4	5	6
0	**0**	0	**0**	1	0	1	1
1	**1**	1	**0**	0	0	0	1
2	**0**	1	**0**	0	0	1	1
3	**1**	1	**0**	1	1	0	0

recovered columns

We could give a decoding that corrects one error (i.e., only one column has failed but its location is unknown). This is not the model in RAID architectures, where disk failures are catastrophic events in which pointers identify the failed disks. (The reader is encouraged to reconstruct the erroneous example array with one failed column in Exercises 14.6.

Algebraic Description of EVENODD In the array of $(p-1) \times (p+2)$, each column in the array is a polynomial modulo $\mathbf{M}_p(x) = (x^p - 1)/(x-1) = x^{p-1} + x^{p-2} + \cdots + x + 1$. It is convenient to assume that the array has an imaginary row of zeros that makes it a $p \times (p+2)$ array. A cyclic shift of a column in such an array can cause the bit corresponding to the last row to be nonzero. However, if this is the case, the arithmetic modulo $\mathbf{M}_p(x)$ forces to take the complement of the shifted column, restoring the zero in the last position. As was mentioned earlier, we will use Eq. (14.8) to denote a polynomial modulo $\mathbf{M}_p(x)$. The codeword for EVENODD is defined as follows:

$$\left.\begin{array}{l} [\mathrm{C}_0(\alpha)\ \mathrm{C}_1(\alpha)\ \cdots\ \mathrm{C}_{p-1}(\alpha)\ \mathrm{C}_p(\alpha)\ \mathrm{C}_{p+1}(\alpha)],\ \text{where } \alpha \text{ is a root of } \mathbf{M}_p(x), \\ \mathrm{C}_p(\alpha) = \displaystyle\sum_{i=0}^{p-1} \mathrm{C}_i(\alpha),\ \text{and } \mathrm{C}_{p+1}(\alpha) = \sum_{i=0}^{p-1} \alpha^i \mathrm{C}_i(\alpha). \end{array}\right\}. \tag{14.15}$$

Note that the parameter Q, defined in Eq. (14.14) and included in Eq. (14.13), essentially renders Eq. (14.13) to be the sum of cyclic shifts modulo $\mathbf{M}_p(x)$ rather than ordinary cyclic shifts.

The following is a parity-check matrix for EVENODD:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 & 0 \\ 1 & \alpha & \cdots & \alpha^{p-1} & 0 & 1 \end{bmatrix}. \tag{14.16}$$

For $p = 5$, for example, α can be expressed by a 4×4 binary matrix defined by $\mathbf{M}_5(x) = x^4 + x^3 + x^2 + x + 1$:

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Note that the parity symbols $\mathrm{C}_p(\alpha)$ and $\mathrm{C}_{p+1}(\alpha)$ depend on the information symbols but not on each other. It is easy to see from the parity-check matrix shown in Eq. (14.16) that the minimum distance of EVENODD is 3, which gives alternative proof to the basic MDS property of the code.

2. X-Codes

A new class of MDS array codes, called X-codes, has size $n \times n$, where n is a prime [XU99a]. The X-codes are of minimum column distance 3; that is, they can correct either two column erasures or one column error. The key novelty in X-code is its simple geometrical construction that achieves optimal complexity for the encoding or the update, whereby a change of any single information bit affects exactly two parity bits. The key idea exists in the code construction such that all parity symbols are placed in rows rather than columns.

In X-codes, information symbols are placed in an array of size $(n-2) \times n$. Parity symbols are constructed from the information symbols along several parity-check lines or diagonals of some slopes with addition operations. But the parity symbols are placed in two additional rows. So the coded array is of size $n \times n$, with the first $n-2$ rows containing information symbols, and the last two rows containing parity symbols. Note that information symbols and parity symbols are mixed in each column of the array.

Encoding Procedure Let $\mathrm{c}_{i,j}$ be the symbol located at the i-th row and the j-th column. The parity symbols of X-codes are constructed according to the following calculations:

$$\left.\begin{aligned} \mathrm{c}_{n-2,i} &= \sum_{k=0}^{n-3} \mathrm{c}_{k,|i+k+2|_n} \\ \mathrm{c}_{n-1,i} &= \sum_{k=0}^{n-3} \mathrm{c}_{k,|i-k-2|_n} \end{aligned}\right\}, \tag{14.17}$$

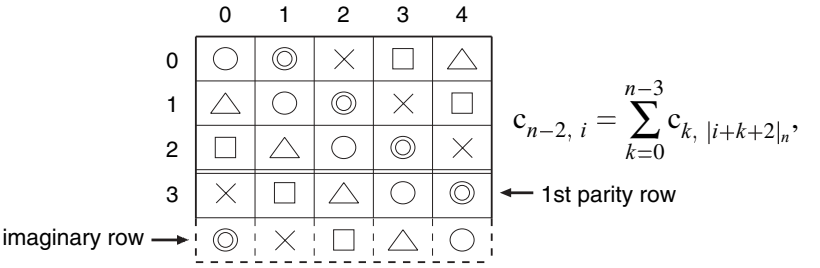
where $i = 0,\ 1,\ \cdots,\ n-1$, and $|x|_n = x \bmod n$. Geometrically the two parity rows are just the checksums along diagonals of slopes $+1$ and -1, respectively.

From this construction it is easy to see that the two parity rows are obtained independently; that is, each information symbol affects exactly one parity symbol in each parity row. All parity symbols depend only on information symbols, and therefore

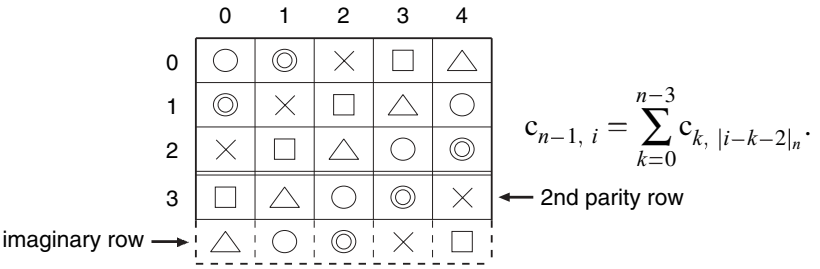
updating one information symbol results in updating only two parity symbols. Thus the X-code has the optimal encoding property, that is, the optimal updating property. Note that the X-code is a cyclic code in terms of columns; that is, the cyclically shifting columns of a codeword of the X-code results in another codeword of the X-code. Also it can be proved that the X-code has column distance 3; that is, it is MDS if and only if n is a prime number. The reader is urged to attempt a proof of this.

Example 14.4

The following arrays show an X-code for $n = 5$:

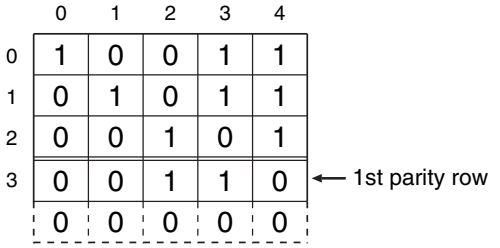


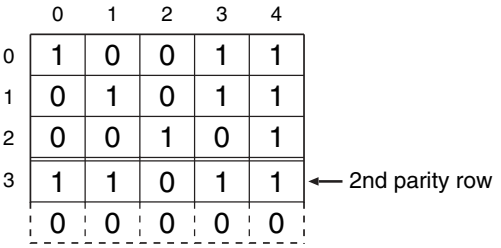
$$c_{n-2,\,i} = \sum_{k=0}^{n-3} c_{k,\,|i+k+2|_n},$$



$$c_{n-1,\,i} = \sum_{k=0}^{n-3} c_{k,\,|i-k-2|_n}.$$

The example codeword for $n = 5$ is as follows:





The parity-check matrix of this example code is written as

$$\mathbf{H} = \left[\begin{array}{c|c|c|c|c} \mathbf{H}_0 & \mathbf{T}\cdot\mathbf{H}_0 & \mathbf{T}^2\cdot\mathbf{H}_0 & \mathbf{T}^3\cdot\mathbf{H}_0 & \mathbf{T}^4\cdot\mathbf{H}_0 \\ \mathbf{H}_1 & \mathbf{T}\cdot\mathbf{H}_1 & \mathbf{T}^2\cdot\mathbf{H}_1 & \mathbf{T}^3\cdot\mathbf{H}_1 & \mathbf{T}^4\cdot\mathbf{H}_1 \end{array}\right]$$

$$= \left[\begin{array}{c|c|c|c|c} 00010 & 00000 & 10000 & 01000 & 00100 \\ 00100 & 00010 & 00000 & 10000 & 01000 \\ 01000 & 00100 & 00010 & 00000 & 10000 \\ 10000 & 01000 & 00100 & 00010 & 00000 \\ 00000 & 10000 & 01000 & 00100 & 00010 \\ \hline 00001 & 00100 & 01000 & 10000 & 00000 \\ 00000 & 00001 & 00100 & 01000 & 10000 \\ 10000 & 00000 & 00001 & 00100 & 01000 \\ 01000 & 10000 & 00000 & 00001 & 00100 \\ 00100 & 01000 & 10000 & 00000 & 00001 \end{array}\right]$$

$$\mathbf{H}_0 = \begin{bmatrix} 00010 \\ 00100 \\ 01000 \\ 10000 \\ 00000 \end{bmatrix} \quad \mathbf{H}_1 = \begin{bmatrix} 00001 \\ 00000 \\ 10000 \\ 01000 \\ 00100 \end{bmatrix}$$

$$\mathbf{T} = \begin{bmatrix} 00001 \\ 10000 \\ 01000 \\ 00100 \\ 00010 \end{bmatrix}$$

It can be easily verified that the example codeword,

$$\mathrm{C} = (\underbrace{10001}_{\text{0-th column}} \quad \underbrace{01001}_{\text{1-st column}} \quad \underbrace{00110}_{\text{2-nd column}} \quad \underbrace{11011}_{\text{3-rd column}} \quad \underbrace{11101}_{\text{4-th column}}),$$

satisfies $\mathrm{C}\cdot\mathbf{H}^T = 0$. □

Decoding Procedure

1. Correcting Two Erasures. If two columns are erased, then the basic unknown symbols of the two columns are information symbols. So the number of unknown symbols is $2(n-2)$. On the other hand, in the remaining array, there are $2(n-2)$ parity symbols related to all the $2(n-2)$ unknown symbols. Hence correction of two erasures is only a problem of solving $2(n-2)$ unknowns from the $2(n-2)$ linear equations. Since the X-code is of distance 3, it can correct two erasures. Thus the $2(n-2)$ linear equations must be linearly independent, meaning the linear equations are solvable. Note that since each parity symbol cannot be affected by more than one information symbol in the same column, each equation has at most two unknown symbols. Some equations have only one unknown symbol. The equations each having one unknown symbol can be solved, and hence these erased symbols can be corrected. The corrected symbols induce other unknown symbols to be solved. The process proceeds recurrently to solve all unknown symbols.

Suppose that the i-th and j-th columns are erased, where $0 \le i < j \le n-1$. Each diagonal traverses only $n-1$ columns. If a diagonal crosses a column at the last row, no symbols of that column are included in this diagonal. This determines the position of the parity symbol related to only one information symbol in these two erased columns, and thus this symbol can be immediately recovered by a simple checksum along this diagonal. With this symbol as the starting point, we have a decoding chain. Together with the remaining one (if $j-i=1$) decoding chain, all unknown symbols can be recovered.

2. Correcting One Error. In order to correct one error, it is necessary to locate the error position. This can be done by computing two syndrome vectors corresponding to the two parity rows. Since the error is a column error, it is natural to compute the syndromes with

respect to columns. Once the error location is determined, the correct value can be computed along the diagonals of either slope.

Suppose $R = [r_{i,j}]$, where $0 \le i, j \le n-1$, is the error corrupted array with one corrupted column. Then we compute two syndromes S_0 and S_1 as follows:

$$S_0[i] = \sum_{k=0}^{n-1} r_{i+k,k},$$

$$S_1[i] = \sum_{k=0}^{n-1} r_{i-k,k},$$

where all subindexes are mod n.

It is easy to see that the two syndromes are respectively the column checksums along the diagonals of slope $+1$ and -1, and they should be all zeros if there is no error in the array R. If there is one (column) error in the array R, then the two syndromes are just the cyclic-shifted version of the error vector with respect to the position of the error column, thus its location can be determined simply by the cyclic equivalence test, which tests if two vectors are equal after cyclic shift of one vector.

Example 14.5 Syndrome Computation for 5×5 X-code [XU99a]

Suppose that the third column of the X-code is an erroneous column. Then the two syndrome vectors, S_0 and S_1, and their corresponding error arrays are as follows:

					S_0
0	0	0	e_0	0	e_3
0	0	0	e_1	0	0
0	0	0	e_2	0	e_0
0	0	0	e_3	0	e_1
0	0	0	0	0	e_2

					S_1
0	0	0	e_0	0	e_2
0	0	0	e_1	0	e_4
0	0	0	e_2	0	0
0	0	0	e_4	0	e_0
0	0	0	0	0	e_1

Note that these two syndromes are actually just the original error column vector (cyclically) shifted by two symbols with different directions. When they are shifted back, then they only differ in at most one position. So the number of the positions shifted gives the location of the error column. □

14.2 CODES FOR DISTRIBUTED STORAGE SYSTEMS

14.2.1 Models of Distributed Storage Systems and Code Conditions

System Models A large-scale distributed storage system consists of a large number of disk subsystems connected by networks. In this section we show how the scheme is used to

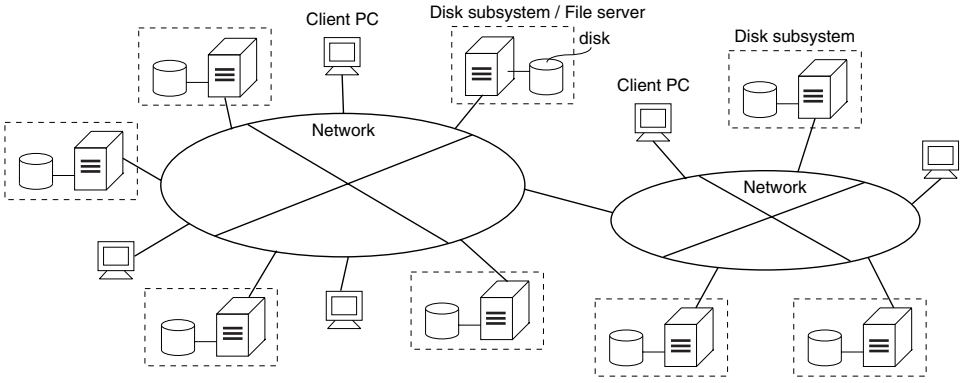
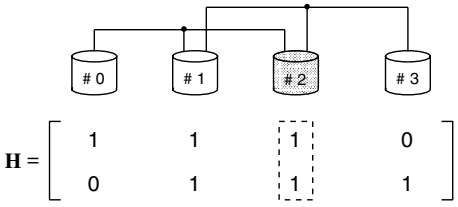


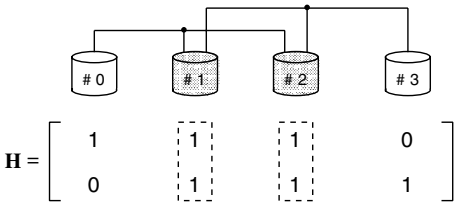
Figure 14.3 *Models of the distributed storage system.*

recover destructed user data due to disk memory failures. Figure 14.3 shows an example of a distributed system in which client PCs and disk subsystems are connected via networks. We consider two models of the user data in distributed storage systems: one is to divide the user file data and to store these divided ones in several disk subsystems in a parallel manner, and the other is to store the whole user data basically in one disk. RAID systems are typical of the former model. In both models the disk subsystems are grouped and their data are encoded by parity-based codes in order to protect and recover the data from disk failure. We will assume that the disk controller in each disk subsystem has information on the groups, that is on which other disks are parity encoded in the group.

In order to express the encoded disk group, we use the parity-check matrix with n columns, each corresponding to one disk subsystem. The i-th row in the matrix expresses the i-th group of the disks, where '1' in the j-th bit of the row expresses the disk $\#j$ in the i-th group. Figure 14.4 gives a simple example of the matrix, where disks $\#0$, $\#1$, and $\#2$



(a) Recovery of data in failed disk #2



(b) Nonrecovery of data in failed disks #1 and #2

Figure 14.4 *Parity-check matrix and corresponding disk groups.*

in the first row are grouped and parity encoded, and also disks #1, #2, and #3 in the second row are grouped and parity encoded. In Figure 14.4(a), the data in the failed disk #2 can be recovered by exclusive-ORing the data in other disks #0 and #1, or disks #1 and #3. On the other hand, in Figure 14.4(b), the data in the failed disks #1 and #2 cannot be recovered because every parity group includes erased data of these two failed disks. From this simple example, some conditions of matrix design are required in order to recover the erased data directly.

In the storage model these are four metrics for redundancy in the distributed system that are important. As was mentioned in Subsection 11.2.4, these metrics are mean time to data loss, check disk overhead, update penalty, and group size.

Code Conditions for Direct Decoding for t Erased Disk Data It can be easily understood that the parity-check matrix with minimum Hamming distance 3 can correct erased data caused by two failed disks by the simple decoding. But how about the matrix with a minimum Hamming distance 4? Does the simple decoding always correct the erased data due to three failed disks? Consider the following example of the parity-check matrix of a distance-4 modified Hamming SEC-DED code:

$$\mathbf{H} = \left[\begin{array}{cccccccc:ccc:ccccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \end{array}\right].$$

If three disks corresponding to the three columns enclosed by the broken line in the matrix above are failed, then we cannot correct the erased data immediately by simple decoding. We can, however, correct the erased data by solving three simultaneous equations. If we have another server in the system that supervises the total disk subsystems and hence solves these equations, then these erased data can be corrected. We assume, however, that there exists no such server in the system, and therefore we adopt the model to correct the erased data simply and directly, and not by solving the equations. This simple decoding of erased data is called here a *direct decoding*.

We consider the error recovery from the erased data caused by t-disk failures in the distributed storage systems that satisfies the above-defined direct decoding. In this model the following theorem determines the condition of the code matrix to recover the data caused by t-disk failures [HELL94].

Theorem 14.1 *Let an $r \times n$ binary matrix $\boldsymbol{H} = [\, \boldsymbol{P} \mid \boldsymbol{I} \,]$ be a parity-check matrix of the code with minimum Hamming distance $t+1$, where $\boldsymbol{I}$ is an $r \times r$ identity matrix and every $k(= n - r)$ distinct columns of $\boldsymbol{P}$ has weight t (≥ 3), and every pair of two columns in any t columns in $\boldsymbol{P}$ has 1's in at most one same row position. Then the code defined by the matrix $\boldsymbol{H}$ corrects t erasures by the direct decoding.*

Proof This theorem can be proved by discussing on any t distinct columns in the matrix $\mathbf{P}$ corresponding to the erased t disks. This is shown in Figure 14.5 where we have one column vector u_i and the other $t-1$ column vectors each with t 1's. Let these $t-1$ columns be $u_{i+1}, u_{i+2}, \ldots, u_{i+t-1}$, where every pair of two columns including u_i has 1's in at most one same row position. In these t columns it can be understood that there exists one

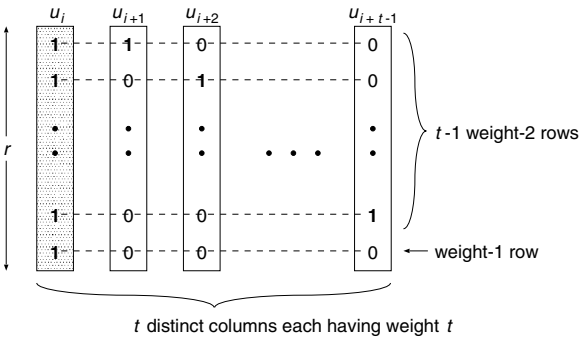


Figure 14.5 *Example array of t column vectors in* ***P****.*

row with weight one in at most one row place. We assume that every row having 1 in u_i has weight larger than or equal to two in these t columns. Then we must have at least one extra column (i.e., t columns) in addition to u_i because every pair of two columns including u_i has 1's in at most one same row position. This contradicts the present case of total t columns including u_i. In these t columns the failed disk that corresponds to u_i can be recovered directly by exclusive-ORing the data of the nonfailed disks in the parity group corresponding to the row with weight one. It is apparent from Figure 14.5 that any t columns of **H** are linearly independent, and therefore the code can correct t erasures. Also there exists at least one row with weight one. These satisfy the conditions of the direct decoding for t erased disk data. Hence, in general, the erased data corresponding to the i-th failed disk can be recovered by exclusive-ORing the correct data of the other disks corresponding to the columns outside these t columns. This recovery of data leads to the recovery of the remaining $t - 1$ erased data in a chain using the recovered disk data. Even if the matrix **I** is appended to **P**, it is apparent that the discussion above also holds for columns in **I** and **P** as well as for weight one columns in **I**. Q.E.D.

14.2.2 BIBD Codes

The matrix **P** shown in Theorem 14.1 can be designed by using the *Steiner system* in a block design [HELL94]. First, let us define the *block design*, meaning *BIBD*.

Definition 14.1 (Balanced Incomplete Block Design, BIBD) [HALL86] A *balanced incomplete block design*, or *BIBD*, is an arrangement of r distinct objects into k blocks satisfying the following conditions:

1. Each block contains exactly t distinct objects.
2. Each object occurs in exactly m different blocks.
3. Every pair of distinct objects a_i, and a_j occurs together in exactly λ blocks.

□

The four parameters k, r, t, m given above can be translated into the parameters of the parity-check matrix **H** in Theorem 14.1 as follows:

k: number of columns in **P** which means the number of information disks.

r: number of rows in **P** which means the number of check disks.

t: weight of column vector in **P** which means the number of erased disks being corrected.

m: weight of row vector in **P**.

$k + r = n$: code length which means the total number of disks.

The words, block and object, in the definition of BIBD are also translated here as follows:

Block: Column vector in **P**.

Object: Binary element 1 at the designated place in a column vector.

Every pair of distinct objects a_i and a_j together in λ blocks: Every pair of elements 1's at the i-th and the j-th row places existing together in λ columns, where $i \neq j$.

Theorem 14.2 [HALL86] *The block design BIBD satisfies the following two elementary relations on the five parameters*:

$$\left.\begin{aligned} kt &= rm \\ m(t-1) &= \lambda(r-1). \end{aligned}\right\} \tag{14.18}$$

The design with $\lambda = 1$ is called a *Steiner system*, and it satisfies Theorem 14.1 because of the condition 3 of Definition 14.1. For $\lambda = 1$, it is important that the relations (14.18) lead to the following:

$$r \equiv 1,\ 3 \pmod 6 \qquad \text{for } t = 3, \tag{14.19}$$

$$r \equiv 1,\ 4 \pmod{12} \qquad \text{for } t = 4. \tag{14.20}$$

The relations (14.19) and (14.20) are important for designing a matrix **H** that can correct three or four erased disk data in this distributed storage systems. The following theorems show how to design a Steiner system that satisfies the relations.

Theorem 14.3 [HALL86] *With the additive group of residues modulo $s = 2q + 1$, the blocks*

$$\begin{array}{lll} [1_1,\ 2q_1,\ 0_2], & \cdots, [i_1,\ (2q+1-i)_1,\ 0_2], & \cdots, [q_1,\ (q+1)_1,\ 0_2] \\ [1_2,\ 2q_2,\ 0_3], & \cdots, [i_2,\ (2q+1-i)_2,\ 0_3], & \cdots, [q_2,\ (q+1)_2,\ 0_3] \\ [1_3,\ 2q_3,\ 0_1], & \cdots, [i_3,\ (2q+1-i)_3,\ 0_1], & \cdots, [q_3,\ (q+1)_3,\ 0_1] \\ & [0_1,\ 0_2,\ 0_3] & \end{array}$$

form a base for a design with $r = 6q + 3$, $k = (3q + 1)(2q + 1)$, $m = 3q + 1$, $t = 3$, and $\lambda = 1$, where x_y or $(x)_y$ means $(y - 1)s + x$.

The remaining blocks are obtained by adding 1, 2, $\cdots$, $s - 1$, to each base block.

Example 14.6

For $q = 1$, $s = 3$, $r = 9$, $k = 12$, $m = 4$, and $t = 3$, we get the following 4 base blocks and remaining 8 blocks:

$$\begin{array}{llll:l} [1_1,\ 2_1,\ 0_2] & [1_2,\ 2_2,\ 0_3] & [1_3,\ 2_3,\ 0_1] & [0_1,\ 0_2,\ 0_3] : & \text{base blocks} \\ [2_1,\ 0_1,\ 1_2] & [2_2,\ 0_2,\ 1_3] & [2_3,\ 0_3,\ 1_1] & [1_1,\ 1_2,\ 1_3] : & +1 \\ [0_1,\ 1_1,\ 2_2] & [0_2,\ 1_2,\ 2_3] & [0_3,\ 1_3,\ 2_1] & [2_1,\ 2_2,\ 2_3] : & +2 \end{array}$$

These are transformed to the following:

$$\begin{array}{llll:l} B_0 : [1,2,3], & B_3 : [4,5,6], & B_6 : [7,8,0], & B_9 : [0,3,6] & \cdots \text{ base blocks} \\ B_1 : [2,0,4], & B_4 : [5,3,7], & B_7 : [8,6,1], & B_{10} : [1,4,7] & \cdots +1 \\ B_2 : [0,1,5], & B_5 : [3,4,8], & B_8 : [6,7,2], & B_{11} : [2,5,8] & \cdots +2 \end{array}$$

Numbers, $x,\ y$ and z in $B_i : [x,\ y,\ z]$ express the elements 1's positions, namely at the x-th, y-th, and z-th row positions in the i-th column. Hence we have the following 9×12 *incidence matrix* $\mathbf{P}$:

$$\mathbf{P} = \begin{array}{c|cccccccccccc|c} & B_0 & B_1 & B_2 & B_3 & B_4 & B_5 & B_6 & B_7 & B_8 & B_9 & B_{10} & B_{11} & \\ \hline & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 2 \\ \hline & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 3 \\ & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 4 \\ & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 5 \\ \hline & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 6 \\ & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 7 \\ & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 8 \end{array}$$

This matrix satisfies the previous code conditions for the direct decoding for 3 erased disks data, and will be used in the extended BIBD codes in Subsection 14.2.4.

Theorem 14.4 [HALL86] *Suppose* $r = 6q + 3 = 3s$, *where* $s = 2q + 1 \not\equiv 0 \pmod 3$. *Let us determine unordered pairs* $(a,\ b)$ *modulo* $3s$ *by the conditions* $a \equiv b \equiv 1 \pmod 3$, $a + b \equiv 0 \pmod s$, $a,\ b \not\equiv 0 \pmod s$. *Then the base blocks*

$$[0,\ a,\ b] \quad (mod\, 3s)$$

and the single $[0,\ s,\ 2s]$ *of period* s *yield a design with* $r = 6q + 3$, $k = (2q + 1)\,(3q + 1)$, $m = 3q + 1$, $t = 3$, *and* $\lambda = 1$.

The remaining blocks of Theorem 14.4 are obtained by adding $1, 2, \cdots, 3s - 1$ to each base block $[0, a, b]$ (mod $3s$), and also by adding $1,\ 2,\ \cdots,\ s - 1$ to the base block $[0,\ s,\ 2s]$.

Example 14.7

For $q = 2, s = 5, r = 15, k = 35, m = 7$, and $t = 3$, we have $(a,\ b) = (1,\ 4),\ (7,\ 13)$, and $(s,\ 2s) = (5,\ 10)$. Therefore we have the following 35 blocks:

[0,	1,	4],	[0,	7,	13],	[0,	5,	10] :	base blocks
[1,	2,	5],	[1,	8,	14],	[1,	6,	11] :	+1
[2,	3,	6],	[2,	9,	0],	[2,	7,	12] :	+2
[3,	4,	7],	[3,	10,	1],	[3,	8,	13] :	+3
[4,	5,	8],	[4,	11,	2],	[4,	9,	14] :	+4
[5,	6,	9],	[5,	12,	3] :				+5
[6,	7,	10],	[6,	13,	4] :				+6
[7,	8,	11],	[7,	14,	5] :				+7
[8,	9,	12],	[8,	0,	6] :				+8
[9,	10,	13],	[9,	1,	7] :				+9
[10,	11,	14],	[10,	2,	8] :				+10
[11,	12,	0],	[11,	3,	9] :				+11
[12,	13,	1],	[12,	4,	10]:				+12
[13,	14,	2],	[13,	5,	11]:				+13
[14,	0,	3],	[14,	6,	12]:				+14

The incidence matrix **P** is shown in Figure 14.6.

For $r = 6q + 1$ and $r = 12q + 1$, we have another set of blocks based on the following theorems:

Theorem 14.5 [HALL86] *Let $r = 6q + 1 = p^n$ where p is a prime. In the field $GF(p^n)$, let x be a primitive element. Then the blocks*

$$[x^0,\ x^{2q}, x^{4q}],\ \cdots,\ [x^i,\ x^{2q+i},\ x^{4q+i}],\ \cdots,\ [x^{q-1},\ x^{3q-1},\ x^{5q-1}]$$

are base blocks of the additive group of $GF(p^n)$ with $r = 6q + 1$, $k = 6q^2 + q$, $m = 3q$, $t = 3$, and $\lambda = 1$.

P =

1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0	0
1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0	1
0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0	2
0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0	3
1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1	4
0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 0	5
0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0	6
0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0	7
0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0	8
0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 1	9
0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0	10
0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 0	11
0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0	12
0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0	13
0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 1	14

Figure 14.6 *Incidence matrix **P** with $t = 3$, $m = 7$, $r = 15$, and $k = 35$.*

Theorem 14.6 [HALL86] *If $r = p^n = 12q + 1$ where p is a prime, and if x is a primitive element of $GF(p^n)$ such that $x^{4q} - 1 = x^w$ where w is odd, the blocks*

$$[0,\ x^0,\ x^{4q},\ x^{8q}],\ \cdots,\ [0,\ x^{2i},\ x^{2i+4q},\ x^{2i+8q}],\ \cdots,\ [0,\ x^{2q-2},\ x^{6q-2},\ x^{10q-2}]$$

are a base with respect to the additive group of $GF(p^n)$ of a design with $r = 12q + 1$, $k = q(12q + 1)$, $m = 4q$, $t = 4$, and $\lambda = 1$.

14.2.3 Additive Codes

A new type of code capable of correcting t erasures is called an *additive-t* code. The additive-t code satisfies the code conditions indicated in Theorem 14.1. An additive-3 code satisfies the practical case of Theorem 14.1 for $t = 3$. The existing work by [HELL94] treats the codes with some restricted parameters; the next subsection discusses this more general class of additive-3 codes.

Lemma 14.1 *The number of solutions of an integer x that satisfy $3x \equiv 1 \pmod{r}$, where $0 \le x \le r - 1$ and r is an integer ($r \ge 3$), is denoted as β_1 and expressed as*

$$\beta_1 = \begin{cases} 0 & \text{for } r = \text{multiple of } 3, \\ 1 & \text{for } r = \text{integer other than multiple of } 3. \end{cases}$$

Proof For some integer m, the relation $3x \equiv 1 \pmod{r}$ is expressed as

$$3x = 1 + mr,$$

and then this is also expressed as

$$3x - (1 + mr) = 0 \qquad \text{for} \quad 0 \le x \le r - 1.$$

Therefore inequality $0 \le x \le r - 1$ can be expressed by $0 \le 1 + mr \le 3(r - 1)$, which comes to $m = \{0,\ 1,\ 2\}$.

Case 1. For $r =$ multiple of 3 (i.e., $r = 3n$):

$$3x - (1 + mr) = 3x - (1 + 3nm) \ne 0.$$

Hence there exist no integer solutions of x, meaning $\beta_1 = 0$.

Case 2. For $r = 3n + 1$:

$$3x - (1 + mr) = 3x - (1 + 3nm + m).$$

In this case, x has a solution $2n + 1$ only when $1 + m$ is 3; that is, $m = 2$, meaning $\beta_1 = 1$.

Case 3. For $r = 3n + 2$:

$$3x - (1 + mr) = 3x - (1 + 3nm + 2m).$$

In this case, x has a solution $n + 1$ only when $1 + 2m$ is 3, that is, $m = 1$, meaning $\beta_1 = 1$. Q.E.D.

Lemma 14.2 *For a given integer r (≥ 3), the number of cases each having three distinct integers y, z, and w, 0 ≤ y, z, w ≤ r−1, that satisfy*

$$y + z + w \equiv 1 \pmod{r} \tag{14.21}$$

is $r^2 - 3r + 2\beta_1$.

Proof It is apparent that the number of combinations of all three integers that satisfy Eq. (14.21) is r^2. The cases with two equal integers and with all three equal integers can be deducted from the r^2 cases. The number of the former case is obtained by ${}_{r-\beta_1}C_1 \times 3 = 3(r - \beta_1)$. The number of the latter case is β_1 by Lemma 14.1. Therefore the number of having three distinct integers that satisfy Eq. (14.21) is expressed as $r^2 - 3(r - \beta_1) - \beta_1 = r^2 - 3r + 2\beta_1$. Q.E.D.

Theorem 14.7 *The additive-3 code* $\boldsymbol{C}$ *defined by the parity-check matrix* $\boldsymbol{H} = [\, \boldsymbol{P}_A \mid \boldsymbol{I} \,]$, *where* $\boldsymbol{I}$ *is an* $r \times r$ *identity matrix and* $\boldsymbol{P}_A$ *is an* $r \times k$ *binary incidence matrix with distinct weight-3 columns, can correct 3 erasures by the direct decoding. Here let integers y, z, and w (0 ≤ y, z, w ≤ r−1) be row numbers of column in* $\boldsymbol{P}_A$ *that satisfy* $y + z + w \equiv 1$ *(mod r). Then the parameter k is determined by* $(r^2 - 3r + 2\beta_1)/6$, *where* β_1 *is the number of solutions of x that satisfy* $3x \equiv 1$ *(mod r), and r is an integer larger than or equal to 3.*

The order of $(y,\ z,\ w)$ is used to obtain the parameter k such that the number of cases determined in Lemma 14.2 (i.e., $r^2 - 3r + 2\beta_1$) is divided by $3! = 6$. The following table shows the parameters r and k.

r	$(r^2 - 3r) + 2\beta_1$	k
4	4 + 2	1
5	10 + 2	2
6	18	3
7	28 + 2	5
8	40 + 2	7
9	54	9
10	70 + 2	12
11	88 + 2	15
12	108	18
13	130 + 2	22
14	154 + 2	26
15	180	30

Note that the codes also enable direct decoding. This is easily proved, and therefore the reader is encouraged to attempt the proof.

Example 14.8

The following shows an example code expressed by **H** with $r = 9$, $k = 9$, and $n = 18$:

$$(y,\ z,\ w) = \{(1,\ 4,\ 5),\ (5,\ 6,\ 8),\ (1,\ 2,\ 7),\ (0,\ 2,\ 8),\ (4,\ 7,\ 8),\ (0,\ 3,\ 7),\ (0,\ 4,\ 6),\ (1,\ 3,\ 6),\ (2,\ 3,\ 5)\},$$

$$\mathbf{H} = [\,\mathbf{P}_A \mid \mathbf{I}\,] = \left[\begin{array}{cccccccc:cccccccc:c} 0&0&0&1&0&1&1&0&0&1&0&0&0&0&0&0&0&0 \\ 1&0&1&0&0&0&0&1&0&0&1&0&0&0&0&0&0&1 \\ 0&0&1&1&0&0&0&0&1&0&0&1&0&0&0&0&0&2 \\ 0&0&0&0&0&1&0&1&1&0&0&0&1&0&0&0&0&3 \\ 1&0&0&0&1&0&1&0&0&0&0&0&0&1&0&0&0&4 \\ 1&1&0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&5 \\ 0&1&0&0&0&0&1&1&0&0&0&0&0&0&0&1&0&6 \\ 0&0&1&0&1&1&0&0&0&0&0&0&0&0&0&0&1&7 \\ 0&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&8 \end{array}\right].$$

In the next subsection we consider a general class of codes satisfying $y + z + w \equiv \alpha \pmod{r}$ and also $3x \equiv \alpha \pmod{r}$, where α is an integer, $0 \leq \alpha \leq r - 1$. This is a general class of additive-3 codes mentioned earlier.

14.2.4 Extended BIBD Codes and Additive Codes

The new generalized condition for direct decoding of the triple erasure correcting codes generates an efficient code that reduces the number of check disk subsystems [OHDE05a, 05b].

Theorem 14.8 *The codes can correct three erased disk data by direct decoding if and only if any three distinct binary weight-3 column vectors, h_i, h_j, and h_k, in the matrix $\boldsymbol{P}$ with r rows (≥ 4) of the codes shown by $\boldsymbol{H} = [\,\boldsymbol{P} \mid \boldsymbol{I}\,]$ satisfy the following:*

$$w(h_i \vee h_j \vee h_k) \geq 5, \tag{14.22}$$

where $x \vee y$ means logical OR of binary vectors x and y.

Proof Let h be a resultant vector of $h_i \vee h_j \vee h_k$, then the relation (14.22) says that h has weight larger than or equal to 5. If each of the corresponding 5 rows of these three distinct column vectors has weight larger than or equal to two, then these three column vectors have the number of 1's larger than or equal to 10. Since each of these three column vectors has weight three, h should have at most weight 9. This means there exists at least one row of these three vectors, for example, the z-th row, $0 \leq z \leq r - 1$, having weight one. Assume that h_i has '1' at the z-th row position, then the other two weight-3 columns h_j and h_k have at least two other row positions having patterns '01' and '10' because these two column vectors are distinct. So the three erasures can be corrected by direct decoding.

Conversely, for the following three distinct weight-3 column vectors, h_i, h_j, and h_k, for example, the resultant vector h satisfies $w(h) = w(h_i \vee h_j \vee h_k) = 4$. This means every row has weight two or three, which does not enable direct decoding. If there exists one row with weight one in these three column vectors in order to satisfy the direct decoding, such as h_j having '1' at this row position, then the remaining two column vectors h_i and h_k should be same. This contradicts the notion that every column is distinct. That is, $w(h) \neq 4$. Therefore $w(h_i \vee h_j \vee h_k) \geq 5$.

$$\begin{matrix} h_i & h_j & h_k \end{matrix} \qquad h = h_i \vee h_j \vee h_k$$

$$\begin{bmatrix} 0 & 0 & 0 \\ \cdot & \cdot & \cdot \\ 0 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \\ \cdot & \cdot & \cdot \\ 0 & 0 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 \\ \cdot \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ \cdot \\ 0 \end{bmatrix} \begin{matrix} \uparrow \\ \\ \\ \\ \\ r \\ \\ \\ \\ \downarrow \end{matrix} .$$

Q.E.D.

The following theorem gives an extension code to the one in Theorem 14.1.

Theorem 14.9 *If the incidence matrix* $\boldsymbol{P} = [\, \boldsymbol{P}_0 \,\vdots\, \boldsymbol{P}_1 \,]$ *satisfies the following conditions, the matrix* $\boldsymbol{H} = [\, \boldsymbol{P} \mid \boldsymbol{I} \,]$ *is the parity-check matrix of the triple erasure correcting codes that satisfies the direct decoding*:

1. *Every weight-3 column in* $\boldsymbol{P}$ *is distinct.*
2. *Every pair of two weight-3 columns, namely* h_i *and* h_j*, in each of* $\boldsymbol{P}_0$ *and* $\boldsymbol{P}_1$ *has 1's at most in one same row position, that is,* $w(h_i \wedge h_j) \leq 1$*, where* $x \wedge y$ *means logical AND of binary vectors of* x *and* y.

This theorem can be easily proved such that any three weight-3 columns in $\mathbf{P}$ satisfy the relation (14.22). The following shows how to design $\mathbf{P}_0$ and $\mathbf{P}_1$ based on the BIBD codes and the additive-3 codes.

Extended BIBD Codes

Theorem 14.10 *The following matrices* $\boldsymbol{P}_0$ *and* $\boldsymbol{P}_1$ *show the incidence matrices in BIBD codes that satisfy Theorem 14.9:*

$$\boldsymbol{P}_0 = \left[\begin{array}{ccc:c:ccc:c} \boldsymbol{P}_{s_0} & \boldsymbol{0} & \boldsymbol{I} & \cdots & \cdots & \boldsymbol{P}_{s_{q-1}} & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{P}_{s_0} & \boldsymbol{0} & \cdots & \cdots & \boldsymbol{I} & \boldsymbol{P}_{s_{q-1}} & \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{P}_{s_0} & \cdots & \cdots & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{P}_{s_{q-1}} & \boldsymbol{I} \end{array}\right],$$

$$\boldsymbol{P}_1 = \left[\begin{array}{ccc:c:ccc} \boldsymbol{P}_{s_0} & \boldsymbol{0} & \boldsymbol{I} & \cdots & \cdots & \boldsymbol{P}_{s_{q-1}} & \boldsymbol{0} & \boldsymbol{I} \\ \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{P}_{s_0} & \cdots & \cdots & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{P}_{s_{q-1}} \\ \boldsymbol{I} & \boldsymbol{P}_{s_0} & \boldsymbol{0} & \cdots & \cdots & \boldsymbol{I} & \boldsymbol{P}_{s_{q-1}} & \boldsymbol{0} \end{array}\right].$$

Here $\boldsymbol{P}_{s_i}$*,* $0 \leq i \leq q-1$*, is an* $s \times s$ *binary matrix, where* $s = 2q+1$*, shown in Theorem 14.3, in which the first column vector of s-th degree has two 1's at the* $(i+1)$*-th and the* $(2q-i)$*-th positions, and the remaining* $s-1$ *distinct vectors are generated by the first column vector cyclic shifted in downward by* $s-1$ *times. The information length of the code defined by* $\boldsymbol{H} = [\, \boldsymbol{P}_0 \,\vdots\, \boldsymbol{P}_1 \mid \boldsymbol{I} \,]$ *is* $k = r(r-2)/3$*, where* $r \equiv 3 \pmod 6$.

An example of $\mathbf{P}_{s_0}$ with $q = 1$, and $s = 3$ is shown below:

$$\mathbf{P}_{s_0} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

From this example we have the incidence matrix $\mathbf{P}_0$ with $r \times k_0$, where $r = 9$, and $k_0 = 12$, the same as considered in Example 14.6.

The maximum numbers of column vectors in $\mathbf{P}_0$ and $\mathbf{P}_1$ are shown as $k_0 = r(r-1)/6$ and $k_1 = r(r-3)/6$, where $r \equiv 3 \pmod 6$, respectively. Therefore the information length of the code defined by $\mathbf{H} = [\ \mathbf{P} \mid \mathbf{I}\ ] = [\ \mathbf{P}_0 \vdots \mathbf{P}_1 \mid \mathbf{I}\ ]$ can be expressed by

$$k = k_0 + k_1 = \frac{r(r-2)}{3},$$

where $r \equiv 3 \pmod 6$. Since $\mathbf{P}_{s_i}$ has all weight-2 columns, the matrix $\mathbf{P}$ satisfies two conditions of Theorem 14.9. Therefore $\mathbf{H} = [\ \mathbf{P}_0 \vdots \mathbf{P}_1 \mid \mathbf{I}\ ]$ is a parity-check matrix of the triple erasure correcting codes that satisfies the direct decoding.

Example 14.9 [OHDE05b]

The following shows the parity-check matrix of the (30, 21) codes that satisfies Theorem 14.9 with parameters of $q = 1$, and $r = 9$:

$$\mathbf{H} = \left[\begin{array}{ccccccccc|c}
0\,1\,1 & 0\,0\,0 & 1\,0\,0 & 1\,0\,0 & 0\,1\,1 & 0\,0\,0 & 1\,0\,0 & 1\,0\,0\,0\,0\,0\,0\,0\,0 \\
1\,0\,1 & 0\,0\,0 & 0\,1\,0 & 0\,1\,0 & 1\,0\,1 & 0\,0\,0 & 0\,1\,0 & 0\,1\,0\,0\,0\,0\,0\,0\,0 \\
1\,1\,0 & 0\,0\,0 & 0\,0\,1 & 0\,0\,1 & 1\,1\,0 & 0\,0\,0 & 0\,0\,1 & 0\,0\,1\,0\,0\,0\,0\,0\,0 \\
\hline
1\,0\,0 & 0\,1\,1 & 0\,0\,0 & 1\,0\,0 & 0\,0\,0 & 1\,0\,0 & 0\,1\,1 & 0\,0\,0\,1\,0\,0\,0\,0\,0 \\
0\,1\,0 & 1\,0\,1 & 0\,0\,0 & 0\,1\,0 & 0\,0\,0 & 0\,1\,0 & 1\,0\,1 & 0\,0\,0\,0\,1\,0\,0\,0\,0 \\
0\,0\,1 & 1\,1\,0 & 0\,0\,0 & 0\,0\,1 & 0\,0\,0 & 0\,0\,1 & 1\,1\,0 & 0\,0\,0\,0\,0\,1\,0\,0\,0 \\
\hline
0\,0\,0 & 1\,0\,0 & 0\,1\,1 & 1\,0\,0 & 1\,0\,0 & 0\,1\,1 & 0\,0\,0 & 0\,0\,0\,0\,0\,0\,1\,0\,0 \\
0\,0\,0 & 0\,1\,0 & 1\,0\,1 & 0\,1\,0 & 0\,1\,0 & 1\,0\,1 & 0\,0\,0 & 0\,0\,0\,0\,0\,0\,0\,1\,0 \\
0\,0\,0 & 0\,0\,1 & 1\,1\,0 & 0\,0\,1 & 0\,0\,1 & 1\,1\,0 & 0\,0\,0 & 0\,0\,0\,0\,0\,0\,0\,0\,1
\end{array}\right].$$

$\mathbf{P}_0$ $\mathbf{P}_1$ $\mathbf{I}$

Extended Additive-3 Codes The extended additive-3 code with larger code length than that of the existing additive code, called an *additive-3$_\alpha$ code*, is defined by the following lemmas and theorems.

Lemma 14.3 *The number of solutions of an integer x that satisfies $3x \equiv \alpha$ (mod r), where $0 \le x \le r-1$, r is an integer (≥ 3), and α is also an integer ($0 \le \alpha \le r-1$), is denoted as β_α and expressed as follows:*

$$\beta_\alpha = \begin{cases} 3 & \text{for } \alpha = \text{multiple of } 3, \text{and } r = \text{multiple of } 3, \\ 1 & \text{for } \alpha = \text{any integer, and, } r = \text{integer other than multiple of } 3, \\ 0 & \text{for } \alpha = \text{integer other than multiple of } 3, \text{and } r = \text{multiple of } 3. \end{cases}$$

The reader is encouraged to prove this. This lemma gives a generalized form for α, that is, Lemma 14.1 is a special case of $\alpha = 1$ of this lemma.

Theorem 14.11 *An additive-3_α code defined by the parity-check matrix $\boldsymbol{H} = [\, \boldsymbol{P}_0 \mid \boldsymbol{I} \,]$, where $\boldsymbol{I}$ is an $r \times r$ identity matrix and $\boldsymbol{P}_0$ is an $r \times k$ binary matrix with distinct weight-3 columns, can correct three erasures by the direct decoding. Let y, z, and w be row numbers of three 1's in each column in $\boldsymbol{P}_0$ that satisfy $y + z + w \equiv \alpha \bmod r$, where $0 \le \alpha \le r - 1$. Then the parameter k is determined by $(r^2 - 3r + 2\beta_\alpha)/6$, where β_α is the number of solutions of an integer x that satisfies $3x \equiv \alpha \pmod r$.*

This theorem can be proved such that any two columns in $\mathbf{P}_0$ have at most one row with two 1's in the same row positions; that is, the AND operation of these two column vectors results in a vector with weight at most one.

Lemma 14.4 *The additive-3_α codes with $\alpha = 0$ have the following information length:*

$$k = \left\lfloor r\frac{(r-3)}{6} \right\rfloor + 1.$$

Let r be multiple of 3, then the additive-3_α code with $\alpha = 0$ has

$$k = \frac{r(r-3)}{6} + 1.$$

Example 14.10 [OHDE05a]

For $r = 9$ and $\alpha = 0$, the information length k is determined as $(r^2 - 3r + 2\beta_\alpha)/6 = (81 - 27 + 6)/6 = 10$, which is larger than the code shown in Example 14.8 by one where $k = 9$ and $\alpha = 1$. In this case the set of three integers is as follows: $(y,\ z,\ w) = \{(0,\ 1,\ 8),\ (0,\ 2,\ 7),\ (0,\ 3,\ 6),\ (0,\ 4,\ 5),\ (1,\ 2,\ 6),\ (1,\ 3,\ 5),\ (2,\ 3,\ 4),\ (3,\ 7,\ 8),\ (4,\ 6,\ 8),\ (5,\ 6,\ 7)\}$. This gives the following incidence matrix $\mathbf{P}_0$:

$$\mathbf{P}_0 = \begin{bmatrix}
1 & 1 & 1 & 1 & & & & & & \\
1 & & & & 1 & 1 & & & & \\
 & 1 & & & 1 & & 1 & & & \\
 & & 1 & & & 1 & 1 & 1 & & \\
 & & & 1 & & & 1 & & 1 & \\
 & & & 1 & & 1 & & & & 1 \\
 & & 1 & & 1 & & & & 1 & 1 \\
 & 1 & & & & & & 1 & & 1 \\
1 & & & & & & & 1 & 1 &
\end{bmatrix}.$$

Theorem 14.12 *Let $\boldsymbol{P}_1$ be an incidence matrix cyclic shifted downward by one row of the matrix $\boldsymbol{P}_0$ with $\alpha = 0$. With using these matrices of $\boldsymbol{P}_0$ and $\boldsymbol{P}_1$, a general class of additive-3_α code $\boldsymbol{C}$* with $\alpha = 0$ *defined by the parity-check matrix $\boldsymbol{H} = [\, \boldsymbol{P}_0 \,\vdots\, \boldsymbol{P}_1 \mid \boldsymbol{I} \,]$ with information length $k = 2(\lfloor r(r-3)/6 \rfloor + 1)$ corrects triple erasures by direct decoding for a given $r\ (\ge 4)$.*

Example 14.11 [OHDE05b]

The following shows the parity-check matrix of the (29, 20) triple erasure correcting additive-3_α (with $\alpha = 0$) code:

$$\mathbf{H} = \left[\begin{array}{ccc|ccc|c} 1&1&1&1&0&0&0&0&0&0 & 1&0&0&0&0&0&0&1&1&0 & 1&0&0&0&0&0&0&0&0&0 \\ 1&0&0&0&1&1&1&0&0&0 & 1&1&1&1&0&0&0&0&0&0 & 0&1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&1&0&1&0&0&0 & 1&0&0&0&1&1&1&0&0&0 & 0&0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&1&1&1&0&0 & 0&1&0&0&1&0&1&0&0&0 & 0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&1&0&1&0 & 0&0&1&0&0&1&1&1&0&0 & 0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&1&0&0&0&1 & 0&0&0&1&0&0&1&0&1&0 & 0&0&0&0&0&1&0&0&0&0 \\ 0&0&1&0&1&0&0&0&1&1 & 0&0&0&1&0&1&0&0&0&1 & 0&0&0&0&0&0&1&0&0&0 \\ 0&1&0&0&0&0&0&1&0&1 & 0&0&1&0&1&0&0&0&1&1 & 0&0&0&0&0&0&0&1&0&0 \\ 1&0&0&0&0&0&0&1&1&0 & 0&1&0&0&0&0&0&1&0&1 & 0&0&0&0&0&0&0&0&1&0 \\ 1&0&0&0&0&0&0&0&1&1 & 0&1&0&0&0&0&0&1&0&1 & 0&0&0&0&0&0&0&0&0&1 \end{array}\right] .$$

$\mathbf{P}_0$ $\mathbf{P}_1$ $\mathbf{I}$

Evaluation Figure 14.7 presents the relation between the information-bit length k and the check-bit length r of the extended BIBD codes shown in Theorem 14.10 and the additive-3_α codes with $\alpha = 0$, denoted as additive-3_0 codes, shown in Theorem 14.12. Note that for $k = 120$ bits the extended codes, both extended BIBD codes and additive-3_α codes, require $r = 21$ bits while the existing BIBD codes and the existing additive-3 codes with $\alpha = 1$, denoted as additive-3_1 codes, require $r = 33$ bits.

Figure 14.8 shows the relation between the group size multiplied by three and the number of check disk subsystems r. The group size is an average row weight in $\mathbf{H}$ that is equal to the weight of $\mathbf{H}$ (i.e., $3k + r$) divided by the number of rows r (i.e., $3k/r + 1$). The group size indicates the average number of disk subsystems that should be read in order to recover single-disk subsystem failures. This influences the distributed system's performance, and therefore it is an important metric of system performance. If the group size is large, then the system has to read the data from a large number of nonfailed disks in order to recover the failed disk data, which eventually degrades system performance. Here we assume that three times as many as the (group size -1) of the disk subsystems

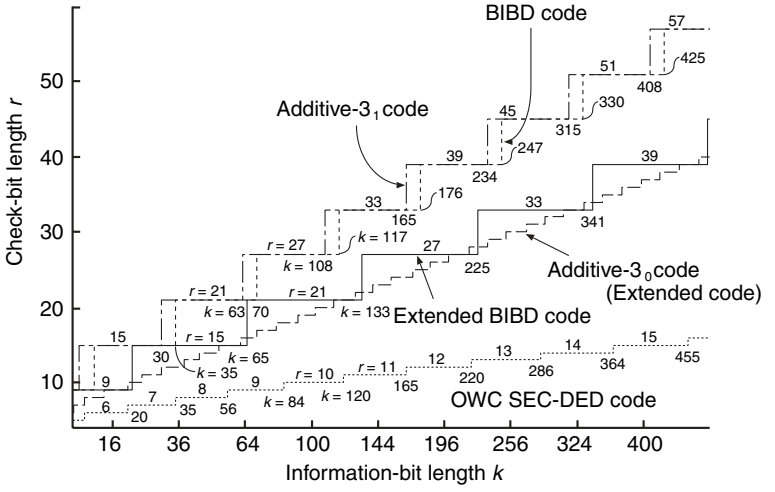


Figure 14.7 *Comparison of information-bit lengths and check-bit lengths of the distributed storage system.*

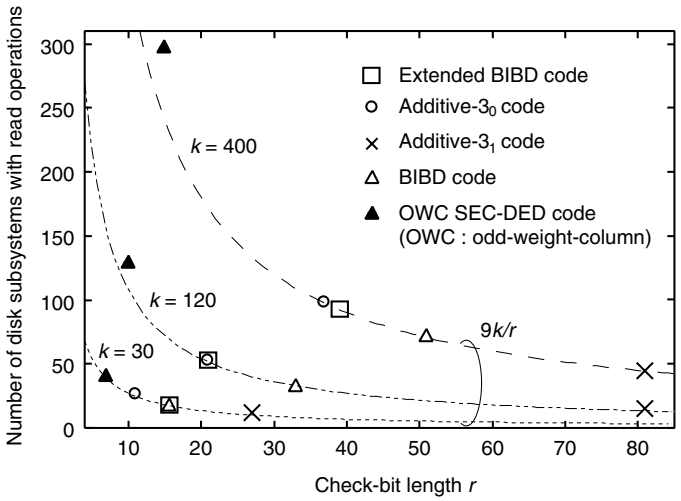


Figure 14.8 *Relation between check-bit length and number of disk subsystems with read operations for three disks recovery.*

(i.e., $3\times(3k/r)$) are required in order to recover three erased disk data. In this figure the cases of the 'OWC SEC-DED code', namely odd-weight-column SEC-DED code, are out of the $9k/r$ line. Because the matrix $\mathbf{P}$ in $\mathbf{H}$ of the code is constructed by all distinct weight-3 columns, which does not necessarily satisfy the conditions of direct decoding, the row operations are performed to satisfy the conditions of direct decoding, which will finally increase the number of 1's in the row.

EXERCISES

14.1 Prove that the code defined by Eq. (14.4) over the ring of binary polynomial $\mathbf{M}_p(x) = (x^p - 1)/(x - 1) = x^{p-1} + x^{p-2} + \cdots + x + 1$, where p is prime, is MDS.

14.2 Prove that the code $\mathbf{C}_p$ has the minimum distance $r + 1$ for $r \leq 3$ by the relation shown in Eq. (14.11).

14.3 Prove that the codes $\mathbf{C}_p$ with $r = 4$ and 5 are MDS for all $p \neq 3$.

14.4 Prove that in EVENODD the diagonal parity Q can be obtained by exclusive-OR of the bits in two parity columns.

14.5 Prove that if p is not prime in EVENODD, then the code has minimum distance 2.

14.6 Assume that the following EVENODD array with $p = 5$ has an erroneous column data in the array. Indicate how to correct the erroneous array.

1	0	0	1	0	1	1
0	1	1	0	0	1	0
1	1	0	0	0	0	1
1	1	0	1	1	1	0

14.7 Prove that the X-code has a column distance 3; that is, it is an MDS array code with size $n \times n$, if and only if n is a prime number.

14.8 Encode the following 5×7 information array of the X-code.

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

14.9 Decode the following two 5×5 binary arrays of the X-code, where the first array includes a single column error and the second array includes two erased columns.

	0	1	2	3	4	
0	1	0	1	1	1	
1	0	1	1	1	1	
2	0	0	1	0	1	
3	0	0	0	1	0	← 1st parity row
4	1	1	0	1	1	← 2nd parity row

1	?	?	1	1	
0	?	?	1	1	
0	?	?	0	1	
0	?	?	1	0	← 1st parity row
1	?	?	1	1	← 2nd parity row

14.10 Derive the relations (14.19) and (14.20).

14.11 Design a Steiner system with $2q + 1 = 5$ based on Theorem 14.3.

14.12 Design the BIBD code with $t = 4$ based on Theorem 14.6.

14.13 Show that the code indicated in Theorem 14.7 enables direct decoding.

14.14 Find five columns of $\mathbf{P}_A$ in Theorem 14.7 for $r = 7$; that is, find five combinations of three integers of $(y,\ z,\ w)$ that satisfy Lemma 14.2.

14.15 Prove that $(r^2 - 3r + 2\beta_1)/6$ has always integer values for a given integer r and β_1 defined by Lemma 14.1. Also prove that k indicated in Theorem 14.7 always takes integer values.

14.16 Design the (80, 65) extended BIBD code with parameters $q = 2$ and $s = 5$.

14.17 Prove Lemmas 14.3 and 14.4.

14.18 Prove Theorems 14.9 through 4.12.

14.19 Prove that the code $\mathbf{C}$ defined by the binary parity-check matrix $\mathbf{H} = [\ \mathbf{P} \mid \mathbf{I}\ ]$, where $\mathbf{I}$ is an $r \times r$ identity matrix and $\mathbf{P}$ an $r \times k$ binary matrix with distinct weight-4 columns, can correct four erasures if the following relation is satisfied for any four columns, u_i, u_j, u_k, and u_m in $\mathbf{P}$:

$$w(u_i \vee u_j \vee u_k \vee u_m) \geq 11,$$

where $w(v)$ means the weight of binary vector v, and $x \vee y$ expresses logical OR of two binary vectors x and y.

14.20 Discuss additive-4 codes.

REFERENCES

[ALVA02] G. A. Alvarez, W. A. Burkhard, and F. Cristian, ''Tolerating Multiple Failures in RAID Architectures with Optimal Storage and Uniform Declustering,'' *Proc. 24th IEEE Int. Symp. on Computer Architecture* (June 2002): 62–72.

[BLAU93] M. Blaum and R. M. Roth, ''New Array Codes for Multiple Phased Burst Correction,'' *IEEE Trans. Info. Theory*, 39 (January 1993): 66–77.

[BLAU94] M. Blaum, J. Brady, J. Bruck, and J. Menon, ''EVENODD: An Efficient Scheme for Tolerating Double Disk Failures in RAID Architectures,'' *IEEE Trans. Comput.*, 44 (February 1994): 192–202.

[BLAU96] M. Blaum, J. Bruck, and A. Vardy, ''MDS Array Codes with Independent Parity Symbols,'' *IEEE Trans. Info. Theory*, 42 (March 1996): 529–542.

[FENG05] G.-L. Feng, R. H. Deng, F. Baq, and J.-C. Shen, ''New Efficient MDS Array Codes for RAID Part I: Read-Solomon-Like Codes for Tolerating Three Disk Failures,'' *IEEE Trans. Comput*., 54 (September 2005): 1071–1080.

[GIBS89] G. A. Gibson, L. Hellerstein, R. M. Karp, R. H. Katz, and D. A. Patterson, ''Failure Correction Techniques for Large Disk Arrays,'' *Proc. 11th IEEE Int. Conf. on Computer Architecture* (April 1989): 123–132.

[GIBS92] G. A. Gibson, *Redundant Disk Arrays, Reliable Parallel Secondary Storage*, MIT Press (1992).

[HALL86] M. H. Hall Jr., *Combinatorial Theory*, Wiley (1986).

[HELL94] L. Hellerstein, G. A. Gibson, R. M. Karp, R. H. Katz, and D. A. Patterson, ''Coding Techniques for Handling Failures in Large Disk Arrays,'' *Algorithmica*, 12 (1994): 182–208.

[OHDE05a] H. Ohde and E. Fujiwara, ''A Code Construction Method for Distributed File Memory Systems'' (in Japanese), *IEICE Technical Report*, FIIS 05, no. 162 (June 2005): 1–8.

[OHDE05b] H. Ohde and E. Fujiwara, ''A Class of Low Density Codes for Distributed Storage Systems'' (in Japanese), *Proc.* 2005 *Forum on Information Technology* (September 2005): 6S-1.

[XU99a] L. Xu and J. Bruck, ''X-Code: MDS Array Codes with Optimal Encoding,'' *IEEE Trans. Info. Theory*, 45 (January 1999): 272–276.

[XU99b] L. Xu, V. Bohossian, J. Bruck, and D. G. Wagner, ''Low-Density MDS Codes and Factor of Complete Graphs,'' *IEEE Trans. Info. Theory*, 45 (September 1999): 1817–1826.

Index