Fault-Tolerant Computation using Algebraic Homomorphisms

Paul E. Beckmann

RLE Technical Report No. 580

June 1993

Research Laboratory of Electronics
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139-4307

This work was supported in part by the Defense Advanced Research Projects Agency monitored by the U.S. Navy Office of Naval Research under Grant N00014-89-J-1489 and in part by the Charles S. Draper Laboratories under Contract DL-H-418472.

Fault-Tolerant Computation using Algebraic Homomorphisms

by

Paul E. Beckmann

Submitted to the Department of Electrical Engineering and Computer Science
on August 14, 1992, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Arithmetic codes are a class of error-correcting codes that are able to protect computation more efficiently than modular redundancy. In this thesis we consider the problem of designing an arithmetic code to protect a given computation. The main contributions are as follows:

- The first constructive procedure, outside of modular redundancy, for generating arithmetic codes for a large class of operations. The approach is mathematically rigorous, based on group-theory, and fully characterizes the important class of systematic-separate codes. The results encompass computation that can be modeled as operations in an algebraic group, ring, field, or vector space.
- A novel set-theoretic framework for characterizing the redundancy present in a system. We present a decomposition of a robust system into a cascade of three systems and determine general requirements for multiple error detection and correction.
- We identify an important class of errors for which the redundancy present in a system may be completely characterized by a single integer, analogous to the minimum distance of a binary error-correcting code.
- We unify the existing literature on arithmetic codes. A wide variety of seemingly unrelated codes are combined into a single general framework.
- A large number of examples illustrating the application of our technique are presented.
- Detailed analyses of two new and practical fault-tolerant systems: fault-tolerant convolution and A/D conversion.

Thesis Supervisor: Bruce R. Musicus
Title: Research Affiliate, M.I.T. Digital Signal Processing Group

Dedicated to the glory of God,

Whatever you do, whether in word or deed,
do it all in the name of the Lord Jesus,
giving thanks to God the Father through him.
— Colossians 3:17

and to my wife, Chin.

A wife of noble character who can find?
She is worth far more than rubies.
— Proverbs 31:10

Acknowledgments

I would like to thank my advisor, Bruce Musicus, for the excellent guidance and support that he has given me during the past three years. His enthusiasm, encouragement, and friendship have made my life as a graduate student both satisfying and enjoyable. Bruce was always available when I reached an impasse, and gave me the freedom to explore new ideas.

I am grateful for the support of Professor Alan Oppenheim. Under his direction, the Digital Signal Processing Group provided a stimulating environment in which to carry out my research. I am also indebted to all the members of the group for their friendship and for many thought provoking discussions. I would like to particularly acknowledge Kambiz Zangi, Mike Richard, Steven Isabelle, and Andy Singer. They were always available and patiently let me bounce ideas off of them. I would also like to thank Deborah Gage for her constant encouragement and help, and Sally Bemus for assistant in the final preparation of the thesis and defense.

This thesis was made possible by the generous financial support of Rockwell International. They supported me unconditionally for the past three years and allowed me to pursue an interesting thesis topic. Additional funding was provided by the Defense Advanced Research Projects Agency and Draper Laboratory.

I would like to thank my parents for their steadfast love and support. They have always been an encouragement and inspiration to me.

Finally, and most importantly, I would like to thank my wife Chin for her unfailing love, patience, and support. She has always been behind me 100%, and is a perpetual source of joy. This thesis would not have been possible without her.

Contents

Chapter 1

Introduction

Traditionally, the problem of computational fault-tolerance has been solved through modular redundancy. In this technique, several identical copies of the system operate in parallel using the same data, and their outputs are compared with voter circuitry. If no errors have occurred, all outputs will agree exactly. Otherwise, if an error has occurred, the faulty module can be easily identified and the correct output determined. Modular redundancy is a general technique and can be applied to any computational task. Unfortunately, it does not take advantage of the structure of a problem and requires a large amount of hardware overhead relative to the protection afforded.

A more efficient method of protecting computation is to use an arithmetic code and tailor the redundancy to the specific operation being performed. Arithmetic codes are essentially error-correcting codes whose error detecting and correcting properties are preserved during computation. Arithmetic codes offer performance and redundancy advantages similar to existing error-correcting codes used to protect communication channels. Unfortunately, arithmetic codes exist for only a limited number of operations.

This thesis addresses the general problem of designing an arithmetic code to protect a given computation. A practical arithmetic code must satisfy three requirements:

1. It must contain useful redundancy.

2. It must be easily encoded and decoded.

3. Its inherent redundancy must be preserved throughout computation.

The first two requirements are shared by error-correcting codes for protecting data transmission, while the third requirement is unique to arithmetic codes. This host of requirements makes designing practical arithmetic codes an extremely difficult problem.

We solve this problem through two key insights. First, we note that many important arithmetic operations can be modeled using group theory. This provides a mathematically rigorous and well-defined foundation for our analysis. Second, we begin our study by focusing on the third requirement of an arithmetic code: that its structure be preserved throughout computation. This requirement restricts arithmetic codes to the class of mappings known as algebraic homomorphisms, which can often be readily identified. From the set of algebraic homomorphisms, we select those with useful distance structures and that are easily encoded and decoded. This leads to practical arithmetic codes.

Our results are extremely general and encompass any computation that can be described as operations in algebraic groups, rings, fields, or vector spaces. This includes a wide range of important and useful arithmetic operations such as low-level integer arithmetic, matrix operations, convolution, and linear transformations. Throughout the thesis, we will emphasize operations which are of interest in signal processing, although our results are more widely applicable.

1.1 Major Contributions

Several important contributions to the study of arithmetic codes are made in this thesis. First and foremost is a constructive procedure for generating arithmetic codes for a large class of operations. This is the first general procedure, outside of modular redundancy, for constructing arithmetic codes. Our approach is mathematically rigorous, based on group theory, and yields the class of systematic-separate codes. The difficult problem of finding arithmetic codes is reduced to the problem of finding subgroups. By finding all subgroups of a given group, we generate all possible systematic-separate codes.

Another contribution is the general set-theoretic framework for analyzing fault-tolerant systems. We give explicit measures for redundancy which indicate when error detection and correction are feasible, and approach this problem from the most general point of view. The effects of multiple errors are considered, and we are able to tradeoff between the abilities to detect and correct errors. We also identify an important class of errors which enables the redundancy in a system to be more easily quantified. We answer the question: When is it possible to analyze the redundancy present in a code by some minimum distance measure?

Our work unifies the vast majority of existing arithmetic codes, including low-level integer codes, as well as higher-level Algorithm-Based Fault-Tolerance schemes. We show that the underlying thread common to these apparently unrelated operations is that computation can be modeled as operations in an algebraic group and that errors influence the result in an additive manner.

This thesis also contributes two novel and practical fault-tolerant systems and analyzes their performance in detail. In the first, we apply a polynomial residue number system to protect the convolution of discrete sequences. Although not the first fault-tolerant convolution algorithm proposed, it is unmatched by its overall efficiency; single system failures may be detected and corrected with as little as 65% overhead. Also, since residue number systems are the basis for fast convolution algorithms, we are able to employ FFTs to reduce the amount of computation required.

Our second major application addresses the problem of reliable A/D conversion. We add redundancy in an original manner by oversampling the input signal and show that error detection and correction reduce to a simple form, requiring about as much additional computation as a single FIR filter.

1.2 Outline of Thesis

Chapter 2 motivates our study of fault-tolerance and reviews previous approaches to the design of robust systems. We discuss approaches based on modular redundancy, as well as more efficient methods utilizing arithmetic codes. We summarize existing low-level codes for protecting integer arithmetic and discuss recently developed high-level codes referred to as Algorithm-Based Fault-Tolerance. We briefly introduce each coding scheme, paying particular attention to the manner in which redundancy is added.

Chapter 3 examines the problem of computational fault-tolerance from a set-theoretic point of view. Motivated by the many examples of fault-tolerant systems in Chapter 2, we decompose a robust system into a cascade of three subsystems. We study redundancy and derive conditions that allow multiple errors to be detected and corrected. We then present a specific class of errors, which we term symmetric errors, that enables the redundancy present in a system to be quantified by a single integer, analogous to the minimum distance of a standard binary error-correcting code. Our definition of symmetric errors is novel, and relates the abstract requirements for error detectability and correctability to the more familiar notion of minimum distance. Our results are extremely general and can be applied to the analysis of any fault-tolerant system. Unfortunately, the set-theoretic approach does not yield a constructive procedure for designing robust systems.

Chapter 4 narrows the focus from arbitrary computation to that which can be modeled as operations in an algebraic group. We first present a group-theoretic framework of redundant computation based on the decomposition of a robust system presented in Chapter 3, and show that the constraints satisfied by an arithmetic code are equivalent to those of an algebraic homomorphism. This reduces the problem of finding suitable arithmetic codes to that of finding applicable algebraic homomorphisms. We recast the general conditions on redundancy for error detection and correction in terms of group theory and show that these functions may be done in an efficient manner using a syndrome homomorphism. Once the framework for groups is complete, we extend it to rings, fields, and vector spaces. These are other algebraic systems that have an underlying group structure. Assuming an additive error model, we show that arithmetic codes for these operations must be homomorphisms as well. This enables the bulk of the results for groups to be applied to these other algebraic systems. We conclude with a wide variety of examples which demonstrate the general nature of our results.

Chapter 5 considers the problem of determining possible homomorphisms for a given algebraic operation. We use a quotient group isomorphism, and this yields the important class of systematic-separate codes. These codes are characterized by a checksum computation performed in a parallel independent channel. We prove that the procedure is capable of finding, up to isomorphisms, all possible systematic-separate codes for a given operation. We then extend the results for groups to rings, fields, and vector spaces. The results for groups carry over completely, and we are able to characterize all possible systematic-separate arithmetic codes in these other systems as well. We conclude this chapter by presenting several examples of operations protected by systematic-separate codes. The examples include novel schemes unique to this thesis as well as schemes presented by other authors.

Chapters 6 and 7 each contained a thorough study of a particular fault-tolerant system. Chapter 6 describes a fault-tolerant convolution algorithm which is an extension of residue number system fault-tolerance schemes applied to polynomial rings. The algorithm is suitable for implementation on multiprocessor systems and is able to concurrently mask processor failures. We develop a fast algorithm based on long division for detecting and correcting multiple processor failures. We then select moduli polynomials that yield an efficient and robust FFT-based algorithm. For this implementation, we study single fault detection and correction, and apply a generalized likelihood ratio test to optimally detect system failures in the presence of computational noise.

Chapter 7 examines a fault-tolerant round-robin A/D converter system. A modest amount of oversampling generates information which is exploited to achieve fault tolerance. A generalized likelihood ratio test is used to detect the most likely failure and also to estimate the optimum signal reconstruction. The error detection and correction algorithms reduce to a simple form and require only a slight amount of hardware overhead. We present a derivation of the algorithms, and discuss modifications that lead to a realizable system. We then evaluate overall performance through software simulations.

Lastly, in Chapter 8 we summarize the major contributions made in this thesis. We also suggest practical and potentially important directions for future research.

Chapter 2

Previous Approaches to Fault-Tolerance

In this section we motivate our study of fault-tolerance and introduce basic concepts and terminology. Different types of fault-tolerance are described, and we discuss traditional approaches to fault-tolerance based on modular redundancy, as well as more recently developed Algorithm-Based Fault-Tolerance schemes.

2.1 Introduction

High reliability is needed in many signal processing applications to ensure continuous operation and to check the integrity of results. High reliability is needed in life critical applications, such as aircraft guidance systems or in medical equipment, where failures can jeopardize human lives, or in remote applications, such as satellites or underwater acoustic monitors, where repair is impossible or prohibitively expensive. Robustness is also needed in systems that must operate in hazardous environments, such as military equipment, or in spacecraft that must be protected against radiation. In all of these applications there is a high cost of failure, and reliability is of great importance.

The complexity of signal processing algorithms has been steadily increasing due to the availability of special purpose, high-speed signal processors. Many algorithms that were once too computationally intensive, are now implemented in real time by multiprocessor systems. In these systems, the large amount of hardware increases the likelihood of a failure occurring, and makes reliable operation difficult.

It is impossible to guarantee that components of a system will never fail. Instead, failures should be anticipated, and systems designed to tolerate failures gracefully. This design methodology is known as *fault-tolerant computing* and it received considerable attention by early computer designers because of the unreliability of existing components. After the development of integrated circuits, which were several orders of magnitude more reliable, fault-tolerance became a secondary issue. Attention was focused on developing faster, more complex circuits, and as a result, circuit densities grew exponentially. In many areas, however, semiconductor reliability has not kept pace with the level of integration, and fault-tolerance is becoming a major issue again.

A component is said to have *failed* when it does not compute the correct output, given its input. A failure is caused by a physical *fault*, and the manifestation of a failure is *errors* within the system [1]. In a fault-tolerant system, the basic idea is to tolerate internal errors, and keep them from reaching and corrupting the output. This process is known as error *masking.*

A fault may be permanent or transient. A permanent fault is caused by a static flaw in a component and may result from manufacturing defects or physical damage during operation. A permanent fault need not produce errors for every input. Transient or soft faults are momentary faults that occur infrequently and are randomly distributed throughout the system. These types of faults can be induced by several sources: alpha particles emitted from the substrate and packaging material; cross coupling between closely spaced signal lines; unpredictable timing glitches from rare, absolute worst case delays; and electromigration phenomenon in small conductors [2]. The current trends toward higher clock speeds and denser components aggravate the problem of transient errors [3].

The highest and most desirable level of fault-tolerance is known as *concurrent* error masking. In this technique, errors are masked during operation and the system continues to function with no visible degradation in performance. In general, concurrent error masking usually requires the following steps:

1. Fault Detection – determine that the output is invalid and that a fault has occurred

within the system.

2. Fault Location – determine which system component failed.

3. Fault Correction – determine the correct output.

Concurrent error masking is difficult and expensive to achieve, and basically requires the entire system to be checked for errors at each time step. Often, lower levels of protection are acceptable and a few erroneous outputs may be tolerated. In these instances, less expensive techniques which check only a part of the system at each time step are adequate. In other applications, there is a high cost for computing erroneous results and a lower cost for delaying the result. Is these applications, concurrent fault detection coupled with off-line fault location/correction may be a viable alternative.

2.2 Modular Redundancy

In order for a system to be fault-tolerant, it must contain some form of redundancy. By redundancy we mean additional states that arise during faults and that are used to detect and correct errors. Without redundancy, it is impossible for a system to be fault-tolerant since it is unable to distinguish between valid and invalid internal states. Utilizing redundancy is in contrast to the goal of eliminating as much redundancy as possible. Redundancy generally increases the complexity of a system and leads to increased cost.

The traditional method of adding redundancy and fault-tolerance to a system is through *modular redundancy* [4, 5]. This is a system-level approach in which several copies of the system operate in parallel, using the same input. Their outputs are compared with voter circuitry and will agree if no errors have occurred. Otherwise, if the outputs are not identical, then an error has occurred and the correct result may be determined using a majority voter.

A system, S, protected by modular redundancy is shown in Figure 2-1. Assume that there are N copies of S labeled S_1 to S_N. Then, it is possible to detect D and correct C errors if $N \geq D + C + 1$ where $D \geq C$. To detect single errors, at least $N = 2$ copies of S are needed (100% overhead). To correct single errors, at least $N = 3$ copies of S are needed (200% overhead).

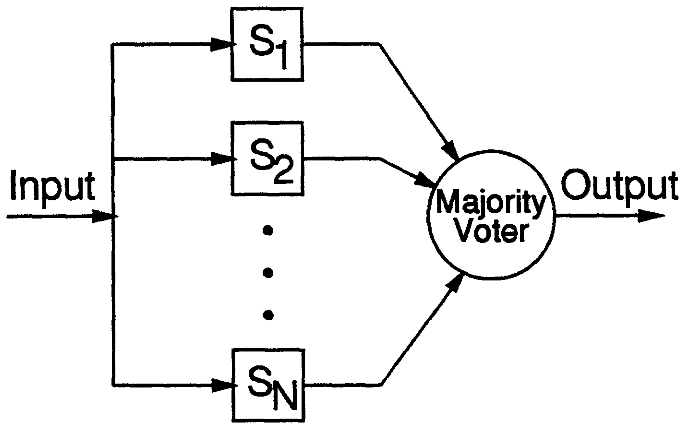


Figure 2-1: Example of N-modular redundancy used to protect system S.

Modular redundancy (MR) is the most widely used fault-tolerance technique. This is because it can be used to protect any system and since it decouples system and fault-tolerance design. That is, the design of S is basically independent of the design of the voter circuitry used to check S. Furthermore, totally self-checking checkers capable of protecting the voting process are available [6].

MR can be applied to a system at a variety of levels. At a low level, logic gates may be duplicated and the error masking performed by the gates themselves [7]. At a higher level, integrated circuits may be duplicated and their outputs compared; or the entire system may be duplicated. The optimal level at which to apply MR depends upon the expected types of faults and the cost of voter circuitry. An example of system level MR is the Jet Propulsion Laboratory's Star Space Shuttle Computer [8]. Five copies of the system operate in parallel and voting is performed in hardware. Other examples include Draper Laboratory's Fault-Tolerant Parallel Processor [9], and commercial systems by Tandem [10] and Stratus [11].

Another important area of research has been in developing self-checking systems. These are systems that are capable of concurrent fault detection. Coupled with MR, a self-checking system can significantly reduce the overhead required for fault-tolerance. N copies of a self-checking system contain sufficient redundancy to detect up to N and correct up to $N-1$ errors.

Using additional copies of system components is just one form of modular redundancy.

If performance is not a major bottleneck, and transient faults are expected, then time redundancy may be used. The system would perform the same operation N times, and then compare results. An example of time redundancy is a technique called Recomputing with Shifted Operands which can protect certain arithmetic operations [12]. In this technique, all operations are done twice, once with normal operands and once with shifted operands. It can be shown that the shifting will prevent any permanent or transient failure in the ALU from producing incorrect, but identical, results in both cases. This technique may be applied to yield totally self-checking systems. The hardware overhead needed is only that of the shifters and equality checker.

The fault-tolerant systems that we have been describing are all capable of concurrent error detection and correction. If a small number of erroneous outputs can be tolerated, then other techniques, such as roving emulators or watchdog processors, may be employed [13]. These schemes check only a portion of the computation at each time step and are well-suited to multiprocessor systems.

2.2.1 Summary

MR is a very general approach to fault-tolerance. It is applicable to any type of system and allows fault-tolerance to be added even after system design is complete. Redundancy is incorporated by a brute force approach: system duplication. With sufficient copies of the system, any desired level of fault-tolerance may be achieved. Fault detection and correction are especially simple and may be implemented by majority voters. The main drawback of MR is that it does not take advantage of the specific structure of a problem and thereby requires a substantial amount of redundancy relative to the protection provided. MR will certainly continue to be widely used, especially in the protection of general purpose computation.

2.3 Arithmetic Codes

A more efficient method of adding redundancy to computation is via an error-correcting code. The basic idea is as follows. Redundancy is added to the representation of data within the system, and the system is modified to operate on the encoded data. During error-free operation, the state of the system remains within a fixed subset of valid states. If an error occurs, the system state is perturbed, and the redundancy is used to detect and correct errors, much like an error-correcting code for a communication system.

Standard error-correcting codes can be applied only to protect systems in which the input matches the output. In general, these codes cannot be used to protect computation because their distance structure is destroyed during computation. They have been successfully used to protect certain components of computer systems, such as buses and storage devices, where reliability is needed and no computation is performed. Encoding, and the error detection and correction are usually performed by dedicated hardware in the memory and disk/tape controllers rather than by the CPU.

Arithmetic codes are a class of error-correcting codes that are capable of protecting certain binary operations. Let $\Box$ denote an arbitrary binary operation applied to a pair of operands g_1 and g_2. The result r is given by

$$r = g_1 \Box g_2. \tag{2.1}$$

For example, g_1 and g_2 could be integers, and $\Box$ could be integer addition, subtraction, or multiplication.

In an arithmetic code, we first encode g_1 and g_2 to obtain operand codewords

$$\begin{aligned} \tilde{g}_1 &= \phi(g_1) & (2.2) \\ \tilde{g}_2 &= \phi(g_2). & (2.3) \end{aligned}$$

Then a different operation $\bigcirc$ is applied to the codewords to yield the encoded result

$$h = \tilde{g}_1 \bigcirc \tilde{g}_2. \tag{2.4}$$

We assume that the system computing (2.4) is not reliable and that an error e could arise during computation. The faulty result is given by

$$h' = \mu(\tilde{g}_1, \tilde{g}_2, e) \tag{2.5}$$

where μ models the manner in which errors affect the result. We denote by $\mathcal{H}$ the set of all possible results, and denote by $\mathcal{H}_V$ the subset of $\mathcal{H}$ corresponding to valid, error-free results. If $h' \in \mathcal{H}_V$ then we declare that no errors occurred during computation. Otherwise, if $h' \notin \mathcal{H}_V$, we declare that an error occurred, and correct the error using the mapping α,

$$h = \alpha(h'). \tag{2.6}$$

Finally, we decode h in order to obtain the desired result,

$$r = \phi^{-1}(h). \tag{2.7}$$

The central motivation behind the use of arithmetic codes is to provide robust computation at a lower cost than modular redundancy. Modular redundancy is always a viable alternative to protecting any computation and thus it serves as a benchmark for evaluating the efficiency of arithmetic codes. Another important observation is that arithmetic codes protect the bulk, but not the entirety, of computation. Functions such as error detection and correction and decoding the final result are usually assumed to be robust. If necessary, these functions may be protected by modular redundancy. When evaluating the overall efficiency of an arithmetic code, the cost of functions protected by modular redundancy should be weighted accordingly.

All existing arithmetic codes can be placed in this framework and described by mappings ϕ and σ, operations $\Box$ and $\bigcirc$, and an error process μ. When designing an arithmetic code, we are given a set of operands, and a binary operation, $\Box$, and must determine an encoding ϕ and operation $\bigcirc$ to be applied to the codewords. Furthermore, ϕ must have useful error detecting and correcting properties and must protect against the expected types of errors arising during the computation of $\bigcirc$. Also, computing ϕ and $\bigcirc$ and performing the error detection and correction should not require substantially more computation than computing

$\Box$ alone. Designing practical arithmetic codes is a difficult task.

Arithmetic codes can be classified as being either systematic or nonsystematic.

Definition 1. A systematic error-correcting code is defined as one in which a codeword is composed of two distinct parts: the original unencoded operand g, and a *parity* or *checksum* symbol t derived from g [14].

We will denote the mapping used to generate parity symbols by θ. A systematic codeword has the form

$$\tilde{g} = \phi(g) = [g, t] = [g, \theta(g)] \tag{2.8}$$

where $[g, t]$ denotes the pair consisting of the elements g and t. A nonsystematic code is defined as one which is not systematic. Systematic codes are desirable since results can be easily extracted from codewords without additional processing.

Systematic codes can be further classified as being either separate or nonseparate depending upon how computation is performed on the codewords.

Definition 2. A systematic code is separate if the operation $\bigcirc$ applied to codewords corresponds to componentwise operations on the information and parity symbols: $\Box$ applied to the information symbols, and $\Diamond$ is applied to the parity symbols.

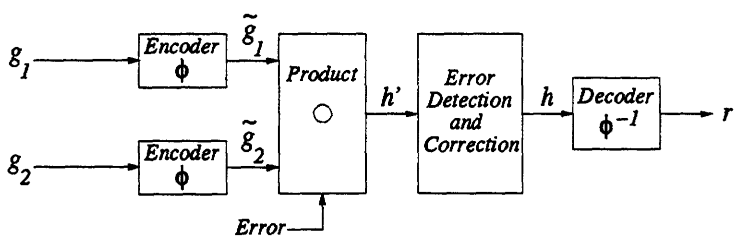
Thus in a separate code,

$$\tilde{g}_1 \bigcirc \tilde{g}_2 = [g_1, t_1] \bigcirc [g_2, t_2] = [g_1 \Box g_2, t_1 \Diamond t_2]. \tag{2.9}$$

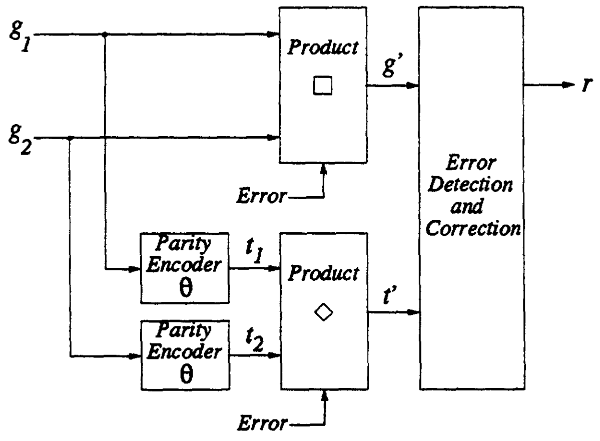
In a nonseparate code, interaction between operands and parity occurs.

In summary, there are three types of arithmetic codes: nonsystematic, systematic-separate, and systematic-nonseparate. Figure 2-2 summarizes these coding schemes and shows possible interaction between information and parity symbols.

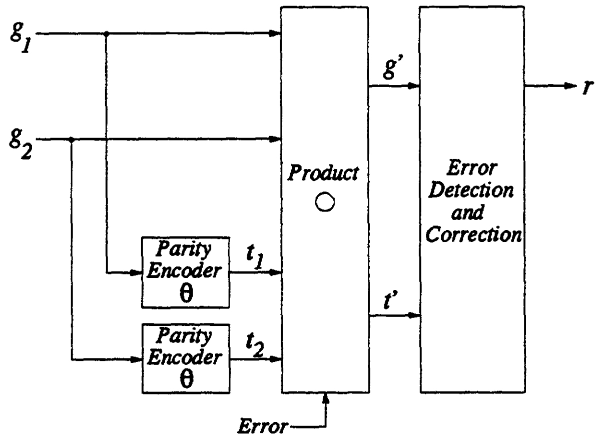
The first generation of arithmetic codes protected integer arithmetic and they relied on the same concepts of distance as binary error-correcting codes. We will now briefly discuss these codes, and our emphasis will be on describing applicable coding schemes ϕ, and on revealing the form of the redundant information. We will not concern ourselves with details of the design procedures or with the process of choosing a specific code.



(a) Nonsystematic



(b) Systematic-Separate



(c) Systematic-Nonseparate

Figure 2-2: Diagram showing the interaction between information and parity symbols for the three different classes of arithmetic codes.

2.3.1 aN Codes

aN codes are nonsystematic arithmetic codes capable of protecting integer addition and subtraction. An operand g is encoded as follows:

$$\tilde{g} = \phi(g) = gN \tag{2.10}$$

where N is an integer. N is referred to as the generator or check base of the code. The set of all possible results $\mathcal{H}$ consists of the set of integers while the subset of valid results consists of multiples of N.

If the operation we wish to protect, $\Box$, is integer addition (subtraction), then the corresponding operation on the codewords, $\bigcirc$, is also integer addition (subtraction). It is easily verified that the codewords are closed under this operation and that a valid result will be a multiple of N.

The performance of aN codes depends on the choice of the multiplier N. In general, additive errors which are a multiple of N cannot be detected. aN codes can also be defined on a subset of integers, and error correction may be possible. General design guidelines are found in [6].

2.3.2 $\langle aN \rangle_M$ Codes

$\langle aN \rangle_M$ codes are systematic-nonseparate codes capable of protecting integer addition, subtraction, and multiplication [6]. They are best described by an example.

Assume that we wish to protect modulo 8 addition, subtraction, and multiplication of the set of integers $\{0, 1, \ldots, 7\}$. Consider the encoding $\tilde{g} = \phi(g) = \langle 25g \rangle_{40}$. This is tabulated below showing both the decimal and binary representations of the numbers.

Decimal g	Binary g	Decimal $\tilde{g}$	Binary $\tilde{g}$
0	000	0	000000
1	001	25	011001
2	010	10	001010
3	011	35	100011
4	100	20	010100
5	101	5	000101
6	110	30	011110
7	111	15	001111

It can be seen that the three least significant bits (rightmost) of the codewords correspond to the information bits. Thus the code is systematic.

The operation $\bigcirc$ applied to the codewords is the same as the operation applied to the information symbols except that now it is performed modulo 40. For example, if modulo 8 multiplication is to be protected, modulo 40 multiplication is performed on the codewords. In computing $\bigcirc$, information and parity symbols interact, and thus the code is nonseparate.

The general form of this code is $\tilde{g} = \langle gN \rangle_M$, and it is systematic for only certain choices of N and M.

2.3.3 Integer Residue Codes

Residue codes are systematic-separate arithmetic codes capable of protecting integer addition, subtraction, and multiplication. Parity symbols are encoded as follows:

$$t_1 = \langle g_1 \rangle_N \tag{2.11}$$

$$t_2 = \langle g_2 \rangle_N \tag{2.12}$$

where N is an integer and $\langle x \rangle_N$ denotes the remainder when x is divided by N.

Let $\Box$ denote the operation that we wish to protect (integer addition, subtraction, or multiplication). The operation $\Diamond$ applied to the parity symbols is the same as $\Box$ except that it is performed modulo N. Thus, if one desires to protect integer addition, residue addition modulo N is used in the parity channel.

As with aN codes, performance depends on the choice of parameter N. Factors which influence the choice of N include the expected types of hardware errors, and the base of the number system used.

2.3.4 Integer Residue Number Systems

The last type of coding scheme that we will consider is called an integer residue number system (RNS). It is a nonsystematic code and is similar in some respects to a residue checksum.

Let $\mathbb{Z}_M$ denote the set of integers $\{0, 1, \ldots, M-1\}$. Assume that we know *a priori* that the result of computation lies in $\mathbb{Z}_M$. Assume further that M can be written as the product of N co-prime factors $M = \prod_{j=1}^{N} m_j$. An RNS utilizes the isomorphism between the ring of integers modulo M and the direct sum of the N smaller rings of integers modulo m_k. Every element $a \in \mathbb{Z}_M$ is uniquely represented by the N-tuple of residues

$$\{a_1, a_2, \ldots, a_N\} \qquad \text{where} \qquad a_j = \langle a \rangle_{m_j} . \tag{2.13}$$

We can perform arithmetic on integers by manipulating their residue representations. Arithmetic is similar to that performed in the parity channel of a systematic-separate residue checksum code. For example, to perform the addition, $r = a + b$, we independently compute the N residue sums $r_j = \langle a_j + b_j \rangle_{m_j}$. Similarly for subtraction and multiplication. We reconstruct the result from the residues r_j using the Chinese Remainder Theorem (CRT) [15].

Fault-tolerance is added to a RNS by adding C redundant moduli $m_{N+1}, \ldots, m_{N+C}$ and keeping the dynamic range limited to M. These redundant moduli must be co-prime to each other and to the original N moduli. The encoding for a becomes

$$\phi(a) = \{a_1, a_2, \ldots, a_{N+C}\} \qquad \text{where} \qquad a_i = \langle a \rangle_{m_i} . \tag{2.14}$$

Assume that the redundant moduli are all larger than the original moduli,

$$m_i > m_j \qquad \text{for} \qquad \begin{cases} i = N+1, \ldots, N+C \\ j = 1, \ldots, N. \end{cases} \tag{2.15}$$

Then it can be shown that there is sufficient redundancy to detect α and correct $\beta \leq \alpha$ errors if and only if $\alpha + \beta \leq C$. Many different methods for error detection and correction have been proposed [16, 17, 18, 19, 20]. Most rely either on multiple CRT or mixed radix conversions. Error detection and correction can also be protected by a fault-tolerant mixed radix conversion algorithm [21].

2.3.5 Comparison of Arithmetic Codes

We now make some general statements concerning the three types of arithmetic codes discussed. Each code has several advantages and disadvantages which govern its use. The statements we make hold most of the time, but one notable exception is integer RNS. We discuss this special case last.

The main advantage of systematic codes is that the desired result r is readily available and extra decoding is unnecessary. This may be critical in high-speed systems where the extra delay associated with decoding is unacceptable.

Separate codes are also more desirable than nonseparate ones because they do not require an increase in the system dynamic range. Consider again the $\langle aN \rangle_M$ code described in Section 2.3.2. The original system had a dynamic range of 8, while the redundant computation required a dynamic range of 40. The longer wordlength arithmetic needed to compute $\bigcirc$ decreases the speed of the fault-tolerant system.

The most desirable coding schemes are usually systematic-separate because they do not have any of the drawbacks mentioned above. Also, the original system computing $\square$ is incorporated unaltered, and only the parity channel and error detection and correction systems must be designed.

Integer RNS is an exception to the above rules because its parallel structure results in efficient implementations. Although it is by definition a nonsystematic code, it actually subdivides arithmetic into several smaller operations, each with less arithmetic complexity than the original operation. Integer RNS also has superior fault coverage since the errors which it protects against closely match those arising in actual systems.

2.4 Algorithm-Based Fault-Tolerance

In this section we discuss a class of high-level arithmetic codes referred to as Algorithm-Based Fault-Tolerance (ABFT). This technique differs from the arithmetic codes presented in the last section in two important aspects. Firstly, the data encoded can be real or complex numbers. The coding schemes which we have discussed thus far have all dealt with either binary or integer data, which are exact number systems. Error-correcting codes, however, may be defined over any field, and in ABFT they are extended to include real or complex data. Secondly, entire sequences or arrays of data are encoded instead of just individual operands. Just as error-correcting codes with low overhead encode sequences of information symbols, low-overhead fault-tolerant computation may be achieved by encoding sequences of operands. The arithmetic codes which we have studied up until now have have only encoded individual operands.

An important motivation for the development of high-level arithmetic codes is the use of parallel, multiprocessor architectures for solving large computational tasks. These systems are amenable to protection by ABFT for the following three reasons:

(i) In these systems, a single operation is usually applied to large amounts of data in a regular fashion. This regularity allows the development of high-level arithmetic codes whose distance structure is preserved during computation.

(ii) The resulting codes are well-suited to the types of errors that occur in multiprocessor systems, such as complete failure of a single or multiple processors.

(iii) In these architectures, there is some flexibility in the partitioning and scheduling of the algorithm among the various processors. This flexibility allows the algorithm to be modified in order to accommodate the encoded data.

2.4.1 Codes for the Transmission of Real Numbers

The first codes for protecting sequences of real or complex data were developed by Marshall [22]. While only concerned with protecting data transmission, he prepared the way for higher-level arithmetic codes. His main contribution was to show that error-correcting codes are not restricted to finite fields, but can be defined over any field. He demonstrated that many codes, such as BCH codes, have direct counterparts defined using real or complex data.

2.4.2 Codes for Matrix Operations

Several arithmetic codes exist for protecting matrix operations. We describe these in the order in which they were developed.

Matrix Checksum Codes

The first arithmetic codes for protecting computation with real or complex data were developed by Huang [23, 24]. He introduced the term Algorithm-Based Fault-Tolerance because he realized that adding redundancy to data requires a corresponding change in the algorithm to operate on the encoded data. He applied a simple arithmetic checksum to protect certain matrix operations. Let $\boldsymbol{A}$ be an $M \times N$ matrix of real or complex data. $\boldsymbol{A}$ is encoded in any of three different ways depending upon which operation is to be protected. The encoding schemes are as follows:

1) column checksum

$$\phi_1(\boldsymbol{A}) = \left[\begin{array}{c} \boldsymbol{I} \\ \hline \underline{e}_c^T \end{array}\right] \boldsymbol{A} = \left[\begin{array}{c} \boldsymbol{A} \\ \hline \underline{e}_c^T \boldsymbol{A} \end{array}\right]$$

2) row checksum

$$\phi_2(\boldsymbol{A}) = \boldsymbol{A}\,[\boldsymbol{I} \mid \underline{e}_r] = [\boldsymbol{A} \mid \boldsymbol{A}\underline{e}_r]$$

3) full checksum

$$\phi_3(\boldsymbol{A}) = \left[\begin{array}{c} \boldsymbol{I} \\ \hline \underline{e}_c^T \end{array}\right] \boldsymbol{A}\,[\boldsymbol{I} \mid \underline{e}_r] = \left[\begin{array}{c|c} \boldsymbol{A} & \boldsymbol{A}\underline{e}_r \\ \hline \underline{e}_c^T \boldsymbol{A} & \underline{e}_c^T \boldsymbol{A}\underline{e}_r \end{array}\right]$$

where $\underline{e}_c^T$ is a $1 \times M$ row vector $\underline{e}_c^T = [1\ 1 \cdots 1]$ and $\underline{e}_r^T$ is a $1 \times N$ row vector $\underline{e}_r^T = [1\ 1 \cdots 1]$. These codes are systematic since the original matrix $\boldsymbol{A}$ is a submatrix of its encoded forms. Codes ϕ_1 and ϕ_2 have a minimum distance of 2 while code ϕ_3 has a minimum distance of 4.

The distance structure of these codes is preserved for various arithmetic operations.

Assume that $\boldsymbol{B}$ is another matrix of appropriate dimension. Then

1) Matrix Addition:

$$\begin{aligned}\phi_1(\boldsymbol{A}) \pm \phi_1(\boldsymbol{B}) &= \phi_1(\boldsymbol{A} \pm \boldsymbol{B})\\ \phi_2(\boldsymbol{A}) \pm \phi_2(\boldsymbol{B}) &= \phi_2(\boldsymbol{A} \pm \boldsymbol{B})\\ \phi_3(\boldsymbol{A}) \pm \phi_3(\boldsymbol{B}) &= \phi_3(\boldsymbol{A} \pm \boldsymbol{B})\end{aligned}$$

2) Scalar Multiplication:

$$\begin{aligned}\alpha\phi_1(\boldsymbol{A}) &= \phi_1(\alpha\boldsymbol{A})\\ \alpha\phi_2(\boldsymbol{A}) &= \phi_2(\alpha\boldsymbol{A})\\ \alpha\phi_3(\boldsymbol{A}) &= \phi_3(\alpha\boldsymbol{A})\end{aligned}$$

where α is a real or complex number

3) Matrix Multiplication:

$$\phi_1(\boldsymbol{A})\phi_2(\boldsymbol{B}) = \phi_3(\boldsymbol{AB})$$

4) Matrix transposition:

$$\begin{aligned}\phi_1(\boldsymbol{A})^T &= \phi_2(\boldsymbol{A}^T)\\ \phi_2(\boldsymbol{A})^T &= \phi_1(\boldsymbol{A}^T)\\ \phi_3(\boldsymbol{A})^T &= \phi_3(\boldsymbol{A}^T)\end{aligned}$$

5) LU decomposition:

If $\boldsymbol{A}$ is LU-decomposable, $\boldsymbol{A} = \boldsymbol{LU}$, where $\boldsymbol{L}$ is lower triangular and $\boldsymbol{U}$ is upper triangular, then:

$$\phi_3(\boldsymbol{A}) = \phi_1(\boldsymbol{L})\phi_2(\boldsymbol{U})$$

where $\phi_1(\boldsymbol{L})$ is a lower triangular column checksum matrix and $\phi_2(\boldsymbol{U})$ is an upper triangular row checksum matrix.

Since these codes have a small minimum distance, Huang applied them to fully parallel architectures in which each processor calculates only a single element of the result matrix. Hence with codes ϕ_1 and ϕ_2, any single processor failure can be detected. While with code ϕ_3, any single processor failure can be detected and corrected.

Under the operations of matrix addition, multiplication by a scalar, and transposition, the codes are separate since no interaction between information and parity occurs. Under matrix multiplication or LU decomposition, the codes are nonseparate. To understand why, multiply a column checksum matrix and a row checksum matrix to obtain a full checksum matrix

$$\phi_1(A)\phi_2(B) = \left[\begin{array}{c} A \\ \hline \underline{e}_c^T A \end{array}\right] [B \mid B\underline{e}_r] = \left[\begin{array}{c|c} AB & AB\underline{e}_r \\ \hline \underline{e}_c^T AB & \underline{e}_c^T AB\underline{e}_r \end{array}\right] = \phi_3(AB). \tag{2.16}$$

The matrices in the upper right, $AB\underline{e}_r$, and lower left, $\underline{e}_c^T AB$, of the result are products of information and parity, and thus this is a nonseparate code.

Weighted Matrix Checksum Codes

The codes developed by Huang are simplistic, and have a small minimum distance. Jou extended Huang's work by adding multiple weighted checksums to rows and columns resulting in codes with a larger minimum distance [25, 26, 27]. The $M \times N$ matrix A is encoded as follows:

1) weighted column checksum

$$\phi_1^w(A) = \left[\begin{array}{c} I \\ \hline E_c^T A \end{array}\right] A = \left[\begin{array}{c} A \\ \hline E_c^T A \end{array}\right]$$

2) weighted row checksum

$$\phi_2^w(A) = A[I \mid E_r] = [A \mid AE_r]$$

3) weighted full checksum

$$\phi_3^w(A) = \left[\begin{array}{c} I \\ \hline E_c^T \end{array}\right] A[I \mid E_r] = \left[\begin{array}{c|c} A & AE_r \\ \hline E_c^T A & E_c^T AE_r \end{array}\right]$$

where E_c^T is a $C \times M$ matrix and E_r is a $N \times D$ matrix. E_c^T and E_r serve the same purpose as the encoding matrix of a linear error-correcting code. If E_c^T and E_r are properly chosen, ϕ_1^w and ϕ_2^w will have a minimum distance of $C + 1$ and $D + 1$ respectively. Similarly, ϕ_3^w can have a minimum distance as large as $(C + 1)(D + 1)$. Weighted checksum codes can be applied to protect the same operations as matrix checksum codes. The large minimum distance permits the detection and correction of multiple errors.

One important issue that was never properly treated by Huang and Jou was that of computational noise. Error-correcting codes require exact arithmetic, and the computational noise inherent to floating point systems corrupts results slightly, and appears as small errors in the output. Huang and Jou glossed over this issue by ignoring all small errors. Later, Nair developed optimal weighted checksum codes which minimized the effects of quantization noise [28, 29, 30]. The design procedure is a constrained optimization, and a tradeoff is made between the ability to detect small errors and quantization noise minimization. Nair also gave a procedure for summing large vectors which minimizes quantization noise. To implement this algorithm efficiently, however, requires the use of a nonstandard ALU. Also, although minimizing quantization noise, Nair does not deal with its effects in a proper manner.

Other Developments in ABFT for Matrix Operations

Several other papers have been written in the area of ABFT for matrix operations, and all rely upon the coding schemes mentioned above. We briefly describe these papers. Anfinson related Jou's work directly to linear error-correcting codes and showed that the coding schemes involved are essentially equivalent [31]. However, one important limitation arises in real number systems: lookup tables cannot be used to correct errors because the set of possible errors is infinite. Also, multiple error correction requires solving an overdetermined nonlinear system. Anfinson also gives a specific code with minimum distance 4 based on a Vandermole matrix that permits simplified error detection and correction.

Banerjee experimentally evaluated the fault coverage and computational overhead of Jou's weighted checksum method [32]. Megson implemented Huang's checksum scheme on a triangular processor array [33]. Anfinson applied Jou's weighted checksum scheme to protect processor arrays performing QR decomposition [34]. Luk applies weighted checksums to systems performing LU decomposition, Gaussian elimination with pairwise pivoting, and QR decomposition [35]. He also analyzed the effects of quantization noise in floating point systems. Bliss computed the probability of system failure as well as the probability of undetected errors for a variety of ABFT coding schemes and architectures [36, 37]. A good summary of papers dealing with ABFT for matrix operations is found in [1].

2.4.3 Codes for Linear Transformations

Musicus and Song proposed a weighted checksum code for protecting linear transformations [38, 39, 40]. Assume that the linear transformation $\boldsymbol{F}$ must be computed N times using different sets of data $\underline{x}_i$ yielding results $\underline{y}_i$,

$$\underline{y}_i = \boldsymbol{F}\underline{x}_i \qquad \text{for} \qquad i = 1, \ldots, N. \tag{2.17}$$

This can be written as a matrix-matrix multiply,

$$\boldsymbol{Y} = \boldsymbol{F}\boldsymbol{X} \tag{2.18}$$

where $\boldsymbol{Y}$ consists of the column vectors $\underline{y}_i$, and $\boldsymbol{X}$ consists of the column vectors $\underline{x}_i$,

$$\boldsymbol{Y} = \begin{bmatrix} | & & | \\ \underline{y}_1 & \cdots & \underline{y}_N \\ | & & | \end{bmatrix} \qquad \text{and} \qquad \boldsymbol{X} = \begin{bmatrix} | & & | \\ \underline{x}_1 & \cdots & \underline{x}_N \\ | & & | \end{bmatrix}. \tag{2.19}$$

To protect (2.18), Musicus and Song use a systematic-separate error-correcting code as shown in Figure 2-3. They compute a parity matrix $\boldsymbol{X}_p$ from $\boldsymbol{X}$ by post-multiplying by an encoding matrix $\boldsymbol{\Theta}$,

$$\boldsymbol{X}_p = \theta(\boldsymbol{X}) = \boldsymbol{X}\boldsymbol{\Theta}. \tag{2.20}$$

$\boldsymbol{\Theta}$ is determined in a manner similar to choosing the encoding matrix for a systematic linear error-correcting code. Then the main computation $\boldsymbol{Y} = \boldsymbol{F}\boldsymbol{X}$ along with the parity computation $\boldsymbol{Y}_p = \boldsymbol{F}\boldsymbol{X}_p$ are performed.

To detect errors, they compute the syndrome matrix

$$\boldsymbol{S} = \boldsymbol{Y}_p - \boldsymbol{Y}\boldsymbol{\Theta}. \tag{2.21}$$

Substituting in for $\boldsymbol{Y}$ and $\boldsymbol{Y}_p$, we find that

$$\boldsymbol{S} = \boldsymbol{F}(\boldsymbol{X}\boldsymbol{\Theta}) - (\boldsymbol{F}\boldsymbol{X})\boldsymbol{\Theta}. \tag{2.22}$$

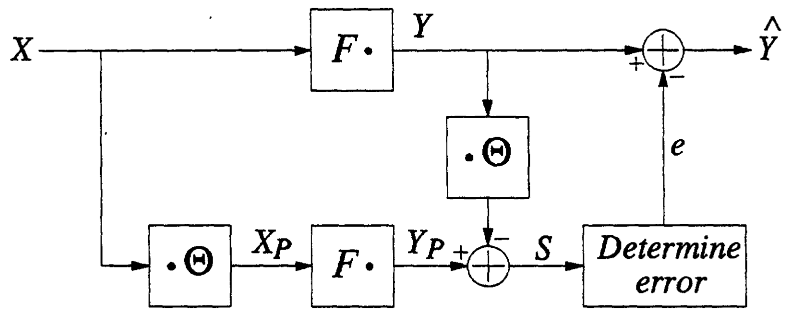


Figure 2-3: Underlying structure of the systematic-separate coding scheme proposed by Musicus and Song to protect linear transformations.

Since matrix multiplication is associative, this equals zero if no errors have occurred. If the syndrome is nonzero, then its value can be used to correct the error. Note that the placement of parentheses in (2.22) is critical since it reduces the amount of computation in the parity channel significantly. (It is assumed that $\boldsymbol{\Theta}$ is chosen such that the parity $\boldsymbol{X}_p = \boldsymbol{X\Theta}$ has far fewer columns than $\boldsymbol{X}$.)

In addition to demonstrating this coding scheme, Musicus and Song made several other important contributions. Firstly, they developed encoding matrices $\boldsymbol{\Theta}$ containing only small integers that have a good minimum distance. At the time of their paper, this was important since it reduced the computation needed to calculate the parity data. Now, however, arithmetic units and signal processors are optimized for floating point arithmetic, and an integer operation is computationally as expensive as a floating point operation. Thus this contribution is no longer significant. Secondly, and most importantly, they developed a stochastic model of the computation and fault processes. Quantization noise was modeled as additive white noise and a generalized likelihood ratio test used to optimally detect and correct errors. Their method is able to reduce overall quantization noise, and yields more accurate solutions in an error-free system. Thirdly, they provided a detailed error analysis and performed thorough simulations of their system operating under various conditions.

2.4.4 Codes for FFTs

Two authors independently developed fault-tolerance schemes to protect FFTs. Both require fully parallel implementations on butterfly architectures. We mention both here, but only consider one to be a true ABFT implementation.

Choi and Malek proposed a fault-tolerant FFT algorithm that detects errors by recomputing the result using an alternate path through the butterflies [41]. Once an error is detected, the faulty butterfly can be located in a few steps. Although offering algorithm-level fault-tolerance, we do not consider this work to be an example of ABFT since the data is not encoded. Rather, we feel that their work more properly fits in the area of modular redundancy since recomputation is needed to detect errors.

Jou proposed a systematic-separate code to protect FFTs [25, 42]. He considers an N-point FFT with input $x[n]$ and output $X[f]$. He utilizes the following relationship between $x[n]$ and $X[f]$ to detect errors:

$$Nx[0] = \sum_{f=0}^{N-1} X[f]. \tag{2.23}$$

This is essentially equivalent to Huang's matrix checksum code and has a minimum distance of 1.

2.4.5 Codes for Convolution

Redinbo developed a fault-tolerant convolution algorithm that is based on linear cyclic error-correcting codes [43, 44]. Let $a[n]$ and $b[n]$ be P-point sequences which are nonzero for $0 \leq n \leq P-1$. The linear convolution, denoted by $c[n] = a[n] * b[n]$, results in a $Q = 2P-1$ point sequence $c[n]$ as follows,

$$c[n] = \sum_{i=0}^{P-1} a[i]b[n-i] \qquad \text{for} \qquad n = 0, 1, \ldots, Q-1. \tag{2.24}$$

Redinbo represents $a[n]$ by the polynomial $a(x)$ as follows:

$$a(x) = \sum_{i=0}^{P-1} a[i]x^i \tag{2.25}$$

and in a similar manner represents $b[n]$ and $c[n]$ by $b(x)$ and $c(x)$. The convolution $c[n] = a[n] * b[n]$ corresponds to the polynomial multiplication $c(x) = a(x)b(x)$.

To add fault-tolerance, Redinbo computes parity symbols from the operands as follows:

$$t_a(x) \;=\; \left\langle x^C a(x) \right\rangle_{g(x)} \tag{2.26}$$

$$t_b(x) = \left\langle x^C b(x) \right\rangle_{g(x)} \qquad (2.27)$$

where $g(x)$ is called the generator polynomial, and $C = \deg g(x)$. Then he computes the desired convolution $c(x) = a(x)b(x)$ and then a product involving the parity symbols,

$$t_c(x) = \langle f(x) t_a(x) t_b(x) \rangle_{g(x)} \qquad (2.28)$$

where $f(x) = \left\langle x^Q \right\rangle_{g(x)}$. This code is equivalent to a BCH code defined over the field of real or complex numbers, and its error detecting and correcting properties depend upon the generator $g(x)$. In general, the code is capable of detecting up to $\deg g(x)$ erroneous output samples in the result, where $\deg g(x)$ denotes the degree of $g(x)$. Existing techniques may be used to choose the generator polynomial $g(x)$, and fast algorithms for detecting and correcting errors exist [14].

2.5 Summary

This chapter discussed existing approaches to protecting computation based on modular redundancy and arithmetic codes. The main advantage of modular redundancy is its ease of use and universal applicability. However, it is much less efficient than techniques based on arithmetic codes. Arithmetic codes add redundancy in a manner similar to error-correcting codes for protecting communication channels, and offer advantages in terms of lower redundancy and better fault coverage.

The main drawback to applying arithmetic codes is that the code must be tailored to the specific operation being performed, and codes exist for only a limited number of operations. In the following chapters, we will study in detail the problem of designing an arithmetic codes. Our goal is to develop a general design procedure for generating useful arithmetic codes for a large class of operations.

Chapter 3

Set-Theoretic Framework

In this chapter we develop a general framework that is able to characterize a wide variety of fault-tolerant system. The framework is based on set theory, and its purpose is to extract common features shared by all fault-tolerant systems. Although not a constructive procedure, the framework is useful for determining if sufficient redundancy exists in order to perform error detection and correction.

We first make some basic definitions using set theory, and model computation as a mapping from a set of operands to a set of results. The effects of errors are then included and a decomposition of a robust system into a cascade of three subsystems is presented. We derive requirements on redundancy assuming a general error mechanism such that errors may be detected and corrected. We show that for a certain class of errors, the redundancy in a system is completely characterized by a single integer, analogous to the minimum distance of an error-correcting code. This leads to a simplified analysis of the inherent redundancy of a system. Lastly, we demonstrate how this framework may be applied by studying a system protected by modular redundancy.

3.1 Set-Theoretic Model of Computation

A set $\mathcal{G}$ is an arbitrary collection of elements. For example, $\mathcal{G}$ could contain integers, real numbers, vectors, or any other type of element. The order of set $\mathcal{G}$, denoted by $\mathcal{O}(\mathcal{G})$, equals the number of elements in $\mathcal{G}$ and may be finite or infinite. The set without any symbols is called the empty-set and is denoted by $\emptyset$.

Let $\mathcal{G}$ and $\mathcal{H}$ be two arbitrary, nonempty sets. The Cartesian product of $\mathcal{G}$ and $\mathcal{H}$, denoted by $\mathcal{G} \times \mathcal{H}$, is the set of pairs

$$\mathcal{G} \times \mathcal{H} = \left\{ [g, h] \mid g \in \mathcal{G}, h \in \mathcal{H} \right\}. \tag{3.1}$$

The Cartesian product can be defined for any finite number of sets in a similar manner. Let $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_N$ be N arbitrary nonempty sets. The Cartesian product of these N sets is defined as

$$\mathcal{G}_1 \times \mathcal{G}_2 \times \cdots \times \mathcal{G}_N = \left\{ [g_1, g_2, \ldots, g_N] \mid g_1 \in \mathcal{G}_1, g_2 \in \mathcal{G}_2, \ldots, g_N \in \mathcal{G}_N \right\}. \tag{3.2}$$

Any deterministic operation can be modeled as a mapping τ from a set of inputs $\mathcal{G}$ to a set of outputs $\mathcal{R}$. The mapping τ associates with each input $g \in \mathcal{G}$ a unique output $r \in \mathcal{R}$. We denote the mapping of g to r by $r = \tau(g)$.

This model also encompasses multiple inputs and multiple outputs in a natural manner. If we have M inputs $g_1 \in \mathcal{G}_1, \ldots, g_M \in \mathcal{G}_M$ and N outputs $r_1 \in \mathcal{R}_1, \ldots, r_N \in \mathcal{R}_N$, then we would let $\mathcal{G} = \mathcal{G}_1 \times \cdots \times \mathcal{G}_M$ and $\mathcal{R} = \mathcal{R}_1 \times \cdots \times \mathcal{R}_N$. This model assumes that any combination of inputs and outputs can arise during operation. If only certain combinations of inputs are possible, then we would choose $\mathcal{G}$ to be the subset of $\mathcal{G}_1 \times \cdots \times \mathcal{G}_M$ corresponding to the domain of τ. Similarly, we would choose $\mathcal{R}$ to be the subset of $\mathcal{R}_1 \times \cdots \times \mathcal{R}_N$ corresponding to the range of τ.

Systems with internal state can also be modeled in this manner as well. We simply treat the state as another set of inputs and the updated state as another set outputs. No other changes are necessary.

Now assume that a fault occurs in the system computing τ. A fault is a permanent or intermittent hardware failure that affects the result of computation. We call the net effect of the fault the *error*, and denote it by e. Let $\mathcal{E}_\tau$ denote the set of all possible errors that are caused by a single fault arising during computation of τ. We assume that $\mathcal{E}_\tau$ can be deduced *a priori* by a careful study of the system used to compute τ. The specific error caused by a fault is unknown, but it will be some element of $\mathcal{E}_\tau$. $\mathcal{E}_\tau$ is a function of the hardware architecture and specific computational steps used to compute τ. $\mathcal{E}_\tau$ contains a special element, denoted by 0, which represents the case of no error. We also define $\mathcal{E}_\tau^* = \mathcal{E}_\tau - \{0\}$. We include the effects of errors in our model by letting τ be a mapping from $\mathcal{G} \times \mathcal{E}_\tau$ to $\mathcal{R}$. $\tau(g, e)$ denotes the result of the system given input g and assuming that error e occurred. Note that a fault need not corrupt the result for all inputs. That is, it is possible that $\mu(g, e) = \mu(g, 0)$ for some $g \in \mathcal{G}$ and error $e \neq 0 \in \mathcal{E}_\tau$.

We can further generalize this model and include the effects of multiple errors. If λ errors occur during the computation of τ, then we will denote the output by $\tau(g, e_1, \ldots, e_\lambda)$. We can also describe this in terms of sets as follows. Let $\mathcal{E}_\tau^{(\lambda)} = \mathcal{E}_\tau \times \cdots \times \mathcal{E}_\tau$ where $\mathcal{E}_\tau$ appears λ times in the Cartesian product. Then τ can be described as a mapping from $\mathcal{G} \times \mathcal{E}_\tau^{(\lambda)}$ to $\mathcal{R}$. Defining τ in this manner requires a detailed knowledge of the set of errors $\mathcal{E}_\tau$ and how they affect the result. Also, in choosing $\mathcal{G} \times \mathcal{E}_\tau^{(\lambda)}$ as the domain of τ assumes that any error could potentially arise for any operand. If the hardware architecture dictates that only certain errors can occur for certain combinations of operands, then we would choose the domain of τ to be the subset of $\mathcal{G} \times \mathcal{E}_\tau^{(\lambda)}$ containing possible operand-error combinations.

Definition 3. We say that the system τ is *robust* for C simultaneous errors if

$$\tau(g, e_1, \ldots, e_C) = \tau(g, 0) \qquad \text{for all} \qquad g \in \mathcal{G}, e_j \in \mathcal{E}_\tau. \tag{3.3}$$

Intuitively, a system is robust if it is unaffected by up to C simultaneous errors. Most systems are not robust, and additional steps are required in order to protect them.

3.2 Redundant Computation

We study robust systems by examining how they exploit redundancy. By redundancy we mean additional states that arise during faults and that are used to detect and correct errors. We model a robust system τ as a cascade of three subsystems: a redundant computation unit μ, an error corrector α, and a result decoder σ. This is illustrated in Figure 3-1. μ computes the desired operation, τ, but utilizes a redundant representation of the output. We denote the possibly faulty output of μ by h' and refer to it as the encoded result.

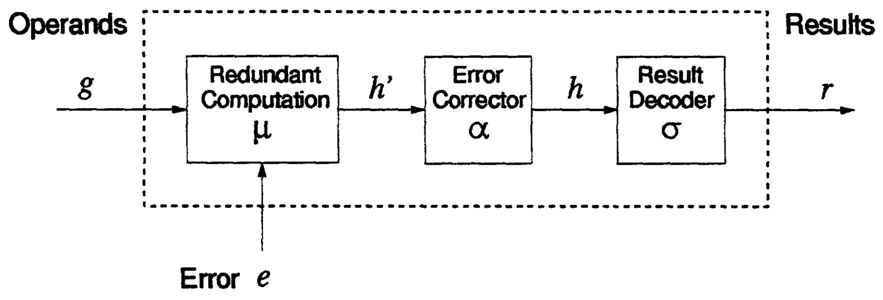


Figure 3-1: Decomposition of a robust system into a cascade of three subsystems: a redundant computation scheme μ, an error corrector α, and a decoder σ.

The redundancy present in h' is used to detect errors, and when possible, to correct them. Let $\mathcal{H}$ denote the set of all possible outputs of μ; those arising from error-free computation as well as those arising during errors. Let $\mathcal{H}_V$ be the subset of $\mathcal{H}$ corresponding to error-free (valid) results,

$$\mathcal{H}_V = \{\mu(g, 0) \mid g \in \mathcal{G}\}. \tag{3.4}$$

We assume that a one-to-one mapping of valid encoded results in $\mathcal{H}_V$ to desired results in $\mathcal{R}$ exists. We denote this mapping by σ and refer to it as the decoder. Located between μ and σ is the error corrector which attempts to correct errors that occurred in μ using only the output h'. α is a mapping from $\mathcal{H}$ onto $\mathcal{H}_V$ whose operation is most easily described as error detection followed by error correction. Error detection utilizes the following test:

$$\begin{array}{rll} \text{If} & h' \in \mathcal{H}_V & \text{we declare that } h' \text{ is valid.} \\ \text{Otherwise, if} & h' \notin \mathcal{H}_V & \text{we declare that } h' \text{ is invalid.} \end{array} \tag{3.5}$$

If no errors are detected then $\alpha(h') = h'$; no correction is required. Otherwise, if an error is detected, α attempts to correct it. If the error is correctable, then α returns the error-free result h. Otherwise, if it is deemed that the error is uncorrectable, then $\alpha(h') = *$ where $*$ signals a detectable, but uncorrectable error. Uncorrectable errors must be handled separately.

The decomposition shown in Figure 3-1 is not always precisely defined, but the rationale behind it is as follows. When an error occurs in a robust system altering its internal state, the effect of the error must be canceled before it reaches the output. That is, the system must contain redundancy at the point of the error, and the system ahead of the error location must perform error correction. In practice, the error corrector α and decoder σ may not be clearly delineated. Often, they are combined into a single mapping which simultaneously corrects errors and decodes the result.

In order for the overall system to be robust, we must make some assumption about the inherent reliability of α and σ. Otherwise an error in either of these systems would be able to corrupt the result. We assume that errors are constrained to μ and that α and σ are inherently robust. In practice, the bulk of computation is performed by μ and the probability of failure of μ far outweighs the probability of either α or σ failing. If necessary, these functions may be protected by modular redundancy.

We begin our examination of robust systems by studying what form of redundancy must exist at the output of the redundant computation μ. For all $h \in \mathcal{H}_V$, define

$$\mathcal{G}_h = \{g \in \mathcal{G} \mid \mu(g,0) = h\}. \tag{3.6}$$

$\mathcal{G}_h$ is the inverse image in $\mathcal{G}$ of the valid result h; all elements in $\mathcal{G}_h$ map to h during error-free computation. Now consider the set

$$\mathcal{H}_{h,\lambda} = \{\mu(g, e_1, \ldots, e_\lambda) \mid g \in \mathcal{G}_h, e_j \in \mathcal{E}_\mu\}. \tag{3.7}$$

This set contains all elements in $\mathcal{H}$ that should have been mapped to h, but that were corrupted by up to λ errors. Note that by definition, $\mathcal{H}_{h,0} = \{h\}$ and $h \in \mathcal{H}_{h,\lambda}$ for all λ. Also, since $0 \in \mathcal{E}_\tau$, $\mathcal{H}_{h,\lambda} \subseteq \mathcal{H}_{h,\lambda+1}$.

Assume that we are interested in detecting up to D errors. Then in order to detect *all* errors, we require that

$$\mu(g, e_1, \ldots, e_D) \notin \mathcal{H}_V \qquad \text{for all} \qquad e_1 \in \mathcal{E}_\mu^*, e_2 \in \mathcal{E}_\mu, \ldots, e_D \in \mathcal{E}_\mu \tag{3.8}$$

where by choosing $e_1 \in \mathcal{E}_\mu^*$, we have assumed that at least one error occurs during the computation of μ. This is an extremely difficult requirement to meet since, in practice, errors do not always corrupt the result. We will concentrate on detecting errors that corrupt the result and will thus require that either

$$\mu(g, e_1, \ldots, e_D) = \mu(g, 0) \tag{3.9}$$

or

$$\mu(g, e_1, \ldots, e_D) \notin \mathcal{H}_V \tag{3.10}$$

for all $g \in \mathcal{G}$ and $e_1 \in \mathcal{E}_\mu, \ldots, e_D \in \mathcal{E}_\mu$. Let $h = \mu(g, 0)$ be the error-free result and let $h' = \mu(g, e_1, \ldots, e_D)$ denote the possibly faulty result. (3.9) implies that $h \in \mathcal{H}_{h,D}$ and (3.10) implies that $\mathcal{H}_{h,D} \cap \mathcal{H}_V = \emptyset$. Combining these yields

$$\mathcal{H}_{h,D} \cap \mathcal{H}_V = \{h\} \qquad \text{for all} \qquad h \in \mathcal{H}_V \tag{3.11}$$

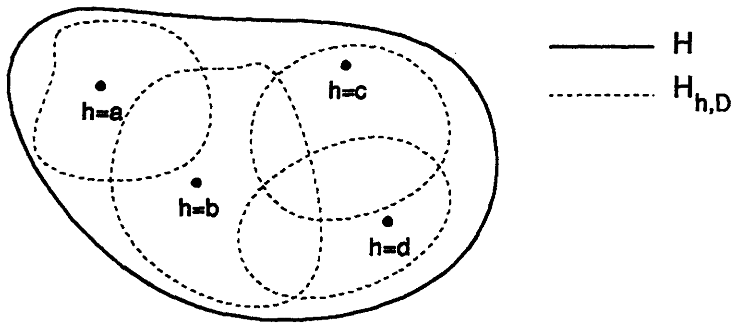
or equivalently,

$$\mathcal{H}_{h_1,D} \cap \{h_2\} = \emptyset \qquad \text{for all} \qquad h_1 \neq h_2 \in \mathcal{H}_V, \tag{3.12}$$

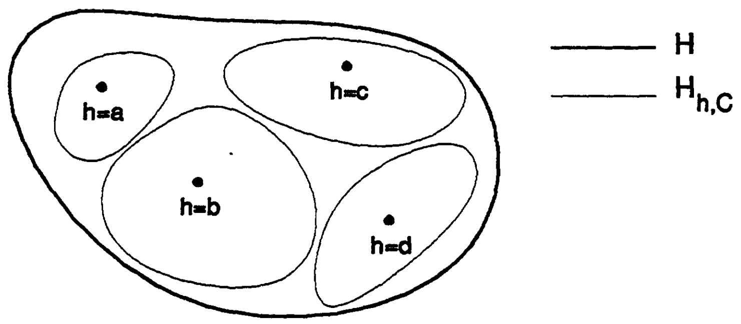
as a necessary and sufficient condition to detect D errors. This requirement is illustrated in Figure 3-2a which shows the output set $\mathcal{H}$ and the error sets $\mathcal{H}_{h,D}$ for four valid results $h = a, b, c,$ and d. In order to be detectable, each subset $\mathcal{H}_{h,D}$ can contain only a single element of $\mathcal{H}_V$, namely h.

Errors that do not corrupt the result are referred to as *latent* errors. They are undetectable solely by examination of the output and require fundamentally different strategies to detect than those we have been discussing. Latent errors within μ may not appear to be a serious problem since they do not corrupt the result. Still, they should be guarded against because, in combination with other errors, they may lead to system failure. Latent errors generally occur in idle parts of the system that are infrequently used. They are especially a problem in systems protected by modular redundancy since a standby system is often switched in to take the place of a faulty system. Errors in a standby system are latent until it is activated.

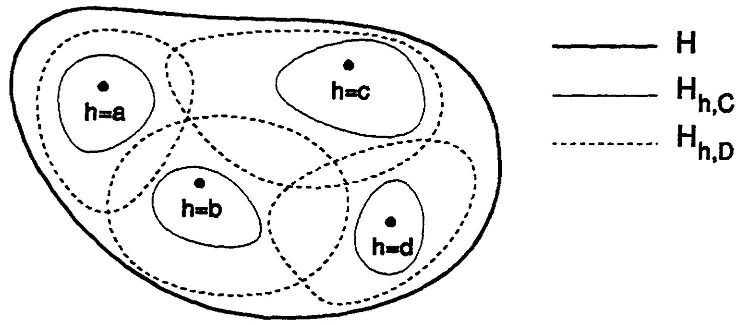
We now examine requirements on redundancy such that C simultaneous errors may be corrected. We are able to correct errors if it is possible to determine the error-free result h given only h'. The set $\mathcal{H}_{h,C}$ contains all elements that should have been mapped to h



(a) Detection of D errors.



(b) Detection and correction of C errors.



(c) Detection of D and correction of C errors.

Figure 3-2: Diagrams showing conceptually the requirements on redundancy for various levels of fault tolerance. The elements $h = a, b, c$, and d denote 4 possible error-free results.

but that were corrupted by up to C errors. The error corrector α must map all elements in $\mathcal{H}_{h,C}$ to h. In order for α to be well-defined, each element in the output set $\mathcal{H}$ must map to a unique element in $\mathcal{H}_V$. This implies that the sets $\mathcal{H}_{h_1,C}$ and $\mathcal{H}_{h_2,C}$ must be disjoint if $h_1 \neq h_2$. That is,

$$\mathcal{H}_{h_1,C} \cap \mathcal{H}_{h_2,C} = \emptyset \qquad \text{for all} \qquad h_1 \neq h_2 \in \mathcal{H}_V. \tag{3.13}$$

This requirement is shown in Figure 3-2b which illustrates the disjoint sets $\mathcal{H}_{h,C}$ for the four values $h = a, b, c,$ and d.

The requirements for error correction are more stringent than those necessary to detect the same number of errors. Assume that we can correct C errors such that (3.13) is true. Then since $h_2 \in \mathcal{H}_{h_2,C}$, this implies that $\mathcal{H}_{h_1,C} \cap \{h_2\} = \emptyset$ for all $h_1 \neq h_2 \in \mathcal{H}_V$. Hence at least C errors can be detected as well.

In the most general case, it is desirable to be able to tradeoff between the number of errors that we want to detect and the number that we want to correct. Assume that we want to be able to detect up to D errors and if C or fewer errors occurred, to correct the result. Otherwise, if between $C+1$ and D errors occurred, the system should recognize this and signal an uncorrectable error $*$. (Note: C must always be less than or equal to D since being able to correct C errors implies being able to detect at least C.) What form of redundancy is required to perform this tradeoff? First of all, to detect D errors, we require that $\mathcal{H}_{h_1,D} \cap \{h_2\} = \emptyset$ for all $h_1 \neq h_2 \in \mathcal{H}_V$. Second of all, to correct C errors, we require that $\mathcal{H}_{h_1,C} \cap \mathcal{H}_{h_2,C} = \emptyset$ for all $h_1 \neq h_2 \in \mathcal{H}_V$. Finally, to distinguish correctable errors (those yielding results in $\mathcal{H}_{h_1,C}$ for some $h_1 \in \mathcal{H}_V$) and uncorrectable errors (those yielding results in $\mathcal{H}_{h_2,D} - \mathcal{H}_{h_2,C}$ for some $h_2 \in \mathcal{H}_V$), we need

$$\left\{ \bigcup_{h_2 \in \mathcal{H}_V} (\mathcal{H}_{h_2,D} - \mathcal{H}_{h_2,C}) \right\} \cap \mathcal{H}_{h_1,C} = \emptyset \qquad \text{for all} \qquad h_1 \in \mathcal{H}_V. \tag{3.14}$$

Simplifying, this becomes

$$(\mathcal{H}_{h_2,D} - \mathcal{H}_{h_2,C}) \cap \mathcal{H}_{h_1,C} = \emptyset \qquad \text{for all} \qquad h_1, h_2 \in \mathcal{H}_V \tag{3.15}$$

or

$$(\mathcal{H}_{h_2,D} - \mathcal{H}_{h_2,C}) \cap \mathcal{H}_{h_1,C} = \emptyset \qquad \text{for all} \qquad h_1 \neq h_2 \in \mathcal{H}_V \tag{3.16}$$

since $(\mathcal{H}_{h_2,D} - \mathcal{H}_{h_2,C}) \cap \mathcal{H}_{h_2,C} = \emptyset$. Expanding (3.16) leads to

$$(\mathcal{H}_{h_2,D} \cap \mathcal{H}_{h_1,C}) - (\mathcal{H}_{h_2,C} \cap \mathcal{H}_{h_1,C}) = \emptyset \qquad \text{for all} \qquad h_1 \neq h_2 \in \mathcal{H}_V. \tag{3.17}$$

Since we assumed that we are able to correct C errors, the second term in the above expression is empty. We find that

$$\mathcal{H}_{h_2,D} \cap \mathcal{H}_{h_1,C} = \emptyset \qquad \text{for all} \qquad h_1 \neq h_2 \in \mathcal{H}_V \tag{3.18}$$

is a sufficient test in order to simultaneously detect D and correct C errors. This redundancy condition is illustrated in Figure 3-2c. The diagram shows the relationship between the sets $\mathcal{H}_{h,C}$ and $\mathcal{H}_{h,D}$ for $h = a, b, c$, and d.

The tradeoff between error detectability and correctability regulated by (3.18) is the most general condition on redundancy which we will study. The former conditions for detectability only, (3.12), and detectability and correctability of the same number of errors, (3.13), are both special cases of (3.18). Setting $C = 0$ in (3.18) yields (3.12), and setting $D = C$ in (3.18) yields (3.13).

The conditions on redundancy for error detection and correction lead to the following error correcting mapping α in a straightforward manner:

$$\alpha(h') = \begin{cases} h & \text{if} \quad h' \in \mathcal{H}_{h,C} \\ * & \text{else.} \end{cases} \tag{3.19}$$

3.3 Symmetric Errors

We have thus far assumed a very general error model in our study of redundancy. We now consider a class of errors that leads to a simplified analysis of the error detecting and correcting potential of a system.

Definition 4. The system μ has a *symmetric error* model if whenever

$$\mu(g, e_1, \ldots, e_\alpha) = \mu(g', e'_1, \ldots, e'_\beta) \tag{3.20}$$

for some $g, g' \in \mathcal{G}$, $e_j, e'_k \in \mathcal{E}_\mu$ and integers α and β, then there exists some $e'_{\beta+1} \in \mathcal{E}_\mu$ such that

$$\mu(g, e_1, \ldots, e_{\alpha-1}) = \mu(g', e'_1, \ldots, e'_{\beta+1}). \tag{3.21}$$

Symmetric errors arise in a wide variety of systems and the redundancy in these systems is completely characterized by a single integer called the minimum distance. We define this as follows.

Definition 5. Let μ be a system with a symmetric error model. Then the minimum distance of μ is the smallest integer t such that

$$\mathcal{H}_{h,t} \cap \mathcal{H}_V \neq \{h\} \qquad \text{for some} \qquad h \in \mathcal{H}_V. \tag{3.22}$$

Conceptually, the minimum distance is a measure of the separation between elements of $\mathcal{H}_V$. It is analogous to the minimum distance of an error-correcting code and can be treated in a similar manner.

Theorem 1. Let μ be a redundant computation that has a symmetric error model and denote the minimum distance of μ by t. Then we can simultaneously detect D and correct C errors if and only if

$$t \geq D + C + 1. \tag{3.23}$$

Proof. We prove the sufficiency of (3.23) by contradiction. Assume that we can't simultaneously detect D and correct C errors. Then according to (3.18), the intersection $\mathcal{H}_{h_2,D} \cap \mathcal{H}_{h_1,C}$ is not empty for some $h_2 \neq h_1 \in \mathcal{H}_V$. Let h be some element in $\mathcal{H}_{h_2,D} \cap \mathcal{H}_{h_1,C}$. Then $h = \mu(g, e_1, \ldots, e_D) = \mu(g', e'_1, \ldots, e'_C)$ for some $g, g' \in \mathcal{G}$ where

$\mu(g,0) = h_2$ and $\mu(g',0) = h_1$ and for some $e_j, e'_k \in \mathcal{E}_\mu$. Since μ has a symmetric error model, we know that there exists some element $e'_{C+1} \in \mathcal{E}_\mu$ such that

$$\mu(g, e_1, \ldots, e_{D-1}) = \mu(g', e'_1, \ldots, e'_{C+1}). \tag{3.24}$$

Applying the symmetric error condition repeatedly, we find that

$$\begin{aligned} \mu(g, e_1, \ldots, e_{D-2}) &= \mu(g', e'_1, \ldots, e'_{C+2}) && (3.25) \\ &\vdots && (3.26) \\ \mu(g, 0) &= \mu(g', e'_1, \ldots, e'_{C+D}) \end{aligned}$$

for some $e'_{C+2}, \ldots, e'_{C+D} \in \mathcal{E}_\mu$. The left hand side of (3.27) equals the error-free result h_2 while the right hand side is some element of $\mathcal{H}_{h_1,C+D}$. Thus,

$$\mathcal{H}_V \cap \mathcal{H}_{h_1,C+D} \neq \{h_1\} \tag{3.27}$$

which implies that $t < D + C + 1$.

To prove necessity assume that $t < D + C + 1$. Thus there exist $h_2 \neq h_1 \in \mathcal{H}_V$ such that $h_2 \in \mathcal{H}_{h_1,C+D}$. This implies that

$$\mu(g, 0) = \mu(g', e'_1, \ldots, e'_{C+D}) \tag{3.28}$$

for some $g, g' \in \mathcal{G}$ and $e'_1, \ldots, e'_{C+D} \in \mathcal{E}_\mu$ where $\mu(g,0) = h_2$ and $\mu(g',0) = h_1$. Applying the symmetric error condition D times, we find that

$$\mu(g, e_1, \ldots, e_D) = \mu(g', e'_1 \ldots, e'_C) \tag{3.29}$$

where $\mu(g, e_1, \ldots, e_D) \in \mathcal{H}_{h_2,D}$ and $\mu(g', e'_1 \ldots, e'_C) \in \mathcal{H}_{h_1,C}$. Therefore $\mathcal{H}_{h_2,D} \cap \mathcal{H}_{h_1,C}$ is not empty and we are thus unable to detect D and correct C errors. $\square$

A system with a symmetric error model can be more easily analyzed because the redun-

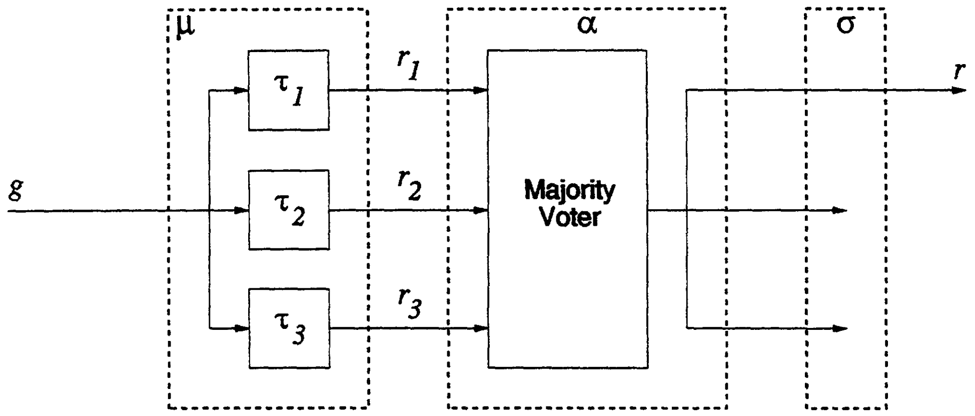


Figure 3-3: Figure illustrating how a system τ protected by triple modular redundancy can be described using the set-theoretic framework.

dancy condition (3.18) does not have to be repeatedly checked for various values of C and D. Instead, once the minimum distance is found, we can use (3.23) to trade-off between error detectability and correctability.

3.4 Example – Triple Modular Redundancy

Triple modular redundancy (TMR) is a good example illustrating how a fault-tolerant system can be analyzed using our framework. A system τ protected by TMR is illustrated in Figure 3-3, and the system has been partitioned according to the standard decomposition of a robust system. Assume that τ has input $g \in \mathcal{G}$ and output $r \in \mathcal{R}$. The redundant computation unit μ is composed of three identical copies of τ, labeled τ_1, τ_2, τ_3, each receiving the same input. The error corrector α detects and corrects errors using a majority voter. Finally, the result is decoded by σ which arbitrarily chooses one of the outputs of α. In practice, α and σ are always combined.

We assume that errors are restricted to μ, and that a single fault in τ_k corrupts only the output r_k. We analyze this system assuming two different computation models:

1. A fault in τ_k leads to a random output $r_k \in \mathcal{R}$.

2. A fault in τ_k forces the output to the fixed value $r_k = \Delta \in \mathcal{R}$.

The first model is more general, and encompasses errors of the second type. Thus, we expect errors obeying the first model to be more difficult to protect against. We will see that this is indeed true.

Assume that the first computation model holds and that faults lead to random outputs. Then the set of all possible outputs equals

$$\mathcal{H} = \left\{ [r_1, r_2, r_3] \mid r_1 \in \mathcal{R}, r_2 \in \mathcal{R}, r_3 \in \mathcal{R} \right\} \tag{3.30}$$

and the subset of error-free outputs is given by

$$\mathcal{H}_V = \left\{ [r, r, r] \mid r \in \mathcal{R} \right\}. \tag{3.31}$$

With this error model, it is possible to transform any valid result $[a, a, a]$ into any other valid result $[b, b, b]$ through a sequence of three errors. Also, $[b, b, b]$ can be changed into $[a, a, a]$ through a similar transformation along the same sequence of outputs. From this, we see that the error model is symmetric and we can thus analyze the behavior of the overall system by determining its minimum distance. The erroneous output sets are given by

$$\begin{aligned}
\mathcal{H}_{[r,r,r],0} &= \left\{ [r, r, r] \right\} && (3.32)\\
\mathcal{H}_{[r,r,r],1} &= \mathcal{H}_{[r,r,r],0} \cup \left\{ [r, r, e], [r, e, r], [e, r, r] \mid e \in \mathcal{R} \right\} && (3.33)\\
\mathcal{H}_{[r,r,r],2} &= \mathcal{H}_{[r,r,r],1} \cup \left\{ [r, e_1, e_2], [e_1, r, e_2], [e_1, e_2, r] \mid e_1, e_2 \in \mathcal{R} \right\} && (3.34)\\
\mathcal{H}_{[r,r,r],3} &= \mathcal{H}. && (3.35)
\end{aligned}$$

By testing when $\mathcal{H}_{[r,r,r],t} \cap \mathcal{H}_V \neq \left\{ [r, r, r] \right\}$, we find that the system has a minimum distance of 3. Thus, there is sufficient redundancy to detect D and correct $C \leq D$ errors where $D + C + 1 \leq 3$. We may do any of the following:

1. Detect 1 and correct 0 errors.

2. Detect 2 and correct 0 errors.

3. Detect and correct 1 error.

The second error model is characteristic of bus, I/O, or power failures. These types of errors often yield outputs that are either all zeros or all ones, and we denote this fixed output state by Δ. Note that Δ is a valid output for some inputs, and faults cannot be detected solely by checking if $r_k = \Delta$. This error model leads to a slightly different system which exhibits a higher level of robustness. The error model is no longer symmetric, and analysis of the system becomes more complicated. We begin by determining relevant subsets of the output. We find that

$$\begin{aligned}
\mathcal{H}_V &= \left\{[r,r,r] \mid r \in \mathcal{R}\right\} && (3.36)\\
\mathcal{H}_{[r,r,r],0} &= \left\{[r,r,r]\right\} && (3.37)\\
\mathcal{H}_{[r,r,r],1} &= \mathcal{H}_{[r,r,r],0} \cup \left\{[r,r,\Delta], [r,\Delta,r], [\Delta,r,r]\right\} && (3.38)\\
\mathcal{H}_{[r,r,r],2} &= \mathcal{H}_{[r,r,r],1} \cup \left\{[r,\Delta,\Delta], [\Delta,r,\Delta], [\Delta,\Delta,r]\right\} && (3.39)\\
\mathcal{H}_{[r,r,r],3} &= \mathcal{H}_{[r,r,r],2} \cup \left\{[\Delta,\Delta,\Delta]\right\}. && (3.40)
\end{aligned}$$

With these subsets available, we can test for error detectability as follows:

$$\begin{aligned}
\mathcal{H}_{[a,a,a],1} \cap \mathcal{H}_V &= \left\{[a,a,a]\right\} && \Longrightarrow \text{ can detect 1 error}\\
\mathcal{H}_{[a,a,a],2} \cap \mathcal{H}_V &= \left\{[a,a,a]\right\} && \Longrightarrow \text{ can detect 2 errors}\\
\mathcal{H}_{[a,a,a],3} \cap \mathcal{H}_V &= \left\{[\Delta,\Delta,\Delta], [a,a,a]\right\} && \Longrightarrow \text{ cannot detect 3 or more errors.}
\end{aligned} \quad (3.41)$$

Similarly, we can test for error correctability. Let $a \neq b$. Then

$$\begin{aligned}
\mathcal{H}_{[a,a,a],1} \cap \mathcal{H}_{[b,b,b],1} &= \emptyset && \Longrightarrow \text{ can correct 1 error}\\
\mathcal{H}_{[a,a,a],2} \cap \mathcal{H}_{[b,b,b],2} &= \emptyset && \Longrightarrow \text{ can correct 2 errors}\\
\mathcal{H}_{[a,a,a],3} \cap \mathcal{H}_{[b,b,b],3} &= \left\{[\Delta,\Delta,\Delta]\right\} && \Longrightarrow \text{ cannot correct 3 or more errors.}
\end{aligned} \quad (3.42)$$

Thus this system can correct up to 2 simultaneous errors. This is in contrast to the first system which only had sufficient redundancy to correct single errors.

Applying (3.18) we are also able to tradeoff error detectability and correctability. We would find that this system is never able to detect or correct more than 2 simultaneous errors.

3.5 Summary

The set-theoretic framework presented in this section is a useful method of analyzing fault-tolerant systems. Our investigation was based entirely on a study of redundancy, and we were unconcerned with the internal workings of the system. This is true in general: the robustness of a system is determined entirely by the redundancy present in the output and the manner in which errors corrupt the result. This suggests an analysis of fault-tolerant systems in terms of the interaction between errors and results. Pursuing this approach, it is possible to determine more elaborate conditions for error detectability and correctability.

This approach, however, do not bring us any closer to the solution of our general problem: constructing robust systems from nonrobust ones. The set-theoretic framework, although a good tool for measuring redundancy, is not a constructive procedure. For the case of arbitrary computation, no general solution, outside of modular redundancy, exists. Instead, we will narrow our focus and study computation that can be modeled as operations in certain algebraic systems. For these types of systems, we give procedures for adding redundancy and efficiently performing error detection and correction.

Chapter 4

Group-Theoretic Framework

We now specialize to the case of computation that can be modeled using group theory. We apply group theory because it encompasses a wide range of arithmetic operations and imposes sufficient structure on computation allowing the form of redundancy to be accurately characterized. Once the framework for groups is complete, we generalize it to rings, fields, and vector spaces, and show that the results for groups carry over completely to these other systems. Background material in group theory helpful to understanding our results is found in [45].

We proceed in the following manner. First, in Sections 4.1 and 4.2, we develop a group-theoretic model of computation and add redundancy by mapping computation to a larger group using an algebraic homomorphism. We then apply results from Chapter 3 in Section 4.3, and give conditions on redundancy such that errors may be detected and corrected. In Section 4.4, the definition of an algebraic homomorphism is relaxed slightly, and a more general mapping called a partial homomorphism is defined. Section 4.5 gives an iterative procedure for choosing a specific coding scheme from a set of possible codes. Then in Section 4.6, we extend the framework to other algebraic systems that have an underlying group structure. Lastly, in Section 4.7, we demonstrate the application of our technique by several examples, and then conclude in Section 4.8.

4.1 Group-theoretic Model of Computation

Definition 6. An Abelian group $G = [\mathcal{G}; \Box, 0_\Box]$ is an algebraic system that consists of a set of elements, $\mathcal{G}$, and a binary operation, $\Box$, called the group product that satisfy the following five properties:

(i) For all $g_1, g_2 \in G$, $g_1 \Box g_2$ is in G. (closure)

(ii) For all $g_1, g_2 \in G$, $g_1 \Box g_2 = g_2 \Box g_1$. (commutativity)

(iii) For all $g_1, g_2, g_3 \in G$, $g_1 \Box (g_2 \Box g_3) = (g_1 \Box g_2) \Box g_3$. (associativity)

(iv) There is an element, $0_\Box$, called the identity, which satisfies $g \Box 0_\Box = g$ for all $g \in G$. (identity)

(v) For every $g \in G$, there is an element in G called the inverse of g, and denoted by g^{-1}, such that $g \Box g^{-1} = 0_\Box$. (inverses)

For example, let $\mathcal{G}$ be the set of integers and let $\Box$ denote integer addition. That is, for $g_1, g_2 \in \mathcal{G}$, $g_1 \Box g_2 = g_1 + g_2$. It can be quickly verified that this is an Abelian group with 0 playing the role of the identity, $0_\Box$, and $-g$ that of g^{-1}. We will frequently use this group, and will denote it by $\mathbb{Z}^+$. An example of an Abelian group containing a finite number of elements is the set of integers $\mathcal{G} = \{0, 1, \ldots, M-1\}$ under modulo M addition. For $g_1, g_2 \in \mathcal{G}$, $g_1 \Box g_2 = \langle g_1 + g_2 \rangle_M$ where $\langle x \rangle_M$ denotes the remainder when x is divided by M. The identity is 0, and g^{-1} is given by $\langle M - g \rangle_M$. We denote this group by $\mathbb{Z}_M^+$.

We assume that the operation we wish to protect can be modeled as a series of $P-1$ products

$$r = g_1 \Box g_2 \Box \cdots \Box g_P \tag{4.1}$$

in some Abelian group $G = [\mathcal{G}; \Box, 0_\Box]$. We define $G^{(P)}$ as the Cartesian product of G with itself P times,

$$G^{(P)} = \underbrace{G \times G \times \cdots \times G.}_{P \text{ times}} \tag{4.2}$$

We place the operation (4.1) in the set-theoretic framework of Chapter 3 by considering r

to be a mapping from $G^{(P)}$ to G,

$$\tau\left([g_1, g_2, \ldots, g_P]\right) = g_1 \Box g_2 \Box \cdots \Box g_P \tag{4.3}$$

where $[g_1, g_2, \ldots, g_P]$ denotes the P-tuple of elements $g_1, g_2, \ldots, g_P$.

We assume that if λ errors occur during computation, then the result is given by

$$\tau([g_1, g_2, \ldots, g_P], e_1, e_2, \ldots, e_\lambda) = g_1 \Box g_2 \Box \cdots \Box g_P \Box e_1 \Box e_2 \Box \cdots \Box e_\lambda \tag{4.4}$$

where e_k is the net effect of the $k^{\underline{th}}$ fault on the result. e_k is an element of $\mathcal{E}_\Box$, the set of all possible single errors, and $\mathcal{E}_\Box$ is a subset of G. This model is a key aspect of our theory, and inherently contains several assumptions. We discuss and justify each in turn. First, it is assumed that errors influence the result in an additive manner through the $\Box$ operation. This additive model closely matches the interaction between faults and errors observed in real systems. For example, soft errors in arithmetic units can be modeled as inducing additive errors in the final result [6]. Additive errors are also the standard model used to describe the interaction between codewords and errors in binary communication systems [14]. The second assumption inherent to our fault-model is that errors are independent of the operands $g_1, g_2, \ldots, g_P$. This is realistic since, in most circumstances, errors result from external faults and do not depend on the specific choice of operands. Still, some data specific errors, such as stuck-at faults [6], are not covered by this model.

4.2 Redundant Computation

The operation shown in (4.4) does not contain any redundancy since every output is a valid result for some set of operands. We add redundancy by mapping computation to another Abelian group, $H = [\mathcal{H}; \bigcirc, 0_\bigcirc]$, of higher order. The additional elements of H will be used to denote erroneous results. We assume that the redundant system has the general form shown in Figure 4-1.

The redundant computation unit functions as follows. Each operand g_k is first encoded

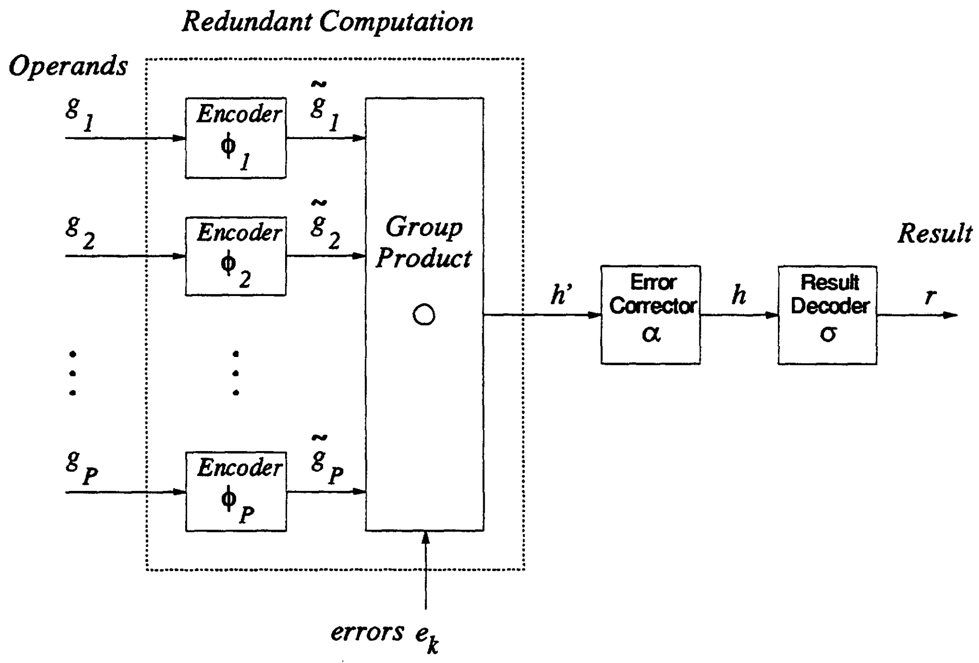


Figure 4-1: Diagram showing the general structure of a redundant fault-tolerant system in which computation can be modeled as a series of group products.

using a mapping ϕ_k to obtain an operand codeword

$$\tilde{g}_k = \phi_k(g_k) \tag{4.5}$$

that is an element of H. In order that no information is lost during encoding, ϕ_k must be one-to-one. Next, we compute the product of codewords in H, and use the same error model developed earlier. We assume that errors affect the result in an additive manner

$$h' = \mu(\,[\tilde{g}_1, \tilde{g}_2, \ldots, \tilde{g}_P]\,, e_1, e_2, \ldots, e_\lambda) = \tilde{g}_1 \bigcirc \tilde{g}_2 \bigcirc \cdots \bigcirc \tilde{g}_P \bigcirc e_1 \bigcirc e_2 \bigcirc \cdots \bigcirc e_\lambda \tag{4.6}$$

where e_k is the net effect of the $k^{\underline{th}}$ fault on the result. e_k is an element of $\mathcal{E}_{\bigcirc}$, a subset of H containing possible single errors. The mappings α and σ have the same basic function as in the set-theoretic framework. α tests for errors in h', and when possible, computes the error-free result h. Then, σ maps h back to G to obtain the desired result,

$$r = \sigma(h). \tag{4.7}$$

The subset of error-free results $\mathcal{H}_V$ is given by

$$\mathcal{H}_V = \{\phi_1(g_1) \bigcirc \phi_2(g_2) \bigcirc \cdots \bigcirc \phi_P(g_P) \mid g_1, g_2, \ldots, g_P \in G\}. \tag{4.8}$$

σ is defined as a one-to-one mapping from $\mathcal{H}_V$ onto G. We assume that errors are restricted to the products $\bigcirc$ and to the encoders ϕ_k, and assume that α, and σ are inherently robust. If necessary α and σ may be protected by modular redundancy.

We have thus far only outlined the basic structure of the system, and will now examine the mappings more closely. We begin by ignoring errors, and determine restrictions on mappings such that the correct result is computed by an error-free system. Ignoring errors and combining (4.1) and (4.5)–(4.7), we find that in order to compute the correct result, the encoders ϕ_k and decoder σ must satisfy

$$g_1 \Box g_2 \Box \cdots \Box g_P = \sigma(\phi_1(g_1) \bigcirc \phi_2(g_2) \bigcirc \cdots \bigcirc \phi_P(g_P)) \quad \text{for all} \quad g_1, g_2, \ldots, g_P \in G. \tag{4.9}$$

Assume that some mappings exist which satisfy (4.9). Then an alternate set of mappings also satisfying (4.9) is given by

$$\begin{aligned} \phi_k'(x) &= h_k \bigcirc \phi_k(x) & (4.10) \\ \sigma'(x) &= \sigma(x \bigcirc h_1^{-1} \bigcirc h_2^{-1} \bigcirc \cdots \bigcirc h_P^{-1}) & (4.11) \end{aligned}$$

where the h_k are arbitrary elements of H. In fact, any encodings ϕ_k which differ by only constant offsets are essentially equivalent. Thus, without any loss of generality, we can assume that the mappings ϕ_k preserve the identity,

$$\phi_k(0_\Box) = 0_\bigcirc \qquad \text{for} \qquad k = 1, 2, \ldots, P. \tag{4.12}$$

We can always select constants h_k and define ϕ_k' and σ' such that this is true.

The decoder σ is defined to be a one-to-one mapping from $\mathcal{H}_V$ onto G. Therefore, σ is invertible, and (4.9) can be written as

$$\sigma^{-1}(g_1 \Box g_2 \Box \cdots \Box g_P) = \phi_1(g_1) \bigcirc \phi_2(g_2) \bigcirc \cdots \bigcirc \phi_P(g_P). \tag{4.13}$$

Setting all operands g_k to the identity $0_\Box$ except g_j, and using our assumption (4.12), we find that $\sigma^{-1}(g_j) = \phi_j(g_j)$. Repeating for all g_j, we find that

$$\sigma^{-1}(g) = \phi_1(g) = \phi_2(g) = \cdots = \phi_P(g). \tag{4.14}$$

Thus the mappings are equivalent, $\sigma^{-1} = \phi_1 = \phi_2 = \cdots = \phi_P$, and from now on we will denote them all by ϕ. (4.13) then becomes

$$\phi(g_1 \Box g_2 \Box \cdots \Box g_P) = \phi(g_1) \bigcirc \phi(g_2) \bigcirc \cdots \bigcirc \phi(g_P) \qquad \text{for all} \qquad g_1, g_2, \ldots, g_P \in G. \tag{4.15}$$

Theorem 2. The constraint on mappings shown in (4.15) is equivalent to

$$\phi(g_1 \Box g_2) = \phi(g_1) \bigcirc \phi(g_2) \qquad \text{for all} \qquad g_1, g_2 \in G. \tag{4.16}$$

Proof. Assume that (4.15) is true, and let $g_3 = 0_\Box, g_4 = 0_\Box, \ldots, g_P = 0_\Box$. Since $\phi_k(0_\Box) = 0_\bigcirc$, we obtain (4.16).

Now assume that (4.16) is true. Then

$$\phi(g_1 \Box (g_2 \Box g_3 \Box \cdots \Box g_P)) = \phi(g_1) \bigcirc \phi(g_2 \Box g_3 \Box \cdots \Box g_P) \tag{4.17}$$
$$\text{for all} \qquad g_1, g_2, \ldots, g_P \in G.$$

Reapplying (4.16) to the term $\phi(g_2 \Box g_2 \Box \cdots \Box g_P)$ on the right hand side in (4.17) yields

$$\phi(g_1) \bigcirc \phi(g_2 \Box g_2 \Box \cdots \Box g_P) = \phi(g_1) \bigcirc \phi(g_2) \bigcirc \phi(g_3 \Box g_4 \Box \cdots \Box g_P). \tag{4.18}$$

Continuing this process yields (4.15). $\Box$

Equation (4.16) is recognized as the defining property of an algebraic *homomorphism.* A homomorphism is a mapping from one algebraic system, G, to another, H, which preserves structure [45]. It is specifically this property of ϕ which we exploit to define redundant computation schemes.

We now examine more closely the form of redundancy introduced by a homomorphism. Let $\tilde{\mathcal{G}} = \{\phi(g) \mid g \in G\}$ denote the set of encoded operands in H. We can show that under $\bigcirc$, $\tilde{\mathcal{G}}$ satisfies the five properties of an Abelian group, and is thus an Abelian group which we denote by $\tilde{G}$.

(i) (closure) Let $\tilde{g}_1, \tilde{g}_2 \in \tilde{G}$. Then $\phi(g_1) = \tilde{g}_1$ and $\phi(g_2) = \tilde{g}_2$ for some $g_1, g_2 \in G$. Consider the product $\tilde{g}_1 \bigcirc \tilde{g}_2 = \phi(g_1) \bigcirc \phi(g_2)$. Since ϕ is a homomorphism, this equals $\phi(g_1 \Box g_2)$, and since $g_1 \Box g_2 \in G$, this implies that $\tilde{g}_1 \bigcirc \tilde{g}_2 \in \tilde{G}$.

(ii) (commutativity) $\tilde{G}$ inherits commutativity from H.

(iii) (associativity) $\tilde{G}$ inherits associativity from H.

(iv) (identity) Since $0_\Box \in G$, $0_\bigcirc$ is in $\tilde{G}$ by our assumption (4.12).

(v) (inverses) Let $\tilde{g} \in \tilde{G}$ and let $\tilde{g} = \phi(g)$ for some $g \in G$. Since G is a group, $g^{-1} \in G$. Then $\phi(g \Box g^{-1}) = \phi(0_\Box) = 0_\bigcirc = \phi(g) \bigcirc \phi(g^{-1}) = \tilde{g} \bigcirc \phi(g^{-1})$. Thus $\phi(g^{-1})$ is the inverse of $\tilde{g}$, and since $g^{-1} \in G$, $\phi(g^{-1}) \in \tilde{G}$.

Since ϕ is one-to-one as well as a homomorphism, the groups G and $\tilde{G}$ are *isomorphic.* They are essentially the same except for the naming of their elements. Given a computation in G, the isomorphism allows us to carry out an analogous computation in $\tilde{G}$.

The subset of valid results in H is then given by

$$\mathcal{H}_V = \left\{ \tilde{g}_1 \bigcirc \tilde{g}_2 \bigcirc \cdots \bigcirc \tilde{g}_P \mid \tilde{g}_1, \tilde{g}_2, \ldots, \tilde{g}_P \in \tilde{G} \right\}. \tag{4.19}$$

Since $\tilde{G}$ is a group, it is closed under $\bigcirc$ and therefore $\mathcal{H}_V = \tilde{G}$.

In summary, to add fault-tolerance to a group G, we map computation to an isomorphic copy, $\tilde{G}$, which is a subgroup of a larger group H. We detect errors by checking if the result is in $\tilde{G}$, and correct them via the mapping α. Then the result is decoded using ϕ^{-1}. These steps are illustrated in Figure 4-2.

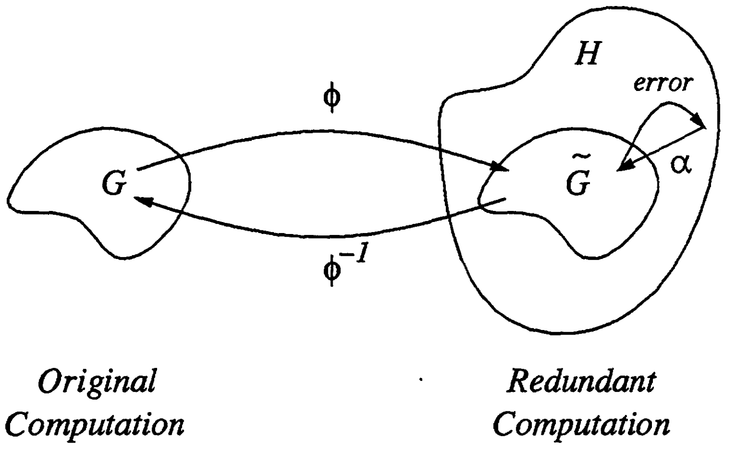


Figure 4-2: Model of a fault-tolerant group operation. Computation in G is mapped to an isomorphic copy $\tilde{G}$ which is a subgroup of a larger group H. Errors force results out of $\tilde{G}$ and are corrected by α.

4.3 Error Detection and Correction

In this section, we determine conditions on redundancy such that errors may be detected and corrected. Using these conditions, we then derive a possible error detection and correction algorithm, and show that it may be reduced to a function of a syndrome. We then give a constructive procedure, based on a quotient group isomorphism, for determining the form of the syndrome mapping.

4.3.1 Redundancy Conditions

We begin with the error model (4.6), and since $\bigcirc$ is associative, we may write

$$h' = (\tilde{g}_1 \bigcirc \tilde{g}_2 \bigcirc \cdots \bigcirc \tilde{g}_P) \bigcirc (e_1 \bigcirc e_2 \bigcirc \cdots \bigcirc e_\lambda) \tag{4.20}$$

$$= h \bigcirc e \tag{4.21}$$

where $h = \tilde{g}_1 \bigcirc \tilde{g}_2 \bigcirc \cdots \bigcirc \tilde{g}_P$ is the error-free result in $\mathcal{H}_V$, and $e = e_1 \bigcirc e_2 \bigcirc \cdots \bigcirc e_\lambda$ is the net effect of all the errors on the result. We can thus consider errors to be applied to the result of an error-free computation. We define $\mathcal{E}_{\bigcirc}^{(\lambda)} = \mathcal{E}_{\bigcirc} \bigcirc \mathcal{E}_{\bigcirc} \bigcirc \cdots \bigcirc \mathcal{E}_{\bigcirc}$ as the product of $\mathcal{E}_{\bigcirc}$ with itself λ times. The error e is some element of $\mathcal{E}_{\bigcirc}^{(\lambda)}$.

Restrictions such that errors are detectable and correctable can be easily derived by applying results from the previous chapter. It can be shown from the associativity of the error model that

$$\mathcal{H}_{h,\lambda} = h \bigcirc \mathcal{E}_{\bigcirc}^{(\lambda)}. \tag{4.22}$$

Substituting this into (3.18), we find that in order to detect D and correct C errors requires that

$$\left(h_2 \bigcirc \mathcal{E}_{\bigcirc}^{(D)}\right) \cap \left(h_1 \bigcirc \mathcal{E}_{\bigcirc}^{(C)}\right) = \emptyset \qquad \text{for all} \qquad h_1 \neq h_2 \in \mathcal{H}_V. \tag{4.23}$$

This may be transformed as follows

$$\begin{aligned}
h_2 \bigcirc e_2 &\neq h_1 \bigcirc e_1 && \text{for all} \quad h_1 \neq h_2 \in \tilde{G}, e_1 \in \mathcal{E}_{\bigcirc}^{(C)}, e_2 \in \mathcal{E}_{\bigcirc}^{(D)} && (4.24)\\
h_1^{-1} \bigcirc h_2 \bigcirc e_2 &\neq e_1 && && (4.25)\\
h \bigcirc e_2 &\neq e_1 && \text{for all} \quad h \neq 0_{\bigcirc} \in \tilde{G}, e_1 \in \mathcal{E}_{\bigcirc}^{(C)}, e_2 \in \mathcal{E}_{\bigcirc}^{(D)} && (4.26)\\
h \bigcirc e_2 &\neq e_1 && \text{for all} \quad h \in \tilde{G}, e_1 \in \mathcal{E}_{\bigcirc}^{(C)} \neq e_2 \in \mathcal{E}_{\bigcirc}^{(D)} && (4.27)\\
h_1^{-1} \bigcirc h_2 \bigcirc e_2 &\neq e_1 && \text{for all} \quad h_1, h_2 \in \tilde{G}, e_1 \in \mathcal{E}_{\bigcirc}^{(C)} \neq e_2 \in \mathcal{E}_{\bigcirc}^{(D)} && (4.28)\\
h_2 \bigcirc e_2 &\neq h_1 \bigcirc e_1 && && (4.29)\\
\tilde{G} \bigcirc e_2 \cap \tilde{G} \bigcirc e_1 &= \emptyset && \text{for all} \quad e_1 \in \mathcal{E}_{\bigcirc}^{(C)} \neq e_2 \in \mathcal{E}_{\bigcirc}^{(D)} && (4.30)
\end{aligned}$$

where we have used the fact that $\tilde{G}$ is a subgroup in steps (4.26) and (4.28). Since $\tilde{G}$ is a subgroup of H, the sets $\tilde{G} \bigcirc e_1$ and $\tilde{G} \bigcirc e_2$ are *cosets* of $\tilde{G}$ in H. It is well-known that two cosets of $\tilde{G}$ in H are either identical or have no elements in common. We can thus treat the cosets as elements rather than sets, and write (4.30) as

$$\tilde{G} \bigcirc e_1 \neq \tilde{G} \bigcirc e_2 \qquad \text{for all} \qquad e_1 \in \mathcal{E}_{\bigcirc}^{(C)} \neq e_2 \in \mathcal{E}_{\bigcirc}^{(D)}. \tag{4.31}$$

In this form, we see that the ability to detect and correct errors implies the uniqueness of the cosets of $\tilde{G}$ in H. This suggests a different method of correcting errors based on first identifying the error e, and then canceling its effect on the result, rather than mapping directly from h' to h. We examine this technique in detail.

4.3.2 Coset-Based Error Detection and Correction

The coset $\tilde{G} \bigcirc e$ contains all possible results that could arise if an error e occurred in the system. (4.31) implies that every detectable error forces the result into a nonzero coset (i.e., the result does not lie in $\tilde{G} \bigcirc 0_{\bigcirc} = \tilde{G}$). Furthermore, every correctable error $e \in \mathcal{E}_{\bigcirc}^{(C)}$ forces the result into a distinct coset,

$$\tilde{G} \bigcirc e_1 \neq \tilde{G} \bigcirc e_2 \qquad \text{for all} \qquad e_1 \neq e_2 \in \mathcal{E}_{\bigcirc}^{(C)} \tag{4.32}$$

and these cosets differ from the cosets of detectable but uncorrectable errors,

$$\tilde{G} \bigcirc e_1 \neq \tilde{G} \bigcirc e_2 \qquad \text{for all} \qquad e_1 \in \mathcal{E}_{\bigcirc}^{(C)} \neq e_2 \in \mathcal{E}_{\bigcirc}^{(D)}. \tag{4.33}$$

Thus, an alternative method of correcting the result is to first determine the net error, if possible,

$$e = \underset{a \in \mathcal{E}_{\bigcirc}^{(C)}}{\arg} \quad h' \in \tilde{G} \bigcirc a. \tag{4.34}$$

If $h' \notin \tilde{G} \bigcirc a$ for all $a \in \mathcal{E}_{\bigcirc}^{(C)}$, then the error is uncorrectable. Otherwise, once the net error is determined, we subtract it from h' to determine the error-free encoded result

$$h = h' \bigcirc e^{-1}. \tag{4.35}$$

The revised error detection and correction test (4.34) works by checking which coset the result lies in. This is an awkward and computationally expensive procedure since we must manipulate sets of elements. We can perform this same procedure more efficiently using a homomorphism ψ from H to another group $T = [\mathcal{T}; \diamond, 0_{\diamond}]$.

Definition 7. Let ψ be a homomorphism from H to T. The set of elements in H which map to the identity in T is called the *kernel* of ψ and is denoted by K_{ψ};

$$K_{\psi} = \{h \in H \mid \psi(h) = 0_{\diamond}\}. \tag{4.36}$$

The kernel of a homomorphism is similar to the nullspace of a linear transformation.

Theorem 3. Let ψ be a homomorphism from H onto T with kernel $K_\psi = \tilde{G}$. Suppose that $h' = h \bigcirc e$ where $h \in \tilde{G}$. Then h' is in the coset $\tilde{G} \bigcirc e$ if and only if $\psi(h') = \psi(e)$.

Proof. Assume that $h' \in \tilde{G} \bigcirc e$. Then $h' = \tilde{g} \bigcirc e$ for some $\tilde{g} \in \tilde{G}$. Applying the homomorphism ψ to h' we find that $\psi(h') = \psi(\tilde{g} \bigcirc e) = \psi(\tilde{g}) \diamond \psi(e) = \psi(e)$ since $\tilde{g} \in K_\psi$. Thus $\psi(h') = \psi(e)$ if $h' \in \tilde{G} \bigcirc e$.

Now assume that $\phi(h') = \psi(e)$. Then $\psi(h') = \psi(\tilde{g}) \diamond \psi(e)$ if and only if $\tilde{g} \in K_\psi = \tilde{G}$. Since ψ is a homomorphism, we may write $\psi(h') = \phi(\tilde{g} \bigcirc e)$. This implies that $h' = \tilde{g} \bigcirc e$ for some $\tilde{g} \in \tilde{G}$. Thus $h' \in \tilde{G} \bigcirc e$. $\square$

The homomorphism ψ condenses all information relevant to error detection and correction into a single element, $\psi(h')$. We will refer to $\psi(h')$ as the *syndrome* of the result h'. Note that $\psi(h') = \psi(h \bigcirc e) = \psi(e)$, and ψ effectively removes any component of the error-free result $h \in \tilde{G}$ from the syndrome.

Theorem 3 shows that a one-to-one correspondence between cosets and syndromes exists. It is convenient to restate the redundancy condition (4.31) in terms of syndromes rather than cosets. We may detect D and correct C errors if and only if

$$\psi(e_1) \neq \psi(e_2) \qquad \text{for all} \qquad e_1 \in \mathcal{E}_{\bigcirc}^{(C)} \neq e_2 \in \mathcal{E}_{\bigcirc}^{(D)}. \tag{4.37}$$

Instead of identifying the error by checking cosets as in (4.34), it can be done using the syndrome as follows. Given the possibly faulty result h', compute $\psi(h')$. Then

$$\begin{array}{rl} \text{If} & \psi(h') = 0_\diamond, \text{ then no errors have occurred, } e = 0_\bigcirc \\ \text{else if} & \psi(h') = \psi(e) \text{ for some } e \in \mathcal{E}_{\bigcirc}^{(C)}, \text{ then error } e \text{ has occurred} \\ \text{else} & \text{an uncorrectable error has occurred.} \end{array} \tag{4.38}$$

If the error is correctable, we would compute the error-free encoded result using (4.35).

If $\mathcal{E}_{\bigcirc}^{(C)}$ contains a small number of elements, then error correction may be performed using a lookup table Λ as follows. Define

$$\Lambda\left(\psi(e)\right) = \begin{cases} e & \text{for all } e \in \mathcal{E}_{\mathcal{O}}^{(C)} \\ * & \text{else.} \end{cases} \tag{4.39}$$

Then, the error may be determined from h' by using $\psi(h')$ as an index into the lookup table, $e = \Lambda\left(\psi(h')\right)$.

Error correction using a lookup table is efficient only when the set of expected errors $\mathcal{E}_{\mathcal{O}}^{(C)}$ is small. In practice, $\mathcal{E}_{\mathcal{O}}^{(C)}$ may be large or even infinite, and in these instances, other techniques must be used. The usual approach is to exploit the structure of $\mathcal{E}_{\mathcal{O}}^{(C)}$ and to invert $\psi(e)$ over the set $\mathcal{E}_{\mathcal{O}}^{(C)}$. An interesting observation is that $\mathcal{O}(T)$ determines the maximum number of distinct errors which may be corrected. Since T contains $\mathcal{O}(T) - 1$ nonzero elements, we may correct, at most, $\mathcal{O}(T) - 1$ different errors.

This method of error correction corrects the result by subtracting the error e from the final result h'. This is computationally efficient only if several products are computed. Otherwise, if only a single product is computed (P=2), then error correction requires as much computation as we originally sought to protect. This is clearly impractical.

4.3.3 Determination of the Syndrome Homomorphism

In order to apply this syndrome-based error correction procedure, we must have some method of determining a suitable homomorphism ψ and group T. Often, ψ and T are readily found by inspection, while in other cases, they may not be so easily identified. In these instances, we propose the following quotient group construction to aid in the identification of ψ and T.

The construction method identifies a homomorphism $\tilde{\psi}$ from H onto another group $\tilde{T} = \left[\tilde{T}; \tilde{\diamond}, \tilde{0}_{\diamond}\right]$ whose kernel equals $\tilde{G}$. We will find that computation in the group $\tilde{T}$ is cumbersome to perform, and by examining $\tilde{T}$, we will recognize that it is isomorphic to a simpler group, in which computation can be more conveniently performed. This simpler group will be denoted T, and it is derived by renaming the elements of $\tilde{T}$. The actual syndrome homomorphism used in practice will be denoted by ψ.

Let $H/\tilde{G}$ denote the collection of cosets of $\tilde{G}$ in H. Each element of $H/\tilde{G}$ is a subset of

H. Let the product of subsets serve as a product in $H/\tilde{G}$. That is, for $\tilde{G} \bigcirc h_i, \tilde{G} \bigcirc h_j \in H/\tilde{G}$,

$$\tilde{G} \bigcirc h_i \bigcirc \tilde{G} \bigcirc h_j = \left\{ h_l \bigcirc h_m \mid h_l \in \tilde{G} \bigcirc h_i, h_m \in \tilde{G} \bigcirc h_j \right\}. \tag{4.40}$$

The set $H/\tilde{G}$ forms an Abelian group under this product. We verify that it satisfies the properties of an Abelian group:

(i) (closure) Let $\tilde{G} \bigcirc h_i$, $\tilde{G} \bigcirc h_j \in H/\tilde{G}$. Then $\tilde{G} \bigcirc h_i \bigcirc \tilde{G} \bigcirc h_j = \tilde{G} \bigcirc \tilde{G} \bigcirc h_i \bigcirc h_j$ by commutativity. Since $\tilde{G}$ is a subgroup, $\tilde{G} \bigcirc \tilde{G} = \tilde{G}$. Thus $\tilde{G} \bigcirc \tilde{G} \bigcirc h_i \bigcirc h_j = \tilde{G} \bigcirc h_i \bigcirc h_j \in H/\tilde{G}$, and we see that $H/\tilde{G}$ is closed.

(ii) (commutativity) $H/\tilde{G}$ inherits commutativity from H.

(iii) (associativity) $H/\tilde{G}$ inherits associativity from H.

(iv) (identity) Consider the element $\tilde{G} \bigcirc 0_{\bigcirc} \in H/\tilde{G}$. For an arbitrary coset $\tilde{G} \bigcirc h \in H/\tilde{G}$, $\tilde{G} \bigcirc h \bigcirc \tilde{G} \bigcirc 0_{\bigcirc} = \tilde{G} \bigcirc h$. Thus $\tilde{G} \bigcirc 0_{\bigcirc}$ is the identity in $H/\tilde{G}$.

(v) (inverses) Let $\tilde{G} \bigcirc h$ be an element of $H/\tilde{G}$. Then $\tilde{G} \bigcirc h^{-1}$ is also an element of $H/\tilde{G}$. Multiplying, we find that $\tilde{G} \bigcirc h \bigcirc \tilde{G} \bigcirc h^{-1} = \tilde{G} \bigcirc 0_{\bigcirc}$ which is the identity in $H/\tilde{G}$. Thus, each element of $H/\tilde{G}$ has an inverse in $H/\tilde{G}$.

The group $H/\tilde{G}$ defined in this manner is called the *quotient group* of H by $\tilde{G}$.

Given $H/\tilde{G}$, we can always define a mapping $\tilde{\psi}$ from H onto $H/\tilde{G}$ by

$$\tilde{\psi}(h) = \tilde{G} \bigcirc h. \tag{4.41}$$

$\tilde{\psi}$ maps h to the coset containing h. Now, consider applying $\tilde{\psi}$ to the product $h_1 \bigcirc h_2$,

$$\begin{aligned} \tilde{\psi}(h_1 \bigcirc h_2) &= \tilde{G} \bigcirc (h_1 \bigcirc h_2) && (4.42) \\ &= \tilde{G} \bigcirc \tilde{G} \bigcirc h_1 \bigcirc h_2 && (4.43) \\ &= \tilde{G} \bigcirc h_1 \bigcirc \tilde{G} \bigcirc h_2 && (4.44) \\ &= \tilde{\psi}(h_1) \bigcirc \tilde{\psi}(h_2). && (4.45) \end{aligned}$$

Hence $\tilde{\psi}$ is a homomorphism from H onto $H/\tilde{G}$. Furthermore, the kernel of $\tilde{\psi}$ equals $\tilde{G}$.

Thus by letting $\tilde{T} = H/\tilde{G}$ and $\tilde{\psi}(h) = \tilde{G} \bigcirc h$, we have a procedure for finding suitable syndrome groups and homomorphisms.

Note that when implementing a system using such an homomorphism, we will *never* perform computation in the quotient group $\tilde{T}$ by manipulating cosets. Rather, by forming the quotient group, we will recognize that $\tilde{T}$ is isomorphic to a simpler group T which we obtain by renaming the elements of $\tilde{T}$, and this simpler group will be used to perform error detection and correction. Examples of this construction technique are given later in the chapter.

4.3.4 Symmetric Errors

Symmetric errors are also easily placed in the group-theoretic framework. The cancellation shown in (3.21) occurs if and only if every $e \in \mathcal{E}_{\bigcirc}$ has an inverse e^{-1} that is in $\mathcal{E}_{\bigcirc}$ as well. The minimum distance is found to be the smallest integer t such that

$$\left(h \bigcirc \mathcal{E}_{\bigcirc}^{(t)}\right) \cap \mathcal{H}_V \neq \{h\} \qquad \text{for some} \qquad h \in \mathcal{H}_V. \tag{4.46}$$

Set $H_V = \tilde{G}$, and multiply both sides of this expression by h^{-1}. This is an invertible transformation, and we obtain

$$\mathcal{E}_{\bigcirc}^{(t)} \cap \tilde{G} \neq \{0_{\bigcirc}\} \tag{4.47}$$

or, equivalently,

$$e_1 \bigcirc e_2 \bigcirc \cdots \bigcirc e_t \neq 0_{\bigcirc} \in \tilde{G} \qquad \text{for some} \qquad e_1, e_2, \ldots, e_t \in \mathcal{E}_{\bigcirc}. \tag{4.48}$$

Thus, the minimum distance can be thought of as the smallest number of errors that must be combined in order to obtain a nonzero element of $\tilde{G}$. This interpretation often allows the minimum distance of a system to be easily computed.

4.4 Partial Homomorphism

This section briefly describes an alternate method of adding redundancy to computation performed in algebraic groups. The mappings that we will use are related to homomorphisms, but only satisfy (4.15) for certain combinations of operands. We will refer to these mappings as *partial* homomorphisms.

Assume that computation has the same overall structure as before: redundancy is adding by mapping computation to another group $H = [\mathcal{H}; \bigcirc, 0_\bigcirc]$ as shown in Figure 4-1. Assume that $\phi_1 = \phi_2 = \cdots \phi_P = \sigma^{-1}$ as well. Previously, we allowed all possible combinations of input operands, and this forced ϕ to be an algebraic homomorphism, and as a direct consequence, the set of valid results $\mathcal{H}_V$ formed a subgroup $\tilde{G}$.

In a partial homomorphism, we select a redundant group H and then explicitly specify the subset of valid results. Call this subset $\mathcal{H}_V$ in accordance with our former notation, and keep in mind that $\mathcal{H}_V$ must no longer be a subgroup. We then restrict computation to operands such that error-free results lie in $\mathcal{H}_V$. That is, computation is only defined for operands $g_1, g_2, \ldots, g_P$ such that

$$\phi(g_1) \bigcirc \phi(g_2) \bigcirc \cdots \bigcirc \phi(g_P) \in \mathcal{H}_V. \tag{4.49}$$

In order for the correct result to be computed in an error-free system, the mapping ϕ must now satisfy

$$\phi(g_1 \square g_2 \square \cdots \square g_P) = \phi(g_1) \bigcirc \phi(g_2) \bigcirc \cdots \bigcirc \phi(g_P) \tag{4.50}$$
$$\text{for all } g_1, g_2, \ldots, g_P \text{ such that } \phi(g_1) \bigcirc \phi(g_2) \bigcirc \cdots \bigcirc \phi(g_P) \in \mathcal{H}_V.$$

We refer to a mapping of this form as a partial homomorphism since it is structurally similar to (4.15) but only holds for certain operands.

We now develop some tools for analyzing systems that utilize partial homomorphisms. The redundancy condition shown in (4.23) was a direct application of the set-theoretic framework, and it did not assume any special form of $\mathcal{H}_V$; it only assumed that errors influence the result in an additive manner. Thus, as long as an additive error model applies, (4.23) may be used to measure the redundancy present in a partial homomorphism. The remainder of the results from Section 4.3 may not be used since they make explicit use of the fact that $\tilde{G}$ is a subgroup.

To determine the form of the error-correcting mapping α, we must return to (3.19).

Substituting in for $\mathcal{H}_{h,C}$, we find that

$$\alpha(h') = \begin{cases} h & \text{if } h' \in h \bigcirc \mathcal{E}_{\bigcirc}^{(C)} \\ * & \text{else.} \end{cases} \tag{4.51}$$

Symmetric errors have the same interpretation since errors affect the result in an additive manner. The minimum distance of a system is determined by (4.46); (4.47) and (4.48) no longer apply since $\mathcal{H}_V$ is not a subgroup.

Our main motivation for studying partial homomorphisms is that they model computation in systems protected by integer residue number systems, an important fault-tolerant system. Although partial homomorphisms do not fit precisely into our framework, there are, nevertheless, many important similarities: the same basic overall structure, mappings with properties akin to algebraic homomorphisms, and an additive error model. We will use the rudimentary tools developed in this section to study a fault-tolerant convolution algorithm which is based on a polynomial residue number system. This is presented in Chapter 6.

4.5 Code Selection Procedure

For a given homomorphism ϕ, we are able to determine if sufficient redundancy exists to detect D and correct C simultaneous errors in the set $\mathcal{E}_{\bigcirc}$. In reality, one begins with G, and needs to determine ϕ and H such that sufficient redundancy exists in order to protect against the expected set of errors $\mathcal{E}_{\bigcirc}$. This is a difficult problem because of the interaction between ϕ, H, and $\mathcal{E}_{\bigcirc}$. Each group H has an operation $\bigcirc$ associated with it which has its own set of errors $\mathcal{E}_{\bigcirc}$. The errors $\mathcal{E}_{\bigcirc}$ are determined by H, which is in turn determined by ϕ. There is also some flexibility in the manner in which the product $\bigcirc$ is computed. Different computational procedures lead to different sets of errors $\mathcal{E}_{\bigcirc}$, and thus it is necessary to check the fault-tolerant capability of different procedures and architectures for computing $\bigcirc$. Thus choosing a code ϕ is not a straightforward task. We propose the following iterative approach:

1. Select some mapping ϕ and group H which satisfy the homomorphism (4.16).

2. Determine the set of errors $\mathcal{E}_{\bigcirc}$ which would arise during computation. Be sure to check for equivalent computational procedures yielding different errors $\mathcal{E}_{\bigcirc}$.

3. Check if ϕ has sufficient redundancy to protect against the expected number of simultaneous errors. If sufficient redundancy exists continue to step 4. Otherwise return to step 1 and repeat with different ϕ and H.

4. Check if the encoding ϕ, group product $\bigcirc$, and error detection and correction may be done in an efficient manner. If they can, then we have identified a practical arithmetic code and we are done. Otherwise return to step 1 and repeat with different ϕ and H.

Although this is an indirect procedure, it can still be very fruitful. The constraint that ϕ be a homomorphism narrows the search for possible groups H. Also, in many instances, the form of the redundancy is recognized as a standard error-correcting code, and existing techniques may be used to choose ϕ.

Suppose that the desired computation consists of a total of $P-1$ group products, and that we wish to detect D and correct C simultaneous errors. Suppose further that no suitable homomorphism is found which contains sufficient redundancy to protect against this number of errors. It may still be possible to attain the desired level of protection by performing error detection and correction repeatedly throughout computation rather than once at the end. Check the intermediate result of computation after every $P' < P$ group products. Since less computation occurs, a lower number of simultaneous errors should occur, and a homomorphism may exist which can protect against this number of errors.

4.6 Application to Other Algebraic Systems

A number of other algebraic systems exist in which useful computation can be performed. These include rings, fields, and vector spaces, and this section discusses how our framework may be extended to protect computation in these systems as well.

These other algebraic systems differ from groups in several ways, the most notable of which is that they have two operations defined on the set of elements. The additional operation yields a greater amount of flexibility in the types of computation which may be performed. In groups the most general computation consisted of computing $P-1$ products of P operands as shown in (4.1). In these other systems, however, computation consists of some arbitrary sequence of the two operations applied to the input operands. By defining the homomorphism properly, it is possible to protect arbitrary computation of this form. We will outline the structure of each algebraic system and then state the necessary form of the homomorphism.

All of the systems which we will study have an underlying structure of an Abelian group, and this structure well-models the effects of errors. In all cases, error-free results form a subgroup of the redundant output. Since an additive error model is used, we can continue to apply the syndrome-based redundancy measures and error detection and correction procedures that were previously developed. An important point to keep in mind is that since errors influence the result in an additive manner, the syndrome homomorphism must only preserve the structure of the additive operation and is thus still defined by (4.16).

4.6.1 Rings

Definition 8. A ring $G = [\mathcal{G}; \square, \boxtimes, 0_\square]$ is an algebraic system with two operations, $\square$ and $\boxtimes$, defined on the set $\mathcal{G}$. Under $\square$, G is an Abelian group with $0_\square$ serving as the identity. The second operation, $\boxtimes$, is defined by the following properties:

(i) For all $g_1, g_2 \in G$, $g_1 \boxtimes g_2$ is in G. (closure)

(ii) For all $g_1, g_2, g_3 \in G$, $g_1 \boxtimes (g_2 \boxtimes g_3) = (g_1 \boxtimes g_2) \boxtimes g_3$. (associativity)

(iii) For all $g_1, g_2, g_3 \in G$,

$$g_1 \boxtimes (g_2 \square g_3) = (g_1 \boxtimes g_2) \square (g_1 \boxtimes g_3) \tag{4.52}$$

$$(g_2 \square g_3) \boxtimes g_1 = (g_2 \boxtimes g_1) \square (g_3 \boxtimes g_1). \tag{4.53}$$

(distributive laws)

$\square$ and $\boxtimes$ are referred to, respectively, as the additive and multiplicative operations of G.

A homomorphism ϕ from a ring $G = [\mathcal{G}; \square, \boxtimes, 0_\square]$ to another ring $H = [\mathcal{H}; \bigcirc, \otimes, 0_\bigcirc]$ is a mapping which preserves both arithmetic operations,

$$\begin{aligned} \phi(g_1 \square g_2) &= \phi(g_1) \bigcirc \phi(g_2) \\ \phi(g_1 \boxtimes g_2) &= \phi(g_1) \otimes \phi(g_2) \end{aligned} \qquad \text{for all} \qquad g_1, g_2 \in G. \tag{4.54}$$

A ring homomorphism maps computation to $\tilde{G}$, a subring of H which is isomorphic to G.

4.6.2 Fields

Definition 9. A field $F = [\mathcal{F}; \square, \boxtimes, 0_\square]$ is a ring in which the nonzero elements of $\mathcal{F}$ form an Abelian group under $\boxtimes$.

Structurally, fields and rings are very similar, and a field homomorphism is defined by (4.54) as well. One difference to keep in mind is that the redundant system H will not be a field, but rather a ring. This is because a field cannot contain any subfields, while it is possible for a ring to contain subfields. The redundant computation occurs in a subfield $\tilde{G}$ of H that is isomorphic to G.

4.6.3 Vector Spaces

Definition 10. A *vector space over a field* $F = [\mathcal{F}; \diamond, \diamondplus, 0_\diamond]$ consists of an Abelian group $G = [\mathcal{G}; \square, 0_\square]$ together with an operation called scalar multiplication and denoted by $\cdot$ such that for all $\alpha, \beta \in F$ and $v, w \in G$,

(i) $\alpha \cdot v \in G$

(ii) $\alpha \cdot (\beta \cdot v) = (\alpha \diamondplus \beta) \cdot v$

(iii) $(\alpha \diamond \beta) \cdot v = (\alpha \cdot v) \square (\beta \cdot v)$

(iv) $\alpha \cdot (v \square w) = (\alpha \cdot v) \square (\alpha \cdot w)$

(v) $1 \cdot v = v$

The elements of G are called *vectors* and the elements of F are called *scalars.*

A vector space homomorphism is a mapping ϕ from a vector space $G = [\mathcal{G}; \Box, 0_\Box]$ over F to another vector space $H = [\mathcal{H}; \bigcirc, 0_\bigcirc]$ over F such that

$$\begin{aligned} \phi(v \Box w) &= \phi(v) \bigcirc \phi(w) \\ \phi(\alpha \cdot v) &= \alpha : \phi(v) \end{aligned} \qquad \text{for all} \qquad v, w \in G \text{ and } \alpha \in F \tag{4.55}$$

where : denotes the operation of scalar multiplication between elements of H and F. The key difference between a vector space homomorphism and a group homomorphism is that redundancy is added to vectors, but not to scalars. A vector space homomorphism maps computation to $\tilde{G}$, a subspace of H which is isomorphic to G.

Carefully distinguishing the operations in a vector space is a tedious process that does not clarify the presentation significantly. Instead, we will use $+$ in place of both $\Box$ and $\Diamond$, and use juxtaposition in place of $\diamond$, $\cdot$, and $:$. Also, 0 will denote the identity in G, H, and F. It will always be clear from context which operation we are referring to. With this new notation, a vector space homomorphism is a mapping satisfying

$$\begin{aligned} \phi(v + w) &= \phi(v) + \phi(w) \\ \phi(\alpha v) &= \alpha \phi(v) \end{aligned} \tag{4.56}$$

for all $v, w \in G$ and $\alpha \in F$.

We will always utilize the familiar vector spaces containing elements of the form

$$G = \left\{ [\alpha_1, \alpha_2, \ldots, \alpha_N] \mid \alpha_i \in F \right\} \tag{4.57}$$

for some integer N. Operations in G are defined as

$$[\alpha_1, \alpha_2, \ldots, \alpha_N] + [\beta_1, \beta_2, \ldots, \beta_N] = [\alpha_1 + \beta_1, \alpha_2 + \beta_2, \ldots, \alpha_N + \beta_N] \tag{4.58}$$

$$\gamma [\alpha_1, \alpha_2, \ldots, \alpha_N] = [\gamma\alpha_1, \gamma\alpha_2, \ldots, \gamma\alpha_N] \tag{4.59}$$

for all $[\alpha_1, \alpha_2, \ldots, \alpha_N]$, $[\beta_1, \beta_2, \ldots, \beta_N] \in G$, and $\gamma \in F$. We denote this vector space by $F^{(N)}$, and when dealing with elements of $F^{(N)}$, will use underbars (e.g., $\underline{v}$) to stress that the elements are vectors.

4.7 Examples

In this section we give several examples to demonstrate the application of our technique. The examples were chosen to illustrate how the framework can be used to analyze fault-tolerant systems, and are not constructive procedures. For each example, we describe the original operation in G and give a possible homomorphism ϕ. In some cases, we determine the error detection and correction procedures and show which errors are protected against. A thorough treatment of each coding scheme is beyond the scope of this thesis since the choice of code is closely tied to the set of expected errors which is, in turn, determined by the specific hardware architecture employed. In many cases, the resulting code is shown to be equivalent to one studied by other authors. In these cases, we refer the reader to references in which a more thorough presentation is made.

Most of the examples that we give are for protecting a single product of two operands. This was done in order to simplify the presentation and more clearly reveal the form of the redundancy introduced by the homomorphism ϕ. In all cases, our techniques are able to protect a series of operations.

4.7.1 Triple Modular Redundancy

Let G be any Abelian group and let H be the direct product of G with itself three times, $H = G \times G \times G$. Elements of H are of the form $[g_1, g_2, g_3]$ where $g_1, g_2, g_3 \in G$, and the product in H is componentwise multiplication

$$[g_1, g_2, g_3] \bigcirc [g_4, g_5, g_6] = [g_1 \Box g_4, g_2 \Box g_5, g_3 \Box g_6]. \tag{4.60}$$

Define the mapping ϕ from G to H as follows,

$$\phi(g) = [g, g, g]. \tag{4.61}$$

Plugging (4.61) into (4.16), we see that this is a valid homomorphism,

$$\phi(g_1 \Box g_2) = [g_1 \Box g_2, g_1 \Box g_2, g_1 \Box g_2] = \phi(g_1) \bigcirc \phi(g_2). \tag{4.62}$$

Also, the subgroup of valid results is given by $\tilde{G} = \left\{ [g,g,g] \mid g \in G \right\}$.

Suppose that (4.60) is computed by three independent processors, with one product computed per processor. Assume that when a processor fails, an error $e \in G$ is added to its output. The set of single errors is of the form

$$\mathcal{E}_{\bigcirc} = \left\{ [e, 0_\Box, 0_\Box], [0_\Box, e, 0_\Box], [0_\Box, 0_\Box, e] \mid e \in G \right\}. \tag{4.63}$$

Since every $e \in G$ has an inverse $e^{-1} \in G$, the errors are symmetric, and we may analyze the system via its minimum distance. It is seen by the following progression of sets that the minimum distance of this code is 3,

$$\begin{aligned}
\mathcal{E}_{\bigcirc}^{(1)} &= \left\{ [e, 0_\Box, 0_\Box], [0_\Box, e, 0_\Box], [0_\Box, 0_\Box, e] \mid e \in G \right\} && (4.64)\\
\mathcal{E}_{\bigcirc}^{(2)} &= \left\{ [e_1, e_2, 0_\Box], [e_1, 0_\Box, e_2], [0_\Box, e_1, e_2] \mid e_1, e_2 \in G \right\} && (4.65)\\
\mathcal{E}_{\bigcirc}^{(3)} &= H. && (4.66)
\end{aligned}$$

Thus we may detect D and correct C errors where $D + C + 1 \leq 3$ and $D \geq C$.

We derive the form of the error detection and correction procedures used in this system via a quotient group construction. We must determine the structure of the cosets as well as the group product within the quotient group. An arbitrary coset is of the form

$$\begin{aligned}
\tilde{G} \bigcirc [a,b,c] &= \left\{ [g \Box a, g \Box b, g \Box c] \mid g \in G \right\} && (4.67)\\
&= \left\{ \left[g', g' \Box a^{-1} \Box b, g' \Box a^{-1} \Box c \right] \mid g' \in G \right\}. && (4.68)
\end{aligned}$$

From this we see that there are only two degrees of freedom, and the quotient group consists of the set of elements

$$H/\tilde{G} = \left\{ \tilde{t}_{[a,b]} \mid a, b \in G \right\} \tag{4.69}$$

where

$$\tilde{t}_{[a,b]} = \left\{ [g, g\Box a, g\Box b] \mid g \in G \right\}. \tag{4.70}$$

The homomorphism $\tilde{\psi}$ from H onto $H/\tilde{G}$ is derived by combining (4.68) and (4.70), and equals

$$\tilde{\psi}([a,b,c]) = \tilde{t}_{[a^{-1}\Box b, a^{-1}\Box c]}. \tag{4.71}$$

Multiplying cosets, we find that the group product is given by

$$\begin{aligned}
\tilde{t}_{[a,b]} \bigcirc \tilde{t}_{[c,d]} &= \left\{ [g, g\Box a, g\Box b] \bigcirc [g', g'\Box c, g'\Box d] \mid g, g' \in G \right\} && (4.72)\\
&= \left\{ [g\Box g', g\Box g'\Box a\Box c, g\Box g'\Box b\Box d] \mid g, g' \in G \right\} && (4.73)\\
&= \left\{ [g, g\Box a\Box c, g\Box b\Box d] \mid g \in G \right\} && (4.74)\\
&= \tilde{t}_{[a\Box c, b\Box d]}. && (4.75)
\end{aligned}$$

Examining this product, we see that $H/\tilde{G}$ is isomorphic to the simpler group $T = G \times G$. The coset $\tilde{t}_{[a,b]}$ is renamed $[a,b]$, and the homomorphism from H onto T is given by

$$\psi([a,b,c]) = \left[a^{-1}\Box b, a^{-1}\Box c\right]. \tag{4.76}$$

We declare that no errors have occurred if $\psi([a,b,c]) = [0_\Box, 0_\Box]$, i.e., if $a = b = c$. This is an equality checker. The identity of the failed processor and the exact value of the error is determined from the syndrome as well. We find that

$$\begin{aligned}
\psi([e, 0_\Box, 0_\Box]) &= [e^{-1}, e^{-1}] && \Longrightarrow \text{ error } e \text{ in processor \# 1}\\
\psi([0_\Box, e, 0_\Box]) &= [e, 0_\Box] && \Longrightarrow \text{ error } e \text{ in processor \# 2}\\
\psi([0_\Box, 0_\Box, e]) &= [0_\Box, e] && \Longrightarrow \text{ error } e \text{ in processor \# 3.}
\end{aligned}$$

Once the error-free encoded result h is found, we determine the desired result r by mapping h back to G using ϕ^{-1}. This corresponds to choosing any element of h,

$$\phi^{-1}\left([g,g,g]\right) = g. \tag{4.77}$$

This example is the standard form of triple modular redundancy. An equality tester detects errors and a majority circuit determines the most likely result. The framework is able to verify the error detecting and correcting capabilities of triple modular redundancy, as well as determine the required error detection and correction procedures.

Although presented in terms of groups, it can be readily seen that these results apply directly to all other algebraic systems. In fact, triple modular redundancy may be utilized to protect *any* system since, when the system is triplicated, the set of valid outputs always form a 1-dimensional subgroup of the larger 3-dimensional redundant output space. Redundancy has the exact same form, and the same error detection and correction procedures may be used.

4.7.2 aN Codes

Let $G = \mathbb{Z}_5^+$, $H = \mathbb{Z}_{55}^+$, and $\phi(g) = 11g$. ϕ is a one-to-one mapping from G to the subgroup $\tilde{G} = \{0, 11, 22, 33, 44\}$. It is a homomorphism since

$$\phi(g_1 \Box g_2) = 11 \langle g_1 + g_2 \rangle_5 = \langle 11g_1 + 11g_2 \rangle_{55} = \phi(g_1) \bigcirc \phi(g_2). \tag{4.78}$$

Assume that $\bigcirc$ is computed by a 6 bit binary adder and that we expect single bit errors of the form $\mathcal{E}_{\bigcirc} = \{0, \pm1, \pm2, \pm4, \pm8, \pm16, \pm32\} = \{0, 1, 2, 4, 8, 16, 32, 48, 56, 60, 62, 63\}$. These are symmetric errors and the minimum distance is found from the sets

$$\begin{aligned}
\mathcal{E}_{\bigcirc}^{(1)} &= \{0, 1, 2, 4, 8, 16, 32, 48, 56, 60, 62, 63\} \\
\mathcal{E}_{\bigcirc}^{(2)} &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 16, 17, 18, 20, 24, 28, \\
&\qquad 30, 31, 32, 33, 34, 36, 40, 44, 46, 47, 48, 49, 50, 52, 54, 55, \\
&\qquad 56, 57, 58, 59, 60, 61, 62, 63\}.
\end{aligned}$$

Since $\mathcal{E}_{\bigcirc}^{(1)} \cap \tilde{G} = \{0\}$ and $\mathcal{E}_{\bigcirc}^{(2)} \cap \tilde{G} \neq \{0\}$, the minimum distance equals 2. Thus we can, at most, detect single errors.

The form of the syndrome mapping ψ can be found using a quotient group homomorphism. It is well-known that, for this example, $H/\tilde{G}$ is isomorphic to $T = \mathbb{Z}_{11}^+$. The mapping from H onto T is given by $\psi(h') = \langle h' \rangle_{11}$. If $\langle h' \rangle_{11} = 0$, we declare that the result is error-free. Since the group $\mathbb{Z}_{11}^{+}$ contains a total of 10 nonzero elements, there is insufficient redundancy to correct the 12 nonzero errors in $\mathcal{E}_{O}^{(1)}$. (This is an alternative method of testing if sufficient redundancy exists to correct a given set of errors.)

This type of code is referred to as an integer aN code. The general form of the encoding is $\phi(g) = gN$ where N is an integer called the generator or check base. These codes have been studied extensively, and for certain values of N, can detect and correct errors [6].

4.7.3 Matrix Rings

Let $\mathcal{G}$ be the set of all $N \times N$ matrices with coefficients that are elements of a field F. This set forms a ring, G, under standard matrix addition and multiplication. We protect computation in G by embedding it in the larger ring H containing matrices of dimension $(N + C) \times (N + C)$. The coefficients of H are elements of F as well. The encoding from G to H is given by

$$\phi(\boldsymbol{A}) = \boldsymbol{\Phi}_1 \boldsymbol{A} \boldsymbol{\Phi}_2 \tag{4.79}$$

where $\boldsymbol{\Phi}_1$ and $\boldsymbol{\Phi}_2$ have dimensions $(N+C)\times N$ and $N\times(N+C)$, respectively, and $\boldsymbol{\Phi}_2\boldsymbol{\Phi}_1 = \boldsymbol{I}$, the $N \times N$ identity matrix. ϕ is a ring homomorphism since

$$\begin{aligned}
\phi(\boldsymbol{A}+\boldsymbol{B}) &= \boldsymbol{\Phi}_1(\boldsymbol{A}+\boldsymbol{B})\boldsymbol{\Phi}_2 &= \boldsymbol{\Phi}_1\boldsymbol{A}\boldsymbol{\Phi}_2 + \boldsymbol{\Phi}_1\boldsymbol{B}\boldsymbol{\Phi}_2 &= \phi(\boldsymbol{A}) + \phi(\boldsymbol{B}) \\
\phi(\boldsymbol{A}\boldsymbol{B}) &= \boldsymbol{\Phi}_1\boldsymbol{A}\boldsymbol{B}\boldsymbol{\Phi}_2 &= \boldsymbol{\Phi}_1\boldsymbol{A}\boldsymbol{\Phi}_2\boldsymbol{\Phi}_1\boldsymbol{B}\boldsymbol{\Phi}_2 &= \phi(\boldsymbol{A})\phi(\boldsymbol{B})
\end{aligned}$$

for all $\boldsymbol{A}, \boldsymbol{B} \in G$. The homomorphism is satisfied for arbitrary matrices $\boldsymbol{\Phi}_1$ and $\boldsymbol{\Phi}_2$ as long as $\boldsymbol{\Phi}_2\boldsymbol{\Phi}_1 = \boldsymbol{I}$.

In this example, we will further assume that the encoding matrices have the following special form

$$\boldsymbol{\Phi}_1 = \left[\begin{array}{c} \boldsymbol{I} \\ \hline \tilde{\boldsymbol{\Phi}}_1 \end{array}\right] \qquad \boldsymbol{\Phi}_2 = \left[\begin{array}{c|c} \boldsymbol{I} & \tilde{\boldsymbol{\Phi}}_2 \end{array}\right] \tag{4.80}$$

where $\tilde{\boldsymbol{\Phi}}_1$ and $\tilde{\boldsymbol{\Phi}}_2$ have dimensions $C \times N$ and $N \times C$ respectively. This leads to a systematic-nonseparate version of the encoding which is more easily analyzed. In order that $\boldsymbol{\Phi}_2\boldsymbol{\Phi}_1 = \boldsymbol{I}$, the rows of $\tilde{\boldsymbol{\Phi}}_1$ must be orthogonal to the columns of $\tilde{\boldsymbol{\Phi}}_2$.

Assume that a single fault occurring during computation corrupts a single element of the $(N+C) \times (N+C)$-dimensional result. We model this as an additive error

$$\boldsymbol{H}' = \boldsymbol{H} + \boldsymbol{E} \tag{4.81}$$

where $\boldsymbol{H}'$ is the faulty result, $\boldsymbol{H}$ is the error-free result, and $\boldsymbol{E}$ is a matrix containing only a single nonzero element e of F. Assume that e can be an arbitrary element of F. Then the induced errors will be symmetric and we can analyze this system by its minimum distance.

Redundant matrix arithmetic of this form was studied in detail by Jou [25], and was mentioned briefly in the introductory material presented in Section 2.4.2. He showed that by proper choice of the encoding matrices $\tilde{\boldsymbol{\Phi}}_1$ and $\tilde{\boldsymbol{\Phi}}_2$, the minimum distance of the code may be as high as $(C+1)^2$. He also presented algorithms for detecting and correcting errors.

The subring of valid result matrices is given by

$$\tilde{G} = \left\{ \left[\begin{array}{c|c} \boldsymbol{A} & \boldsymbol{A}\tilde{\boldsymbol{\Phi}}_2 \\ \hline \tilde{\boldsymbol{\Phi}}_1\boldsymbol{A} & \tilde{\boldsymbol{\Phi}}_1\boldsymbol{A}\tilde{\boldsymbol{\Phi}}_2 \end{array} \right] \middle| \boldsymbol{A} \in G \right\}. \tag{4.82}$$

In order to detect errors, we compute a syndrome based on the result $\boldsymbol{H}'$. Since the redundant space has a dimension of $(N+C)^2$, and since the subspace of valid results has dimension N^2, the syndrome group must have dimension $(N+C)^2 - N^2 = 2C + C^2$. Any homomorphism from H onto a $2C+C^2$ dimensional space that has kernel $\tilde{G}$ will suffice as the syndrome mapping. One possible mapping is as follows. First, partition $\boldsymbol{H}'$ into the submatrices

$$\boldsymbol{H}' = \left[\begin{array}{c|c} \boldsymbol{R}' & \boldsymbol{P}_1 \\ \hline \boldsymbol{P}_2 & \boldsymbol{P}_3 \end{array} \right] \tag{4.83}$$

where $\boldsymbol{R}'$ is $N \times N$, $\boldsymbol{P}_1$ is $N \times C$, $\boldsymbol{P}_2$ is $C \times N$, and $\boldsymbol{P}_3$ is $C \times C$. Then, compute the syndrome consisting of the three matrices

$$\psi\left(\left[\begin{array}{c|c} \boldsymbol{R}' & \boldsymbol{P}_1 \\ \hline \boldsymbol{P}_2 & \boldsymbol{P}_3 \end{array} \right] \right) = [\boldsymbol{S}_1, \boldsymbol{S}_2, \boldsymbol{S}_3] \tag{4.84}$$

where

$$\begin{aligned} \boldsymbol{S}_1 &= \boldsymbol{R}'\tilde{\boldsymbol{\Phi}}_2 - \boldsymbol{P}_1 && (4.85) \\ \boldsymbol{S}_2 &= \tilde{\boldsymbol{\Phi}}_1\boldsymbol{R}' - \boldsymbol{P}_2 && (4.86) \\ \boldsymbol{S}_3 &= \tilde{\boldsymbol{\Phi}}_1\boldsymbol{R}'\tilde{\boldsymbol{\Phi}}_2 - \boldsymbol{P}_3. && (4.87) \end{aligned}$$

The result is in $\tilde{G}$ if and only if all three matrices in the syndrome equal zero. The general form of the error correction procedure is described in [25] and is quite complicated. The procedure basically first localizes the error by examining the nonzero elements of $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$. Then the exact value of the error $\boldsymbol{E}$ is determined using all three syndromes.

Many variations of this basic encoding scheme are possible. Our analysis assumed a very simple error model, but it can be tailored to the specific hardware architecture used to perform the computation. Also, operations with nonsquare matrices can be protected by embedding computation in square matrices.

4.7.4 Finite Fields

Let G be the field consisting of the integers $\{0, 1, \ldots, P-1\}$ where P is a prime number along with modulo P addition and multiplication. All finite fields containing P elements are isomorphic to G.

We embed computation in the ring H consisting of the set of integers $\{0, 1, \ldots, Q-1\}$ under modulo Q addition and multiplication. Assume that $Q = PN$ for some integer N that is relatively prime to P. We will denote the additive operations in G and H by $+$, and the multiplicative operations by juxtaposition. Define π as the multiplicative inverse of N in G. π is the element in G satisfying

$$\langle \pi N \rangle_P = 1. \tag{4.88}$$

The mapping from G to H which adds redundancy is given by

$$\phi(g) = \langle \pi N g \rangle_Q \tag{4.89}$$

where the product πN can be precomputed. ϕ satisfies the additive property of a ring homomorphism since

$$\begin{aligned}\phi(g_1+g_2) &= \langle \pi N \langle g_1+g_2\rangle_P\rangle_Q && (4.90)\\ &= \langle \pi N g_1 + \pi N g_2\rangle_Q && (4.91)\\ &= \phi(g_1)+\phi(g_2). && (4.92)\end{aligned}$$

Showing that ϕ satisfies the multiplicative property of a ring homomorphism is slightly more complicated,

$$\begin{aligned}\phi(g_1)\phi(g_2) &= \left\langle \langle \pi N g_1\rangle_Q \langle \pi N g_2\rangle_Q \right\rangle_Q && (4.93)\\ &= \langle \pi N \pi N g_1 g_2\rangle_Q && (4.94)\\ &= N\langle \pi(\pi N) g_1 g_2\rangle_P && (4.95)\end{aligned}$$

where the last line follows since N divides Q. We can reduce this further by noting that $\langle \pi N\rangle_P = 1$. Then,

$$\phi(g_1)\phi(g_2) = N\langle \pi g_1 g_2\rangle_P = \langle \pi N g_1 g_2\rangle_Q = \phi(g_1 g_2). \tag{4.96}$$

Under this homomorphism, the subgroup of error-free results consists of the multiples

$$\tilde{G} = \{0, N, 2N, \ldots, (P-1)N\}. \tag{4.97}$$

The redundancy is equivalent to that in aN codes which were presented in Section 4.7.2. The same error detection and correction procedures may be used, and the performance of the code is determined by N. Guidelines for choosing N are given in [6].

4.7.5 Linear Transformations

Suppose that we wish to protect the linear transformation $\underline{r} = L(\underline{g})$ from $\mathbb{R}^{(M)}$ to $\mathbb{R}^{(N)}$. This computation can be modeled as the vector-matrix multiplication

$$\underline{r} = \underline{g}\boldsymbol{L} \tag{4.98}$$

where $\boldsymbol{L}$ is an $M \times N$ matrix describing the transformation. We can protect this operation by splitting the transformation into smaller operations, each of which may be placed in our framework. Rewrite (4.98) as

$$\begin{aligned} \underline{r} &= [g_1 \cdots g_M] \begin{bmatrix} \text{---}\underline{l}_1\text{---} \\ \vdots \\ \text{---}\underline{l}_M\text{---} \end{bmatrix} && (4.99) \\ &= \sum_{i=1}^{M} g_i \underline{l}_i && (4.100) \end{aligned}$$

where $\underline{l}_i$ is the $i^{\underline{th}}$ row of $\boldsymbol{L}$, and g_i is a scalar denoting the $i^{\underline{th}}$ element of $\underline{g}$. Computation consists of a weighted linear combination of N-dimensional vectors in $\mathbb{R}^{(N)}$, and this may be placed in our framework.

We add redundancy by embedding computation in $\mathbb{R}^{(N+C)}$. Redundancy is added to the vectors $\underline{l}_i$ but not to the scalars g_i. The homomorphism that encodes the vectors is given by

$$\underline{\tilde{l}}_i = \phi(\underline{l}_i) = \underline{l}_i \boldsymbol{\Phi} \tag{4.101}$$

where $\boldsymbol{\Phi}$ is an $N \times (N + C)$ matrix with full row rank (this ensures that ϕ is one-to-one). Next, we perform a sequence of operations on the encoded vectors similar to the desired computation shown in (4.100),

$$\underline{h} = \sum_{i=1}^{M} g_i \underline{\tilde{l}}_i. \tag{4.102}$$

As usual, an additive error model is assumed. If λ errors occur, then the result is given by

$$\underline{h}' = \underline{h} + \underline{e} \tag{4.103}$$

where $\underline{e} = \sum_{i=1}^{\lambda} \underline{e}_i$, and $\underline{e}_i$ is the net effect of the $i^{\underline{th}}$ error on the result. We assume that $\underline{e}_i$ contains only a single nonzero element and is of the form $[0 \cdots 0 \; e_i \; 0 \cdots 0]$ where $e_i \in \mathbb{R}$. This is a symmetric error, and the distance of the code depends on $\boldsymbol{\Phi}$. Codes similar in form to these were analyzed by Wolf [46] and Jou [25]. By proper choice of $\boldsymbol{\Phi}$, they may have a minimum distance as large as $C + 1$.

To detect and correct errors, we must compute a syndrome based on the result $\underline{h}'$. Recall that the syndrome mapping ψ is a homomorphism from H onto another vector space in which the kernel $K_\psi = \tilde{G}$. In this case, $\tilde{G}$ is an N-dimensional subspace of $\mathbb{R}^{(N+C)}$ equal to the row space of $\boldsymbol{\Phi}$. It can be shown that the left nullspace of the matrix $\boldsymbol{\Psi} = \boldsymbol{I} - \boldsymbol{\Phi}^T(\boldsymbol{\Phi}\boldsymbol{\Phi}^T)^{-1}\boldsymbol{\Phi}$ equals $\tilde{G}$. Thus $\boldsymbol{\Psi}$ is a homomorphism from $\mathbb{R}^{(N+C)}$ onto $\mathbb{R}^{(C)}$, and the syndrome is given by $\underline{h}'\boldsymbol{\Psi}$.

The error correction procedure is quite elaborate and is outside the scope of this presentation. We refer the reader to [46] and [25] for a more complete presentation.

4.8 Summary

This chapter applied the general set-theoretic results of Chapter 3 to the special case of computation in an algebraic group. We showed that in order for the structure of the redundancy to be preserved throughout computation, redundancy must be added via an algebraic homomorphism. Conditions for error detectability and correctability were rephrased in terms of group operations, and we showed that error detection and correction may be performed in an efficient manner using a syndrome. We then extended these results to other algebraic systems that have the underlying structure of an Abelian group. Our results for groups carried over completely to these other algebraic systems. We concluded with a wide variety of examples of the application of our techniques.

The results of this section were specifically geared towards computation performed in algebraic groups, rings, fields, and vector spaces. We feel, however, that many of our results are more widely applicable, especially those concerning error detection and correction. Section 4.3 was based entirely on these three assumptions:

(i) the set of outputs may be modeled as an Abelian group H.

(ii) valid results lie in a subgroup $\tilde{G}$ of H.

(iii) errors influence the result in an additive manner as shown in (4.21).

Any redundant system satisfying these properties, independent of exactly what computation occurs, may utilize the techniques which we have developed.

The main drawback of this chapter is that we do not have, as of yet, a general method of determining possible homomorphisms ϕ. This makes the results and techniques developed difficult to apply. We solve this problem in Chapter 5 in which we give a step-by-step procedure for finding homomorphisms for the important class of systematic-separate codes.

Chapter 5

Systematic-Separate Codes

The results of the previous chapter are only useful if a procedure for determining possible homomorphisms exists. In this chapter, we consider the class of systematic-separate codes which protect computation using a parallel, independent parity channel. For these codes, we provide a constructive procedure for determining all possible homomorphisms. These homomorphisms may or may not be useful for protecting against the expected set of errors which would arise during computation, and the iterative code selection procedure is still needed to determine suitable codes.

We first introduce systematic-separate codes in Section 5.1 and show how they fit into our framework. We then present a procedure for finding homomorphisms in Section 5.2, and prove that it identifies all possible systematic-separate codes. The technique relies upon a quotient group homomorphism, and reduces the problem of finding homomorphisms to that of finding subgroups. We derive the form of the error detection and correction algorithms in Section 5.3, and reinterpret redundancy conditions. Section 5.4 considers systems protected by multiple parity channels, and 5.5 extends the results for groups to rings, fields, and vector spaces. Finally, we present several examples of systems protected by these codes in Section 5.6, and conclude in 5.7.

5.1 Description of Codes

A systematic arithmetic code is defined as one in which a codeword is composed of two distinct parts: the original unencoded operand g, and a *parity* or *checksum* symbol derived from g. For our purposes, we can model a systematic code as mapping G to a Cartesian product of groups, $H = G \times T$, where $T = [\mathcal{T}; \Diamond, 0_\Diamond]$ denotes the group of parity symbols. An element $g \in G$ is mapped to its corresponding element in H as follows:

$$\tilde{g} = \phi(g) = [g, t] = [g, \theta(g)] \tag{5.1}$$

where θ is the parity mapping from G to T. A nonsystematic code is defined as one which is not systematic. Systematic codes are desirable since the result decoder σ is unnecessary.

Systematic codes can be further classified as being either separate or nonseparate depending upon how computation is performed on the codewords. In a separate code, the group operation $\bigcirc$ in H corresponds to componentwise multiplication of the operands and parity,

$$\begin{aligned} \tilde{g}_1 \bigcirc \tilde{g}_2 \bigcirc \cdots \bigcirc \tilde{g}_P &= [g_1, t_1] \bigcirc [g_2, t_2] \bigcirc \cdots \bigcirc [g_P, t_P] & (5.2) \\ &= [g_1 \Box g_2 \Box \cdots \Box g_P, t_1 \Diamond t_2 \Diamond \cdots \Diamond t_P]. & (5.3) \end{aligned}$$

In a nonseparate code, interaction between operands and parity occurs.

The general form of an operation protected by a systematic-separate code is shown in Figure 5-1. It consists of two parallel, independent channels. In the upper channel, the original product $g_1 \Box g_2 \Box \cdots \Box g_P$ which we sought to protect is computed. We will refer to this as the original or main computation. In the lower channel, we compute the parity symbols, and then their product $t_1 \Diamond t_2 \Diamond \cdots \Diamond t_P$. Then the results of both channels are used to detect and correct errors. The details of the error detection and correction procedures are developed in Section 5.3.

Since ϕ is a homomorphism, this places additional constraints on θ. Starting with (4.16)

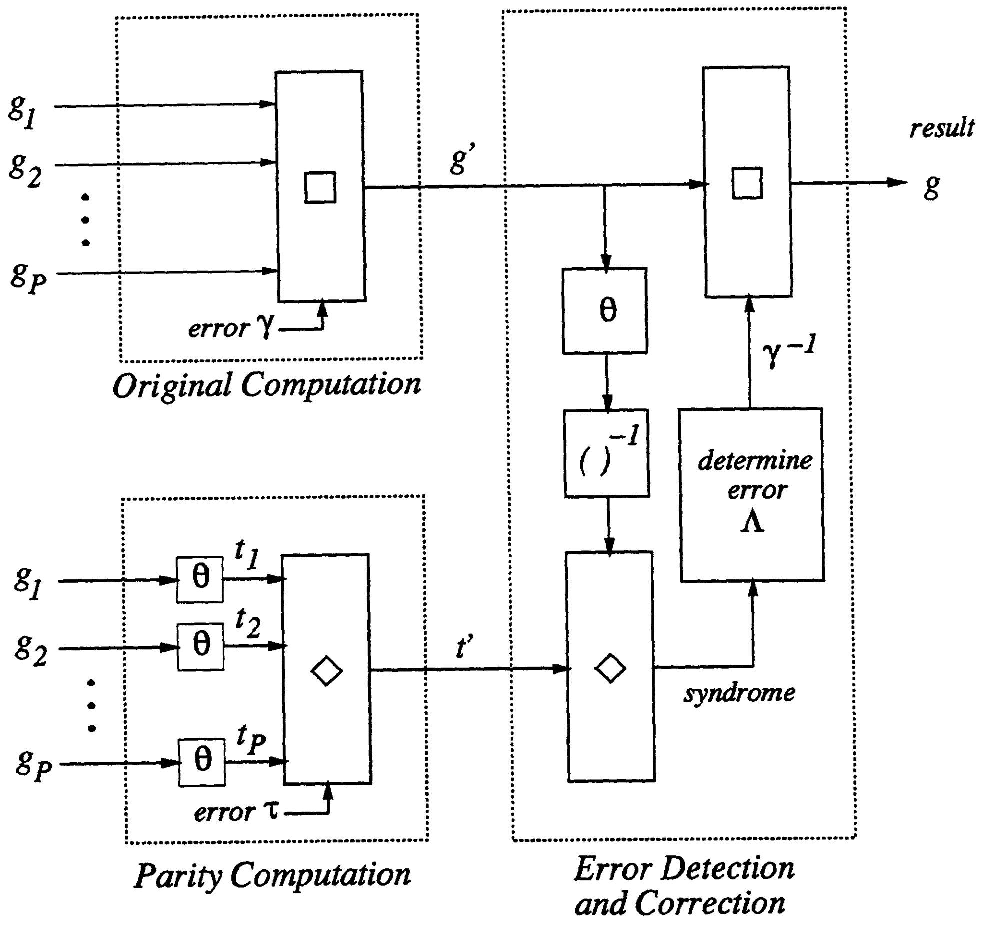


Figure 5-1: General model of a group operation protected by a systematic-separate coding scheme.

and substituting in (5.1), we find that

$$\begin{aligned} [g_1 \Box g_2, \theta(g_1 \Box g_2)] &= [g_1, \theta(g_1)] \bigcirc [g_2, \theta(g_2)] && (5.4) \\ &= [g_1 \Box g_2, \theta(g_1) \Diamond \theta(g_2)]. && (5.5) \end{aligned}$$

Equating terms in T, we find that θ must satisfy

$$\theta(g_1 \Box g_2) = \theta(g_1) \Diamond \theta(g_2) \qquad \text{for all} \qquad g_1, g_2 \in G. \tag{5.6}$$

Thus θ must be a homomorphism from G to T.

The problem of finding systematic-separate codes becomes that of finding groups T and homomorphisms θ. This is the same problem which we faced in Chapter 4 with H and ϕ. In order to solve this problem, we make an important simplifying assumption: we require that θ map G *onto* T. We make this assumption for three important reasons. First, requiring θ to be onto ensures that T will have the same number, or fewer elements as G. Hence the complexity of the parity calculation $\Diamond$ will most likely be the same, or less than, the complexity of the original product $\Box$. Second, efficient use of parity symbols is made since there are no unused elements. Third, the structure of G is heavily reflected in T, and we can use a well-known isomorphism involving quotient groups of G to identity possible groups T and homomorphisms θ.

5.2 Determination of Possible Homomorphisms

We now begin the derivation of the main result of this chapter. Our goal is to provide a constructive procedure for determining all possible groups $T = [\mathcal{T}; \Diamond, 0_\Diamond]$ and homomorphisms θ which yield systematic-separate codes. We rely upon a quotient group isomorphism, and find groups $\tilde{T} = \left[\tilde{\mathcal{T}}; \tilde{\Diamond}, \tilde{0}_\Diamond\right]$ which are isomorphic to all possible parity groups T. As in Chapter 4, we will never perform computation in a quotient group by manipulating cosets. Rather, by forming the quotient group $\tilde{T}$, we will recognize that $\tilde{T}$ is isomorphic to a simpler group T in which computation can be more conveniently performed.

We showed in Section 4.3.3 that the collection of cosets $H/\tilde{G}$ forms a group under the product of subsets. This group was referred to as the quotient group of H by $\tilde{G}$, and its construction relied on the fact that $\tilde{G}$ was a subgroup of H. A quotient group G/N may be formed in a similar manner using any subgroup N of G, and a homomorphism from G onto G/N is given by $\tilde{\theta}(g) = N \Box g$. Letting $\tilde{T} = G/N$ and $\tilde{\theta}(g) = N \Box g$, we have a procedure for finding suitable parity groups and mappings. By selecting different subgroups N, we obtain different groups $\tilde{T}$ and mappings $\tilde{\theta}$.

We now show that this procedure is guaranteed to find, up to isomorphisms, all possible groups $\tilde{T}$ and mappings $\tilde{\theta}$. Assume that we have some homomorphism θ from G onto T. We prove that T is isomorphic to G/N for some subgroup N of G.

Theorem 4. Let θ be a homomorphism from G onto T. The kernel K_θ is a subgroup of G.

Proof. See Lemma 2.7.3 in [45]. $\Box$

Theorem 5. Let θ be a homomorphism from G onto T with kernel K_θ. The quotient group G/K_θ is isomorphic to T.

Proof. See Theorem 2.7.1 in [45]. $\Box$

Thus, any group of parity symbols T is isomorphic to G/N for some subgroup N. A natural question arises: by finding all subgroups N of G, is our method guaranteed to find all possible systematic-separate coding schemes? This is answered in the following theorem.

Theorem 6. Let θ be a homomorphism from G onto T, and let θ' be another homomorphism from G onto T'. If $K_\theta = K_{\theta'}$ then $T \approx T'$.

Proof. We know by Theorem 5 that $T \approx G/K_\theta$ and $T' \approx G/K_{\theta'} = G/K_\theta$. Using transitivity we find that $T \approx T'$. $\Box$

Thus, by finding all possible subgroups N of G, we are able to find, up to isomorphisms, all possible parity groups. We do not distinguish between isomorphic copies of the same parity group because they are structurally identical and have equivalent error detecting and correcting properties.

The reason that we can always define T in this manner is that structurally G and G/N are very similar. Given an element $\tilde{t} \in G/N$, we know its corresponding element in $g \in G$ "up to N." That is, we know that g is in the coset $N \Box g$. A certain amount of information is blotted out by this mapping, but the remaining information is often sufficient for detecting and correcting errors [45].

The problem of finding groups T and homomorphisms θ has been reduced to one of finding subgroups. For many groups that compute useful arithmetic operations, finding subgroups is a trivial task. In these instances we are able to determine all applicable systematic-separate coding schemes.

5.3 Error Detection and Correction

In this section we derive the form of the error detection and correction algorithms which would be used in a systematic-separate coding scheme. We also restate the redundancy conditions in terms of the specific structure of systematic-separate codes.

Assume that we expect up to λ simultaneous errors to occur during computation. From the error model (4.21) and from the definition of $\bigcirc$, the result $[g', t']$ may be written as

$$\begin{aligned} [g', t'] &= [g, t] \bigcirc [\gamma, \tau] && (5.7) \\ &= [g \Box \gamma, t \Diamond \tau] && (5.8) \end{aligned}$$

where $[g, t]$ is the error-free result and $[\gamma, \tau]$ is the net effect of the error on the result. γ and τ represent the errors in the main and parity channels respectively. $[\gamma, \tau]$ is an element of $\mathcal{E}_{\bigcirc}^{(\lambda)}$, the set containing up to λ simultaneous errors.

The key to understanding systematic-separate codes is to determine the form of the syndrome homomorphism ψ. From the definition of ϕ, the subgroup of valid results is given by

$$\tilde{G} = \left\{ [g, \theta(g)] \mid g \in G \right\}. \qquad (5.9)$$

Forming the quotient group $H/\tilde{G}$, we see that an arbitrary coset has the form

$$\begin{aligned} \tilde{G} \square [a,b] &= \left\{ [g \square a, \theta(g) \diamond b] \mid g \in G \right\} & (5.10) \\ &= \left\{ \left[g', \theta(g' \square a^{-1}) \diamond b\right] \mid g' \in G \right\} & (5.11) \\ &= \left\{ \left[g', \theta(g') \diamond \theta(a^{-1}) \diamond b\right] \mid g' \in G \right\} & (5.12) \end{aligned}$$

for some $a \in G$ and $b \in T$. Two cosets $\tilde{G} \bigcirc [a,b]$ and $\tilde{G} \bigcirc [c,d]$ are equivalent if and only if $\theta(a^{-1}) \diamond b = \theta(c^{-1}) \diamond d$. Thus, the entire quotient group consists of the elements

$$\tilde{T} = \{\tilde{t}_a \mid a \in T\} \tag{5.13}$$

where

$$\tilde{t}_a = \left\{ [g, \theta(g) \diamond a] \mid g \in G \right\}. \tag{5.14}$$

The mapping $\tilde{\psi}$ from H onto $H/\tilde{G}$ is found from (5.12), and equals

$$\tilde{\psi}(\,[a,b]\,) = \tilde{T} \bigcirc [a,b] = \tilde{t}_{\theta(a)^{-1} \diamond b}. \tag{5.15}$$

Multiplying cosets, we find that the group product in $\tilde{T}$ is given by

$$\begin{aligned} \tilde{t}_a \bigcirc \tilde{t}_b &= \left\{ [g, \theta(g) \diamond a] \bigcirc [g', \theta(g') \diamond b] \mid g, g' \in G \right\} & (5.16) \\ &= \left\{ [g \square g', \theta(g \square g') \diamond a \diamond b] \mid g, g' \in G \right\} & (5.17) \\ &= \left\{ [g, \theta(g) \diamond a \diamond b] \mid g \in G \right\} & (5.18) \\ &= \tilde{t}_{a \diamond b}. & (5.19) \end{aligned}$$

From this, we recognize that $\tilde{T}$ is isomorphic to the original parity group T. The mapping which computes the syndrome of the result $[g', t']$ is given by

$$\psi(\,[g', t']\,) = \theta(g')^{-1} \diamond t'. \tag{5.20}$$

Detecting and correcting errors in a systematic-separate code is now a matter of computing the syndrome and using it to identity the error. This may be done using a lookup table Λ as is illustrated in Figure 5-1. Note that we do not bother correcting the result t' of the parity channel since we are only interested in the result g of the main channel.

Given the syndrome homomorphism ψ, it is easy to determine redundancy conditions by applying (4.37). Substituting in the definition of ψ, we find that we can detect D and correct C errors if and only if,

$$\theta(\gamma_1)^{-1} \diamond \tau_1 \neq \theta(\gamma_2)^{-1} \diamond \tau_2 \qquad \text{for all} \qquad [\gamma_1, \tau_1] \in \mathcal{E}_{\bigcirc}^{(C)} \neq [\gamma_2, \tau_2] \in \mathcal{E}_{\bigcirc}^{(D)}. \tag{5.21}$$

In many cases, the complexity of the main channel far outweighs that of the parity channel. In these instances, we might expect the probability of failure in the main channel to be much larger than the probability of failure in the parity channel, and we could model the parity channel as being error-free. Errors would then be of the form

$$\mathcal{E}_{\bigcirc} = \left\{ [\gamma, 0_{\diamond}] \mid \gamma \in \mathcal{E}_{\square} \right\} \tag{5.22}$$

where $\mathcal{E}_{\square}$ is the set of all possible single errors occurring in the main channel. In order to detect D and correct C errors of this form requires that

$$\theta(\gamma_1) \neq \theta(\gamma_2) \qquad \text{for all} \qquad \gamma_1 \in \mathcal{E}_{\square}^{(C)} \neq \gamma_2 \in \mathcal{E}_{\square}^{(D)}. \tag{5.23}$$

An alternate method of analyzing systematic-separate codes is to study undetectable errors. An error $[\gamma, \tau]$ is undetectable if it leads to a syndrome that equals $0_{\diamond}$. This occurs when $\theta(\gamma) = \tau$. These errors have a balanced effect on the main and parity channels and corrupt the result in a manner such that it appears to be error-free. If a fault corrupts the main channel only, such that the error is of the form $[\gamma, 0_{\diamond}]$, then it is undetectable if and only if $\theta(\gamma) = 0_{\diamond}$. This implies that $\gamma \in K_{\theta}$. Undetectable errors of this type do not affect which coset the result of the main channel lies in. If a fault corrupts the parity channel only, such that the error is of the form $[0_{\square}, \tau]$, then the error is always detectable since it leads to a nonzero syndrome.

5.4 Multiple Parity Channels

We have thus far been analyzing systems protected by a single parity channel. We now discuss how several parity channels can be combined to protect against a wider range of errors. Our main result is that a system protected by multiple parity channels is always isomorphic to a system protected by a single parity channel of higher order.

Consider a system containing N independent parity channels as shown in 5-2. This figure, for simplicity, illustrates the protection of a single group product. Multiple products may be protected in the same manner. The underlying structure of this system is identical to that of Figure 5-1 except that a total of N parity channels are used to check the result. Denote the parity group used by the $i^{\underline{th}}$ parity channel by T_i and denote the $i^{\underline{th}}$ parity mapping by θ_i. Also denote the kernel of the $i^{\underline{th}}$ parity mapping by K_{θ_i} and the $i^{\underline{th}}$ syndrome by S_i. Since the parity channels are independent and parallel, and each performs a group operation, the collection of parity channels is itself a group T which is isomorphic to the direct product $T_1 \times T_2 \times \cdots \times T_N$.

It would be convenient if the multiple parity system were equivalent to a system protected by a single parity channel utilizing group T. This, however, is not the case since the overall homomorphism from G to $T_1 \times T_2 \times \cdots \times T_N$ is not necessarily onto. Instead, we must analyze this system by determining the kernel K_θ of the overall homomorphism from G to $T_1 \times T_2 \times \cdots \times T_N$. In order for $g \in G$ to be in K_θ, g must be in K_{θ_i} for all i. Therefore,

$$K_\theta = K_{\theta_1} \cap K_{\theta_2} \cap \cdots \cap K_{\theta_N}. \tag{5.24}$$

The performance of the multiple parity channel system is equivalent to a system with a single parity group $T \approx G/K_\theta$.

Although multiple parity channels are isomorphic to a single channel, they may have several advantages. The multiple channel system decomposes the parity computation into operations in N smaller groups. This yields more efficient, less complex parity operations in a manner similar to residue number systems [15]. By selecting kernels K_{θ_i} containing common elements, it is possible to incorporate redundancy directly into the parity channel. This redundancy, though not providing additional coverage for the main channel, can be

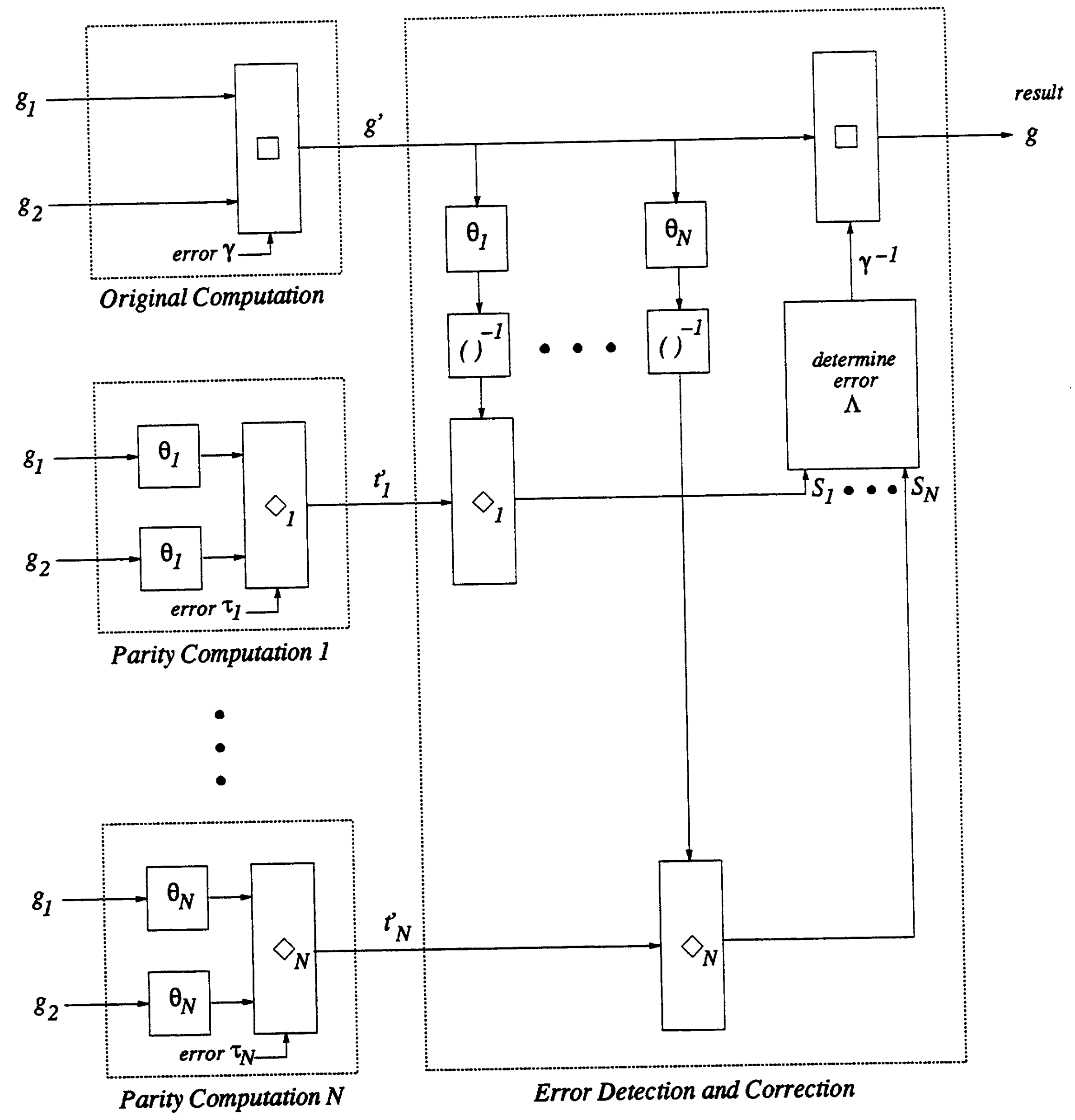


Figure 5-2: Diagram showing the architecture of a system protected via multiple parity channels. The example shows a single group product protected by N independent parity channels.

used to detect and correct errors within the parity channels. Lastly, the multiple channel architecture will have a different set of parity channel errors than the single channel system. This set may lend itself more readily to error detection and correction.

5.5 Application to Other Algebraic Systems

This section discusses the construction of systematic-separate codes in other algebraic systems. We cover rings, fields, and vector spaces, which are algebraic systems containing two distinct operations. The general form of the computation which we desire to protect is some arbitrary sequence of these operations applied to the input operands. We will show that systematic-separate codes may be defined that are capable of protecting computation of this form. The essential properties of systematic-separate codes are unchanged from those for groups. The only major difference is the manner in which homomorphisms are found. A quotient-type construction is still used, but other algebraic structures take the place of subgroups.

5.5.1 Rings

The kernels of ring homomorphisms are algebraic structures known as ideals.

Definition 11. An *ideal* N of a ring $G = [\mathcal{G}; \square, \boxtimes, 0_\square]$ is a subgroup under $\square$ with the additional property that

$$n \boxtimes g \in N \text{ and } g \boxtimes n \in N \qquad \text{for all} \qquad n \in N \text{ and } g \in G. \tag{5.25}$$

Quotient rings are constructed as G/N, and by finding all ideals, we are guaranteed to find all ring homomorphisms. Error detection and correction reduces to a syndrome test, with the same structure shown in Figure 5-1.

5.5.2 Fields

As in rings, the kernels of field homomorphisms are ideals. Unfortunately, a field contains only trivial ideals, $N = G$ and $N = \{0_\square\}$, and thus *a nontrivial systematic-separate* *code cannot be defined.* The only alternative is to use modular redundancy or to embed computation in a larger ring using a nonsystematic code.

5.5.3 Vector Spaces

Systematic-separate codes are of special interest in vector spaces because a nonsystematic encoding may be computationally expensive. The kernels of vector space homomorphisms are algebraic structures known as subspaces.

Definition 12. Let $G = [\mathcal{G}; \Box, 0_\Box]$ be a vector space over F. A *subspace* N of G is a set such that

$$\alpha g_1 \Box \beta g_2 \in N \qquad \text{for all} \qquad \alpha, \beta \in F \text{ and } g_1, g_2 \in N. \tag{5.26}$$

Quotient spaces are constructed as G/N, and by finding all subspaces, we are guaranteed to find all systematic-separate codes.

Quotient spaces are easily described for the vector space $F^{(N)}$. The homomorphism from $F^{(N)}$ onto $F^{(C)}$, $C \leq N$, is always isomorphic to the vector-matrix multiplication

$$\underline{t} = \theta(\underline{g}) = \underline{g}\Theta \tag{5.27}$$

where Θ is an $N \times C$ dimensional matrix of rank C. The kernel of θ corresponds to the left nullspace of Θ. Because of the flexibility in choosing subspaces as kernels, the set of all possible homomorphisms is equivalent to the set of all rank C matrices of dimension $N \times C$. This large set of available homomorphisms allows codes to be constructed that protect against a wide variety of errors. Also, the structure of these code is similar to that of linear error-correcting codes, and we may apply results from this area in a straightforward manner.

5.6 Examples

We now illustrate our theory through several examples. Our intention is to derive the form of the redundancy in a wide variety of systems by using a quotient-type construction. Many of these examples are equivalent to coding schemes already presented by other authors. The power of our approach is that it is not only able to derive the form of redundancy in these systems, but also in many instances, shows that no other coding schemes exist. For simplicity, most of the examples deal with protecting a single binary operation, but may be extended to multiple operations in a natural manner.

5.6.1 Trivial Codes

We begin by giving two coding schemes that can be applied to any group. Every group G contains two trivial subgroups, $N = \{0_\square\}$ and $N = G$, and these yield two systematic-separate codes. First, consider $N = \{0_\square\}$. Then $\tilde{T} = G/N$ is isomorphic to G. Computation in the parity channel is isomorphic to the main channel, and this corresponds to double modular redundancy. Second, when $N = G$, then $\tilde{T} = G/N \approx [\{0_\square\}; \square, 0_\square]$. θ maps all elements to the identity $0_\square$. Although defining a valid homomorphism, this encoding has no error detecting or correcting ability.

Trivial codes have exact counterparts in rings, fields, and vector spaces.

5.6.2 Integer Residue Codes

Let $G = \mathbb{Z}^+$ be the ring of integers under standard integer addition and multiplication. Ideals of G are of the form

$$N = \{0, \pm M, \pm 2M, \ldots\} \tag{5.28}$$

where M is an integer. If $M = 0$ or $M = 1$, then the kernel defines a trivial code, and we assume in the following that $M \geq 2$. Cosets of N in G are of the form

$$N + g = \{g, g \pm M, g \pm 2M, \ldots\}. \tag{5.29}$$

Since elements of the kernel are evenly spaced by M, the cosets $N+g$ and $N+(g+iM)$ are equal for all integers i. Thus, there are only M unique cosets, and we denote them by

$$\begin{aligned}
\tilde{t}_0 &= \quad N+0 &&= \{\,0, \pm M, \pm 2M, \ldots\,\} \\
\tilde{t}_1 &= \quad N+1 &&= \{\,1, 1\pm M, 1\pm 2M, \ldots\,\} \\
&\vdots \\
\tilde{t}_{(M-1)} &= N+(M-1) &&= \{\,(M-1), (M-1)\pm M, (M-1)\pm 2M, \ldots\,\}.
\end{aligned}$$

The mapping $\tilde{\theta}$ from G onto $\tilde{T}$ maps $g \in G$ to the coset containing g, and is given by

$$\tilde{\theta}(g) = \tilde{t}_{\langle g \rangle_M}. \tag{5.30}$$

The sum $\tilde{\Diamond}$ and product $\tilde{\diamondsuit}$ in $\tilde{T}$ can be shown to be

$$\tilde{t}_i \tilde{\Diamond} \tilde{t}_j = (N+i)+(N+j) = N + \langle i+j \rangle_M = \tilde{t}_{\langle i+j \rangle_M} \tag{5.31}$$

$$\tilde{t}_i \tilde{\diamondsuit} \tilde{t}_j = (N+i)(N+j) = N + \langle ij \rangle_M = \tilde{t}_{\langle ij \rangle_M}. \tag{5.32}$$

Examining $\tilde{T}$, we see that it is isomorphic to $T = \mathbb{Z}_M^+$, the ring of integers $\{0, 1, \ldots, M-1\}$ under modulo M addition and multiplication. This isomorphism is accomplished by mapping coset $\tilde{t}_i$ to the integer i. The mapping θ from G to T equals

$$t_g = \theta(g) = \langle g \rangle_M \tag{5.33}$$

and the parity operations equal

$$t_i \Diamond t_j = \langle t_i + t_j \rangle_M \tag{5.34}$$

$$t_i \diamondsuit t_j = \langle t_i t_j \rangle_M. \tag{5.35}$$

This fault-tolerance scheme is equivalent to a residue checksum modulo M. In fact, we have just shown that the only systematic-separate code capable of protecting integer addition and multiplication is a residue checksum. The only flexibility which we have is in the choice of M. This is the same result obtained by Peterson [47], but we have obtained it through a group-theoretic argument.

Let g' and t' denote the possibly faulty results of the main and parity channels. In order to detect and corrects errors, we compute the syndrome

$$S = t' - \langle g' \rangle_M . \tag{5.36}$$

If $S = 0$ we declare that the result is correct. Otherwise, the value of S identifies the specific error. Since the parity ring contains $M - 1$ nonzero values, the system can correct up to $M - 1$ different errors.

If G were instead the finite ring $\mathbb{Z}_N^+$ containing N elements, then we can apply a similar checking scheme. The only difference is that we are now constrained in the choice of the modulus M. In order for (5.33) to define a valid homomorphism, M must divide N.

5.6.3 Real Residue Codes

Residue codes can also be defined to protect the addition of real numbers. Let $G = \mathbb{R}^+ = \left[\mathbb{R}; +, 0\right]$ be the group of real numbers where $+$ denotes normal real number addition. Subgroups of G are also of the form shown in (5.28) where M is now a real number. The quotient group $\tilde{T} = G/N$ contains an infinite number of elements and we represent its members by the cosets

$$\{\tilde{t}_g \mid 0 \leq g < M\} \tag{5.37}$$

where

$$\tilde{t}_g = N + g. \tag{5.38}$$

The mapping $\tilde{\theta}$ becomes

$$\tilde{\theta}(g) = \tilde{t}_{\langle g \rangle_M^{\mathbb{R}}} \tag{5.39}$$

where $\langle g \rangle_M^{\mathbb{R}}$ denotes the real number residue of g modulo M which we define as

$$\langle g \rangle_M^{\mathbb{R}} = g - pM \tag{5.40}$$

where p is an integer chosen such that $0 \leq \langle g \rangle_M^{\mathbb{R}} < M$. The product in $\tilde{T}$ is given by

$$\tilde{t}_a \tilde{\diamond} \tilde{t}_b = \tilde{t}_{\langle a+b \rangle_M^{\mathbb{R}}}. \tag{5.41}$$

Examining (5.39) and (5.41) we see that $\tilde{T}$ is isomorphic to the group

$$T = \left[\{ t \mid 0 \leq t < M \} ; \langle + \rangle_M^{\mathbb{R}}, 0 \right]. \tag{5.42}$$

The mapping from G to T is given by

$$t_g = \theta(g) = \langle g \rangle_M^{\mathbb{R}} \tag{5.43}$$

and the product of elements in T equals

$$t_i \diamond t_j = \langle t_i + t_j \rangle_M^{\mathbb{R}}. \tag{5.44}$$

This coding scheme is essentially a residue checksum defined over the real numbers.

Residue codes can also be defined over the group $\mathbb{C}^+$ of complex numbers under addition. Since $\mathbb{C}^+$ is isomorphic to the direct product $\mathbb{R}^+ \times \mathbb{R}^+$, the parity channel is isomorphic to two real residue codes; one defined on the real part, and one on the imaginary part. Ignoring trivial codes in either channel, the general form of the code is

$$T = [\{[r_1, r_2] \mid 0 \leq r_1 < M_1, 0 \leq r_2 < M_2\} ; \diamond, [0,0]] \tag{5.45}$$

$$t_g = [t_{g,r}, t_{g,i}] = \theta(g) = \left[\langle \mathrm{Re}[g] \rangle_{M_1}^{\mathbb{R}}, \langle \mathrm{Im}[g] \rangle_{M_2}^{\mathbb{R}} \right] \tag{5.46}$$

and

$$[t_{a,r}, t_{a,i}] \diamond [t_{b,r}, t_{b,i}] = \left[\langle t_{a,r} + t_{b,r} \rangle_{M_1}^{\mathbb{R}}, \langle t_{a,i} + t_{b,i} \rangle_{M_2}^{\mathbb{R}} \right] \tag{5.47}$$

where M_1 and M_2 are arbitrary positive real numbers.

These two examples illustrate how addition of real or complex numbers may be protected by a residue code. In both cases, we have identified all possible subgroups and thus we have identified the form of all possible coding schemes. One may be tempted, as in integer residue codes, to protect multiplication as well as addition of real or complex numbers. Unfortunately, $\mathbb{R}$ and $\mathbb{C}$ are fields and we therefore cannot simultaneously protect addition and multiplication.

5.6.4 Multiplication of Nonzero Real Numbers

It is possible to protect only the multiplication of real numbers. Let $G = \left[\mathbb{R}^*; \times, 1\right]$ where $\mathbb{R}^*$ denotes the set of nonzero real numbers, and $\times$ is ordinary real number multiplication. For simplicity of notation, we drop the group product in $g_1 \times g_2$ and merely write $g_1 g_2$. Exponents will also have the expected meaning of repeated products.

The group G has several types of subgroups, and each leads to a different coding scheme. Some subgroups are of the following types

$$\begin{aligned}
N_I &= \{1\} \\
N_{II} &= G \\
N_{III} &= \{g \mid g > 0 \text{ and } g \in G\} \\
N_{IV} &= \{-1, 1\} \\
N_V &= \{\ldots, \pm M^{-2}, \pm M^{-1}, \pm 1, \pm M, \pm M^2, \ldots\} \text{ where } M \in G \\
N_{VI} &= \{\ldots, M^{-2}, M^{-1}, 1, M, M^2, \ldots\} \text{ where } M \in G
\end{aligned}$$

As we have already shown, N_I and N_{II} lead to trivial codes.

Now consider the kernel N_{III}. It partitions G into two cosets which we denote by $\tilde{t}_+$ and $\tilde{t}_-$,

$$\tilde{t}_+ = \{g \mid g > 0, g \in G\} \tag{5.48}$$

$$\tilde{t}_- = \{g \mid g < 0, g \in G\}. \tag{5.49}$$

The homomorphism $\tilde{\theta}$ is given by

$$\tilde{\theta}(g) = \begin{cases} \tilde{t}_+ & \text{if } g > 0 \\ \tilde{t}_- & \text{if } g < 0. \end{cases} \tag{5.50}$$

Multiplying cosets, we find that the product $\tilde{\diamond}$ in $\tilde{T}$ obeys

$$\begin{aligned} \tilde{t}_+ \tilde{\diamond} \tilde{t}_+ &= \tilde{t}_+ & \qquad \tilde{t}_+ \tilde{\diamond} \tilde{t}_- &= \tilde{t}_- \\ \tilde{t}_- \tilde{\diamond} \tilde{t}_+ &= \tilde{t}_- & \qquad \tilde{t}_- \tilde{\diamond} \tilde{t}_- &= \tilde{t}_+ . \end{aligned}$$

From this we can see that $\tilde{T}$ is isomorphic to $T = \mathbb{Z}_2^+$, the integers modulo 2. The mapping from G to T is given by

$$\theta(g) = \begin{cases} 0 & \text{if } g > 0 \\ 1 & \text{if } g < 0 \end{cases} \tag{5.51}$$

and the parity operation becomes

$$t_i \diamond t_j = \langle t_i + t_j \rangle_2 . \tag{5.52}$$

This coding scheme is shown in Figure 5-3. Basically, it checks the result using the rule:

positive	×	positive	=	positive
positive	×	negative	=	negative
negative	×	positive	=	negative
negative	×	negative	=	positive.

Errors influence the result through the group product, in this case, multiplication. If an error e occurs in the main channel, then it is undetectable if $e > 0$. We can only detect an error if it affects the sign of the result.

The kernel N_{IV} partitions G into an infinite number of cosets of the form

$$\tilde{t}_g = N_{IV} \times g = \{-g, g\} . \tag{5.53}$$

We will choose the set of cosets $\{\tilde{t}_g \mid g > 0\}$ to represent the elements of $\tilde{T}$. The mapping $\tilde{\theta}$ from G onto $\tilde{T}$ becomes

$$\tilde{\theta}(g) = \tilde{t}_{|g|} \tag{5.54}$$

where $|g|$ denotes the absolute value of g. Multiplying elements of $\tilde{T}$ we find that

$$\tilde{t}_a \tilde{\diamond} \tilde{t}_b = \tilde{t}_{ab} . \tag{5.55}$$

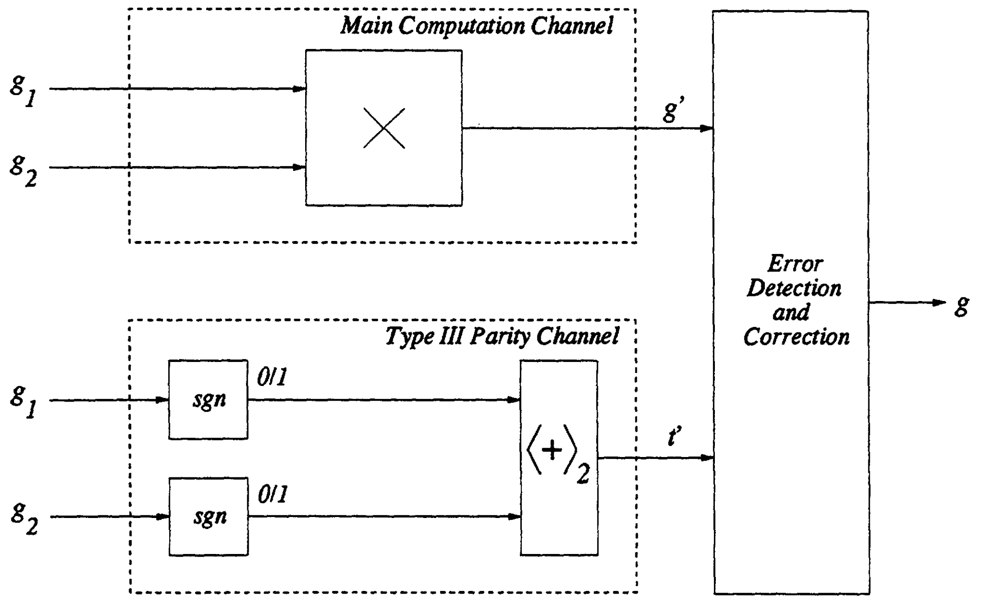


Figure 5-3: This figure illustrates one possible systematic separate coding scheme which may be used to protect the multiplication of nonzero real numbers. The code was derived using the subgroup N_{III} and is equivalent to a sign check of the result.

The group $\tilde{T}$ is isomorphic to the group of positive real numbers under multiplication, $\tilde{T} \approx T = \left[\left\{t \mid t > 0, t \in \mathbb{R}\right\}; \times, 1\right]$. Then

$$t_g = \theta(g) = |g| \tag{5.56}$$

and the group product is real number multiplication of strictly positive values

$$t_a \diamond t_b = t_a t_b. \tag{5.57}$$

This coding scheme is shown in Figure 5-4. In the parity channel, we perform the original multiplication ignoring the signs of the operands. Then we check if this equals the absolute value of the result from the main channel. This scheme can detect all errors in the main channel except $e = -1$. This error affects only the sign, but not the absolute value of the result.

It is interesting to note that a combination of types III and IV parity channels can detect any error in the main computation channel. This is a consequence of the results for

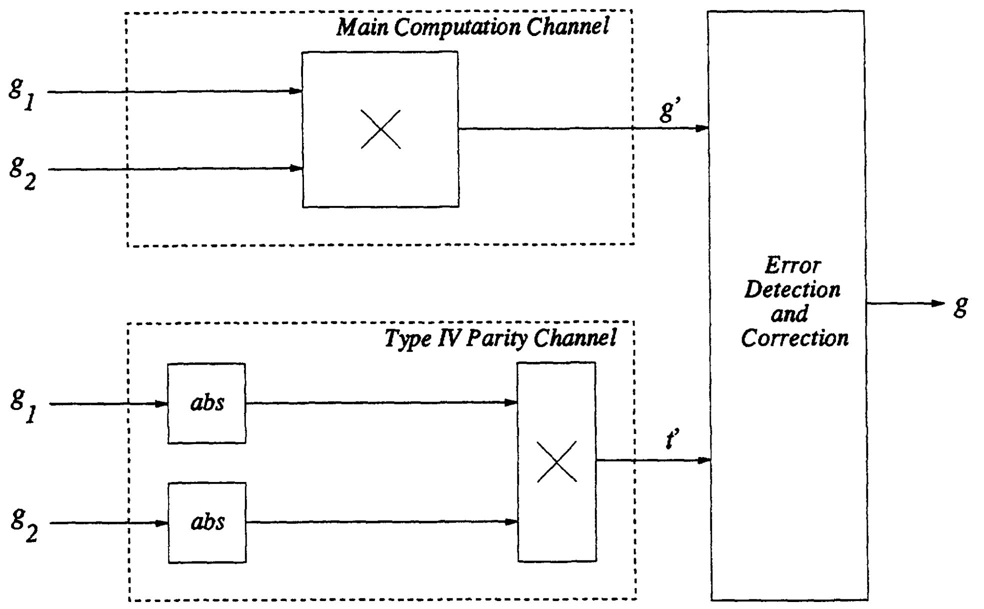


Figure 5-4: Systematic-separate coding scheme derived from the kernel N_{IV} for protecting the product of nonzero real numbers. The code checks only the magnitude of the result.

multiple parity channels. The intersection of these kernels equals

$$N_{III} \cap N_{IV} = N_I = \{1\} \tag{5.58}$$

and the overall parity computation is isomorphic to the original computation in G. This is equivalent to double modular redundancy.

The kernel N_V leads to yet another coding scheme. The cosets $\tilde{T} = G/N_V$ are of the form

$$\tilde{t}_g = N_V \times g = \left\{\ldots, \pm gM^{-2}, \pm gM^{-1}, \pm g, \pm gM, \pm gM^2, \ldots\right\}. \tag{5.59}$$

As before, we choose a convenient set of cosets to represent the elements of the quotient group, $\tilde{T} = \{\tilde{t}_g \mid 1 \leq \tilde{t}_g < M\}$. The mapping $\tilde{\theta}$ is then given by

$$\tilde{t}_g = \tilde{\theta}(g) = \tilde{t}_{\langle\langle |g| \rangle\rangle_M^{\mathrm{R}}} \tag{5.60}$$

where $\langle\langle x \rangle\rangle_M^{\mathrm{R}}$ denotes the multiplicative residue of x modulo M which we define as

$$\langle\langle x \rangle\rangle_M^{\mathrm{R}} = \frac{x}{M^p} \tag{5.61}$$

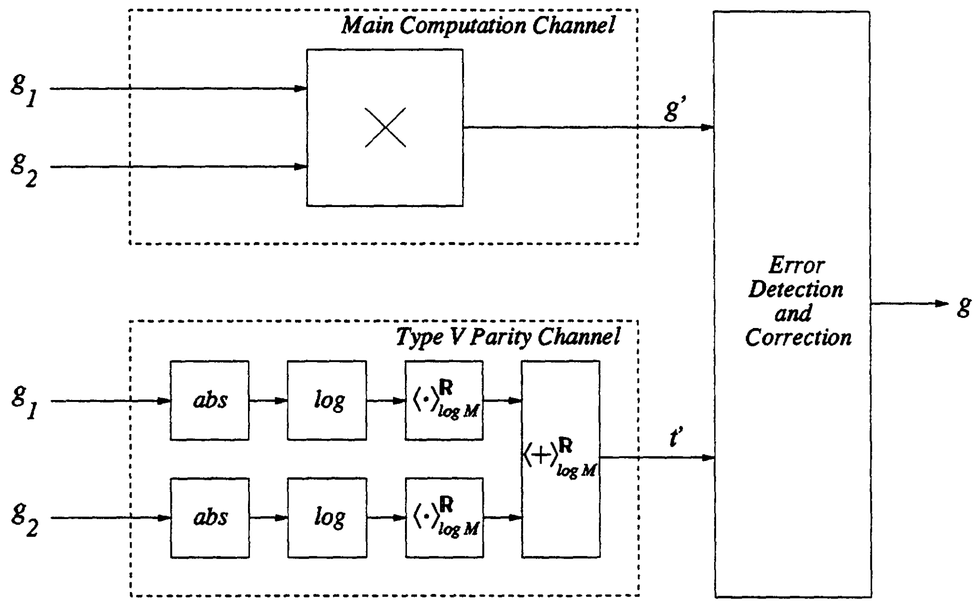

Main Computation Channel
g_1
g_2
$\times$
g'
Type V Parity Channel
abs
log
$\langle\cdot\rangle^{\mathbb{R}}_{\log M}$
$\langle + \rangle^{\mathbb{R}}_{\log M}$
t'
Error Detection and Correction
g

Figure 5-5: Systematic-separate coding scheme derived from the kernel N_V for protecting the product of nonzero real numbers.

where p is an integer chosen such that $1 \leq \langle\langle x \rangle\rangle^{\mathbb{R}}_M < M$. Multiplying cosets we find that

$$\tilde{t}_a \tilde{\diamond} \tilde{t}_b = \tilde{t}_{\langle\langle ab \rangle\rangle^{\mathbb{R}}_M} . \tag{5.62}$$

The operation performed by the parity channel can be more easily understood if we map $\tilde{T}$ to an isomorphic copy $T = \left[\{t \mid 0 \leq t < \log M\} ; \langle + \rangle^{\mathbb{R}}_{\log M} , 0\right]$. The element $\tilde{t}_g \in \tilde{T}$ is renamed to $\log g$. Then

$$t_a \diamond t_b = \langle t_a + t_b \rangle^{\mathbb{R}}_{\log M} \tag{5.63}$$

and the mapping from G to T is given by

$$t_g = \theta(g) = \langle \log |g| \rangle^{\mathbb{R}}_{\log M} . \tag{5.64}$$

The basic structure of this coding scheme is shown in Figure 5-5. By defining T in this manner, we see that this is in essence a real residue code defined on the logarithm of the operands. Also, the signs of the operands are dropped in the parity channel and we work with positive numbers.

The kernel N_{VI} leads to an even more complicated system, and its structure can be understood by noting that

$$N_{VI} = N_{III} \cap N_V. \tag{5.65}$$

Hence, this kernel yields a system which is isomorphic to a parallel combination of types III and V codes.

Thus we see that each type of kernel yields a different coding scheme, and each scheme has different error detecting and correcting properties. The code to use in a specific application depends mainly on the set of errors $\mathcal{E}_O$ that we want to protect against. Code type III can only protect against errors in the sign of the result while the other codes can protect against a wider variety of errors. These other codes, however, are computationally expensive, and further work is needed to refine them into efficient and practical implementations.

5.6.5 Linear Convolution

An important operation in signal processing is the linear convolution of two sequences. Let $a[n]$ and $b[n]$ be P-point sequences which are nonzero for $0 \le n \le P-1$. The linear convolution, denoted by $c[n] = a[n] * b[n]$, results in a $Q = 2P-1$ point sequence $c[n]$ as follows,

$$c[n] = \sum_{i=0}^{P-1} a[i]b[n-i] \qquad \text{for} \qquad n = 0, 1, \ldots, Q-1. \tag{5.66}$$

Assume that the samples are members of a field F. For example, F could be the field of real or complex numbers. Denote by $F[x]$ the set of all polynomials in an indeterminate x with coefficients in F. Then $G = [F[x]; +, \times, 0]$ is a ring with $+$ and $\times$ denoting ordinary polynomial addition and multiplication. We will represent the sequence $a[n]$ as an element of G as follows:

$$a(x) = \sum_{i=0}^{P-1} a[i]x^i \tag{5.67}$$

and will similarly represent $b[n]$ and $c[n]$ by $b(x)$ and $c(x)$. It can be shown that the convolution $c[n] = a[n] * b[n]$ corresponds to the polynomial product

$$c(x) = a(x) \times b(x). \tag{5.68}$$

Thus, if we can protect this ring product, we can protect convolution.

To protect (5.68), we choose a systematic-separate code. The ideals of G are sets of polynomials of the form

$$N = \{g(x)M(x) \mid g(x) \in G\} \tag{5.69}$$

where $M(x)$ is an element of G. We skip the details of this derivation and proceed directly to the final result. We compute the parity symbols

$$\begin{aligned} t_a(x) &= \langle a(x)\rangle_{M(x)} \\ t_b(x) &= \langle b(x)\rangle_{M(x)} \end{aligned} \tag{5.70}$$

where $\langle a(x)\rangle_{M(x)}$ denotes the remainder when $a(x)$ is divided by $M(x)$, and then perform the parity computation

$$t_c(x) = \langle t_a(x) \times t_b(x)\rangle_{M(x)}\,. \tag{5.71}$$

Since all ideals are of the form shown in (5.69), all systematic-separate codes are isomorphic to the one presented here.

To finish the design of a fault-tolerant convolution system, we must choose the polynomial $M(x)$ to protect against the expected set of errors. We can apply existing error coding techniques by noting that (5.70) is the standard method of encoding a systematic cyclic error-correcting code. Furthermore, fast algorithms for detecting and correcting errors exist [14].

The technique outlined in this section is similar to a method proposed by Redinbo [43] and discussed in Section 2.4.5. He protects R-point circular convolution which is equivalent to convolution in the polynomial ring modulo $N(x) = x^R - 1$. Recall from our example of integer residue checksums in Section 5.6.2, that when we changed from the infinite group $\mathbb{Z}^+$ to the finite group $\mathbb{Z}_N^+$, the modulus of the residue channel had to divide N. Similarly, to protect convolution modulo $N(x)$, it can be shown that $M(x)$ must divide $N(x)$. The scheme which we developed in this section is isomorphic to the one proposed by Redinbo, and hence they have equivalent error detecting and correcting properties. Our scheme, however, is preferable since the parity encoding θ, and parity computation are easier to implement.

5.6.6 Linear Transformations

We continue the example of linear transformations which was begun in Section 4.7.5. Recall that the underlying operation which we sought to protect occurs in the N-dimensional vector space $\mathbb{R}^{(N)}$ and consists of the weighted sum of vectors shown in (4.100). As discussed in Section 5.5.3, any parity vector space used to check this computation is isomorphic to a C-dimensional vector space, and the parity encoding is given by (5.27). Thus to protect (4.100), we first compute the parity vectors

$$\underline{t}_i = \underline{l}_i \Theta \qquad \text{for} \qquad i = 1, 2, \ldots, M \tag{5.72}$$

where Θ is a $N \times C$-dimensional matrix with rank C. Then, we compute the original computation along with an identical sequence of parity operations,

$$\underline{q} = \sum_{i=1}^{M} g_i \underline{t}_i. \tag{5.73}$$

(In this example, we will use $\underline{q}$ to denote the result of the parity channel rather than $\underline{t}$.) We can write the main and parity computations jointly as

$$\left[\text{---} \underline{r} \text{---} \mid \text{---} \underline{q} \text{---} \right] = [g_1 \cdots g_M] \left[\begin{array}{c|c} \text{---} \underline{l}_1 \text{---} & \text{---} \underline{t}_1 \text{---} \\ \vdots & \vdots \\ \text{---} \underline{l}_M \text{---} & \text{---} \underline{t}_M \text{---} \end{array} \right]. \tag{5.74}$$

This emphasizes that the sequence of operations applied to the parity vectors is identical to that applied to the vectors $\underline{l}_i$.

This coding scheme is similar to the weighted checksum code used by Jou and described in Section 2.4.2. Errors are modeled as adding some vector $\underline{e}$ to the result of the main channel yielding $\underline{r}' = \underline{r} + \underline{e}$. Error detection consists of computing the syndrome $\underline{s} = \underline{q} - \underline{r}'\Theta$. If $\underline{s} = 0$, then we declare that no errors have occurred. This can detect any error $\underline{e}$ as long as $\underline{e}\Theta \neq 0$. Errors can be corrected if each error has a unique nonzero syndrome.

We can generalize this example from a single to multiple linear transformations, and this leads to yet another practical coding scheme. Suppose that we wish to apply the same linear transformation $\boldsymbol{L}$ to P vectors $\underline{g}_1, \ldots, \underline{g}_P$ obtaining results $\underline{r}_1, \ldots, \underline{r}_P$. Using the approach from the previous paragraphs yields a scheme in which each transformation is independently protected. The overall computation involved equals

$$\left[\begin{array}{c|c} —\underline{r}_1— & —\underline{q}_1— \\ \vdots & \vdots \\ —\underline{r}_P— & —\underline{q}_P— \end{array}\right] = \left[\begin{array}{c} —\underline{g}_1— \\ \vdots \\ —\underline{g}_P— \end{array}\right] \left[\begin{array}{c|c} —\underline{l}_1— & —\underline{t}_1— \\ \vdots & \vdots \\ —\underline{l}_M— & —\underline{t}_M— \end{array}\right] \tag{5.75}$$

where there is a parity $\underline{q}_i$ corresponding to the $i^{\underline{th}}$ result $\underline{r}_i$.

Suppose that each transformation $\underline{r}_i = \underline{g}_i \boldsymbol{L}$ is computed on a separate processor and that a general error model is used. If an error occurs in the processor computing the $i^{\underline{th}}$ transformation, then

$$\underline{r}_i' = \underline{r}_i + \underline{e} \tag{5.76}$$

where $\underline{e}$ is some element of $\mathbb{R}^{(N)}$. The coding scheme, in its present form, is unable to protect against these errors since $\underline{e}\boldsymbol{\Theta} = 0$ for some $\underline{e} \in \mathbb{R}^{(N)}$.

A possible coding scheme to protect against this set of errors may be derived by adding redundancy *between* rows. Rewrite (5.75), without the parity vectors $\underline{t}_i$, using column vector notation as follows

$$\begin{bmatrix} | & & | \\ \tilde{\underline{r}}_1 & \cdots & \tilde{\underline{r}}_N \\ | & & | \end{bmatrix} = \begin{bmatrix} | & & | \\ \tilde{\underline{g}}_1 & \cdots & \tilde{\underline{g}}_M \\ | & & | \end{bmatrix} \begin{bmatrix} | & & | \\ \tilde{\underline{l}}_1 & \cdots & \tilde{\underline{l}}_N \\ | & & | \end{bmatrix}. \tag{5.77}$$

The $i^{\underline{th}}$ column of the result is formed from the weighted some of column vectors

$$\tilde{\underline{r}}_j = \sum_{i=1}^{M} \tilde{l}_{i,j} \tilde{\underline{g}}_i \tag{5.78}$$

where $\tilde{l}_{i,j}$ equals the $i^{\underline{th}}$ element of the column vector $\tilde{\underline{l}}_j$. We can thus consider computation to consist of a sequence of column operations in $\mathbb{R}^{(P)}$, and add redundancy accordingly. We compute the parity column vectors from the columns $\underline{\tilde{g}}_j$ as follows

$$\underline{\tilde{t}}_j = \Theta \underline{\tilde{g}}_j \qquad \text{for} \qquad j = 1, 2, \ldots, M \tag{5.79}$$

where Θ is now a $C \times P$ dimensional matrix of rank C. We then perform computation in the main and parity channels. The overall computation equals

$$\left[\begin{array}{ccc} | & & | \\ \underline{\tilde{r}}_1 & \cdots & \underline{\tilde{r}}_N \\ | & & | \\ \hline | & & | \\ \underline{\tilde{q}}_1 & \cdots & \underline{\tilde{q}}_N \\ | & & | \end{array}\right] = \left[\begin{array}{ccc} | & & | \\ \underline{\tilde{g}}_1 & \cdots & \underline{\tilde{g}}_M \\ | & & | \\ \hline | & & | \\ \underline{\tilde{t}}_1 & \cdots & \underline{\tilde{t}}_M \\ | & & | \end{array}\right] \left[\begin{array}{ccc} | & & | \\ \underline{\tilde{l}}_1 & \cdots & \underline{\tilde{l}}_N \\ | & & | \end{array}\right] \tag{5.80}$$

where the vector $\underline{\tilde{q}}_j$ denotes the result of the $j^{\underline{th}}$ parity channel. We detect errors by computing syndromes for each parity vector

$$\underline{\tilde{s}}_j = \underline{\tilde{q}}_j - \Theta \underline{\tilde{r}}_j \qquad \text{for} \qquad j = 1, 2, \ldots, N. \tag{5.81}$$

If $\underline{\tilde{s}}_j = 0$ for all j, then we declare that the result is error-free.

This example brings out a very important point: the form of the redundancy depends upon how we view the underlying operations occurring in the system (i.e., what algebraic system G we select to model computation). In both cases, the computation which we sought to protect consisted of applying the same linear transformations $\boldsymbol{L}$ to P different operand vectors. At first, we considered the underlying computation to consist of row operations, and this yielded the coding scheme shown in (5.75). Next, we considered the computation to consist of column operations and this led to the coding scheme shown in (5.80).

It is also important to keep in mind that both of these coding schemes define valid homomorphisms, and that either homomorphism may be used independently of exactly which operations are used to compute the actual result. For example, the computation shown in (5.80) does not have to be performed as column operations; we can compute each row of the result independently, as originally suggested, and use the second form of parity information,

$$\begin{bmatrix} \text{---}\underline{r}_1\text{---} \\ \vdots \\ \text{---}\underline{r}_P\text{---} \\ \hline \text{---}\underline{q}_1\text{---} \\ \vdots \\ \text{---}\underline{q}_C\text{---} \end{bmatrix} = \begin{bmatrix} \text{---}\underline{g}_1\text{---} \\ \vdots \\ \text{---}\underline{g}_P\text{---} \\ \hline \text{---}\underline{t}_1\text{---} \\ \vdots \\ \text{---}\underline{t}_C\text{---} \end{bmatrix} \begin{bmatrix} \text{---}\underline{l}_1\text{---} \\ \vdots \\ \text{---}\underline{l}_M\text{---} \end{bmatrix}. \tag{5.82}$$

This flexibility allows redundancy to be added which protects against the expected set of hardware errors. Sufficient redundancy exists to detect and correct errors of the form (5.76).

This coding scheme is equivalent to the one proposed by Musicus and Song and outlined in Section 2.4.3. They give general guidelines for selecting the parity encoding matrix $\boldsymbol{\Theta}$ and describe how to perform error detection and correction efficiently.

An interesting extension to this coding scheme is to combine the row checksums of (5.75) and the column checksums of (5.80). This incorporates a higher amount of redundancy and leads to a more robust system. This combined approach is similar to the weighted matrix checksum code presented by Jou and described in Section 2.4.2.

5.6.7 Gaussian Elimination and Matrix Inversion

As the previous example illustrates, any computation consisting of a weighted sum of vectors may be placed in our framework. Gaussian elimination and matrix inversion are two important computational procedures that may be described in this fashion.

Gaussian elimination is a well-known procedure for the solution of simultaneous linear equations [48]. Let $\boldsymbol{A}$ be an $N \times N$ matrix consisting of the coefficients of the equations to be solved,

$$\boldsymbol{A} = \begin{bmatrix} \text{---}\underline{a}_1\text{---} \\ \vdots \\ \text{---}\underline{a}_N\text{---} \end{bmatrix}. \tag{5.83}$$

Gaussian elimination consists of two steps: elimination and back-substitution. During elimination, we add weighted copies of the first row of $\boldsymbol{A}$ to the other rows in order to eliminate the first coefficient from the remaining equations. Next, we add weighted copies of the second row to rows 3 through N eliminating the second coefficient. This process continues until the matrix is upper triangular. During back-substitution, we add weighted copies of the last row to the other rows in order to eliminate the last coefficient in each row. This procedure continues until the matrix is diagonal.

Gaussian elimination may be protected by a systematic-separate code in a straightforward manner. First compute the a parity vector $\underline{t}_i = \underline{a}_i\boldsymbol{\Theta}$ for each rows of $\boldsymbol{A}$, and append these to the rows of $\boldsymbol{A}$,

$$\tilde{\boldsymbol{A}} = \left[\begin{array}{c|c} \text{---}\underline{a}_1\text{---} & \text{---}\underline{t}_1\text{---} \\ \vdots & \vdots \\ \text{---}\underline{a}_N\text{---} & \text{---}\underline{t}_N\text{---} \end{array}\right]. \tag{5.84}$$

Then proceed with Gaussian elimination of the left half of $\tilde{\boldsymbol{A}}$. For each row operation in $\boldsymbol{A}$, perform the corresponding operation to the neighboring parity row. Once the process is complete, the parity rows may be used to detect and correct errors just as in the previous vector space examples.

Matrix inversion can also be computed using Gaussian elimination. We first form an $N \times 2N$ matrix $\boldsymbol{B}$ containing $\boldsymbol{A}$ on the left half and an $N \times N$ identity matrix on the right. To each row of $\boldsymbol{B}$ we append parity vector $\underline{t}_i = \underline{b}_i\boldsymbol{\Theta}$ where $\underline{b}_i$ denotes the ith row of $\boldsymbol{B}$. This forms the matrix

$$\tilde{\boldsymbol{B}} = \left[\begin{array}{c|ccccc|c} \text{---}\underline{a}_1\text{---} & 1 & 0 & . & \cdots & 0 & \text{---}\underline{t}_1\text{---} \\ \text{---}\underline{a}_2\text{---} & 0 & 1 & 0 & \cdots & 0 & \text{---}\underline{t}_2\text{---} \\ \vdots & & & \ddots & & & \\ \text{---}\underline{a}_N\text{---} & 0 & \cdots & & 0 & 1 & \text{---}\underline{t}_N\text{---} \end{array}\right] = [\boldsymbol{A}|\boldsymbol{I}|\boldsymbol{T}]. \tag{5.85}$$

Next, use Gaussian elimination to eliminate the elements in the left half of $\tilde{\boldsymbol{B}}$. The elimination process transforms the identity matrix $\boldsymbol{I}$ into $\boldsymbol{A}^{-1}$, and the redundancy of the parity code is preserved throughout computation.

These examples of Gaussian elimination of matrix inversion are equivalent to schemes proposed by Jou [25]. Our group-theoretic framework, however, gives us much deeper insights. If we view these computations as operations on rows or columns of vectors, then the parity schemes proposed are the *only* systematic-separate codes available.

5.7 Summary

The main emphasis of this chapter was on developing a procedure for finding possible homomorphisms. We focused on systematic-separate codes, an important class of arithmetic codes characterized by a parity-like operation in a parallel, independent channel. We used a quotient group isomorphism, and reduced the problem of finding systematic-separate codes to that of finding subgroups. The power of our approach is that, in many cases, we are able to determine all possible systematic-separate codes that may be used to protect a given operation. Also, we can show when it is impossible to protect an operation with a code of this form. In these instances, a nonsystematic code or modular redundancy must be used. The results for groups carry over completely to rings, fields, and vector spaces, and we showed that the structure of fields prohibits the definition of systematic-separate codes.

We showed that the error detection and correction algorithms may be reduced to functions of a syndrome, and then reinterpreted redundancy requirements for the special case of systematic-separate codes. A wide variety of examples of the application of our technique were presented, and our focus was on determining homomorphisms and revealing the structure of the parity channel. We showed that in many cases, the results of other authors were special cases of the application of our framework.

In the next two chapters, we will study in detail two specific fault-tolerant systems. For each system, we will present an architecture and derive a realistic fault model. Then we will tailor a code to protect against the expected set of errors.

Chapter 6

Fault-Tolerant Convolution

In this chapter we present a detailed study of a fault-tolerant convolution algorithm. The algorithm is based on a polynomial residue number system, and is a direct extension of integer residue number system coding schemes applied to polynomial rings.

6.1 Introduction

Convolution is an important operation used in many digital signal processing applications. It is the basis of digital filtering and correlation, and performing convolution is often the most computationally intensive step of an algorithm. Extremely high throughput convolution systems use multiprocessor configurations [49]. Computation is distributed over several processors, and each computes a portion of the result. Unfortunately, the large amount of hardware in multiprocessor systems increases the likelihood that a soft or hard failure will occur and corrupt the result. Thus some degree of fault-tolerance is desirable in these systems.

Protection of convolution operations has been previously studied by Redinbo [44]. His approach is discussed in Section 2.4.5, and he applies generalized cyclic error-correcting codes with known minimum distance properties [22]. These codes are basically BCH codes defined over the fields of real or complex numbers. Redundancy is incorporated by pre-multiplying sequences by a generator polynomial. The resulting code is systematic and separate, with parity symbols being processed in an independent channel. As discussed in Section 5.6.5, Redinbo's coding scheme is isomorphic to the systematic-separate code derived by applying our framework to polynomial rings.

The main limitation of Redinbo's work is that the error detecting and correcting properties of generalized cyclic codes only allow errors equal in number to the degree of the generator polynomial to be detected. This dictates that direct convolution, rather than more efficient FFT-based methods, be used.

This chapter presents a novel nonsystematic arithmetic code for protecting convolution that is based on a polynomial residue number system (RNS). Computation is decomposed into independent residue channels and redundancy incorporated by adding extra residue channels. This decomposition yields an algorithm which is computationally equivalent to the Winograd Convolution Algorithm (WCA), and its parallel structure makes it ideally suited for multiprocessor implementations. Furthermore, and most importantly, our fault-tolerant algorithm has the same underlying structure as fast convolution algorithms, which are based on polynomial residue number systems, and this makes our algorithm fast and efficient as well.

The derivation of the fast fault-tolerant algorithm is done in several steps, and its practicality becomes evident only at the very end. In Section 6.2 we present background material in polynomial rings and summarize the WCA. Then in Section 6.3 we add redundancy to the WCA and show how the polynomial RNS defines a partial homomorphism. We then present a fault detection and correction scheme which can handle multiple processor failures. The derivation is very general, and yields a wide variety of implementations. Section 6.4 focuses on single error correction, and we choose moduli such the algorithm may be computed efficiently with FFTs. In Section 6.5 we apply a generalized likelihood ratio test to deal with computational noise inaccuracies, and in Section 6.6 discuss the efficiency of our algorithm. We conclude with Section 6.7 in which we summarize the contributions made in this chapter.

6.2 Winograd Convolution Algorithm

In this section we describe the fundamentals of the Winograd Convolution Algorithm which forms the basis of our fault-tolerant algorithm. The operation we are interested in protecting is the linear convolution of two finite length data sequences. Let $a[n]$ and $b[n]$ be P-point sequences which are non-zero for $0 \leq n \leq P-1$. The linear convolution, denoted by $a[n] * b[n]$, results in a $Q = 2P - 1$ point sequence $c[n]$ as follows:

$$c[n] = \sum_{i=0}^{n} a[i]b[n-i] \qquad \text{for} \qquad n = 0, \ldots, Q-1. \tag{6.1}$$

A key idea we exploit is to represent individual samples as elements of a field F, and entire sequences as elements of a polynomial ring [50]. Let $F[x]$ denote the set of polynomials with coefficients in F. $F[x]$ is a ring under the operations of polynomial addition and multiplication, and is called the ring of polynomials in x over F. We represent the sequence $a[n]$ by the polynomial

$$a(x) = \sum_{i=0}^{P-1} a[i]x^i \tag{6.2}$$

and represent $b[n]$ and $c[n]$ by $b(x)$ and $c(x)$ in a similar fashion. The degree of polynomial $a(x)$, denoted by $\deg a(x)$, refers to the highest power of x in $a(x)$.

Let $M(x)$ be an element of $F[x]$, and denote by $F[x]/M(x)$ the set of polynomials in $F[x]$ with degree less than $\deg M(x)$. $F[x]/M(x)$ is a ring under normal polynomial addition and multiplication modulo $M(x)$. We use the notation $r(x) = \langle a(x) \rangle_{M(x)}$ to represent the remainder when $a(x)$ is divided by $M(x)$. $M(x)$ is called the modulus and $r(x)$ is called the *residue*. The division algorithm for polynomials guarantees the uniqueness of the modulo operation [45].

It is well-known [51] that the polynomial product $c(x) = a(x)b(x)$ is equivalent to the linear convolution in (6.1). Furthermore, the product can be computed in the finite degree polynomial ring $F[x]/M(x)$ by choosing an $M(x)$ such that $\deg M(x) > \deg c(x)$. The modulo operation will not affect the result and a linear convolution will still be computed.

A polynomial residue number system (RNS) [51] is an isomorphic representation of the finite degree polynomial ring $F[x]/M(x)$. It decomposes a large ring into a direct sum of several smaller rings. To define a polynomial RNS isomorphic to $F[x]/M(x)$, we first factor $M(x)$ into N relatively prime polynomials,

$$M(x) = m_1(x)m_2(x)\cdots m_N(x) \tag{6.3}$$

where each $m_k(x)$ is a member of $F[x]/M(x)$. Then, it can be shown that the direct sum of rings $F[x]/m_1(x) \times \cdots \times F[x]/m_N(x)$ is isomorphic to $F[x]/M(x)$. The mapping from $a(x) \in F[x]/M(x)$ to its direct sum representation is accomplished by computing the N residues

$$a_k(x) = \langle a(x) \rangle_{m_k(x)} \qquad \text{for} \qquad k = 1, \ldots, N \tag{6.4}$$

where $a_k(x) \in F[x]/m_k(x)$. We denote this isomorphism by

$$a(x) \approx \{a_1(x), a_2(x), \ldots, a_N(x)\}. \tag{6.5}$$

The inverse mapping from a set of residues $\{a_1(x), a_2(x), \ldots, a_N(x)\}$ to $a(x) \in F[x]/M(x)$ is computed by the Chinese Remainder Theorem (CRT) for polynomials:

$$a(x) = \sum_{k=1}^{N} \langle a_k(x) D_k(x) \rangle_{m_k(x)} M_k(x) \tag{6.6}$$

where

$$M_k(x) = \frac{M(x)}{m_k(x)} \tag{6.7}$$

and $D_k(x)$ is chosen such that

$$\langle M_k(x) D_k(x) \rangle_{m_k(x)} = 1. \tag{6.8}$$

The isomorphism between $F[x]/M(x)$ and its direct sum allows us to perform computation in $F[x]/M(x)$ by operations in each of the smaller rings $F[x]/m_1(x), \ldots, F[x]/m_N(x)$. The isomorphism holds for both ring operations. Let $a(x)$ and $b(x)$ be elements of $F[x]/M(x)$ with residue representations,

$$a(x) \approx \{a_1(x), \ldots, a_N(x)\} \qquad \text{and} \qquad b(x) \approx \{b_1(x), \ldots, b_N(x)\}. \tag{6.9}$$

The residue representation of the sum $\langle a(x) \pm b(x) \rangle_{M(x)}$ or product $\langle a(x) b(x) \rangle_{M(x)}$ can be computed by N independent residue additions or multiplications:

$$\langle a(x) \pm b(x) \rangle_{M(x)} \quad \approx \quad \{a_1(x) \pm b_1(x), \ldots, a_N(x) \pm b_N(x)\} \tag{6.10}$$

$$\langle a(x)b(x)\rangle_{M(x)} \approx \left\{\langle a_1(x)b_1(x)\rangle_{m_1(x)}, \ldots, \langle a_N(x)b_N(x)\rangle_{m_N(x)}\right\}. \qquad (6.11)$$

A polynomial RNS is the basis of the Winograd Convolution Algorithm (WCA). We have already discussed the main steps involved, and we summarize them here for clarity. To compute the linear convolution $c(x) = a(x)b(x)$, choose an $M(x)$ such that $\deg M(x) > \deg c(x)$. Define an RNS by factoring $M(x)$ into N co-prime polynomials. (Both of these steps are done off-line). Then, given the sequences $a(x)$ and $b(x)$, compute the two sets of residues (6.9). Perform the residue multiplications in each of the smaller rings (6.11), and then reconstruct using the CRT (6.6). The WCA thus computes a long convolution using N simpler polynomial products in independent residue channels.

Convolving two sequences of length P using direct convolution (6.1) requires $\mathcal{O}(P^2)$ operations. If we use the WCA instead and choose the moduli polynomials carefully, then convolution can be computed with as little as $\mathcal{O}(P \log_2 P)$ operations. The savings over direct convolution can be substantial.

6.3 Fault-Tolerant System

In this section we incorporate redundancy in the WCA to yield a robust algorithm. The WCA is mapped onto a multiprocessor architecture and a suitable error model is derived. We then show the connection between the system developed in this chapter and the results of Chapter 4. Then an algorithm for detecting and correcting multiple processor failures is presented.

We add redundancy to the WCA by adding extra residue channels and constraining the length of the convolution. We start with a Q-point linear convolution computed using N moduli, and add C extra moduli $m_{N+1}(x), \ldots, m_{N+C}(x)$. These moduli must be co-prime to each other and to the original N moduli. Since $N + C$ residues are used, computation is isomorphic to the larger ring $F[x]/M^+(x)$ where $M^+(x) = \prod_{k=1}^{N+C} m_k(x)$. In $F[x]/M^+(x)$ a convolution of length $Q^+ = \sum_{k=1}^{N+C} \deg m_k(x)$ could be computed. However, we restrict the lengths of the input sequences such that the result has a length $Q \leq \sum_{k=1}^{N} \deg m_k(x)$. Only a portion of the allowable output length is used and the $Q^+ - Q$ high order coefficients of the result should be zero.

6.3.1 Multiprocessor Architecture

We distribute computation of the WCA in a multiprocessor system such that one residue is corrupted per processor failure. We accomplish this by performing the computation needed for each residue channel on a separate, independent processor. Thus, we assume that $N+C$ processors are available, and assign to the $k^{\underline{th}}$ processor the computation of the $k^{\underline{th}}$ input residues

$$a_k(x) = \langle a(x) \rangle_{m_k(x)} \tag{6.12}$$

$$b_k(x) = \langle b(x) \rangle_{m_k(x)} \tag{6.13}$$

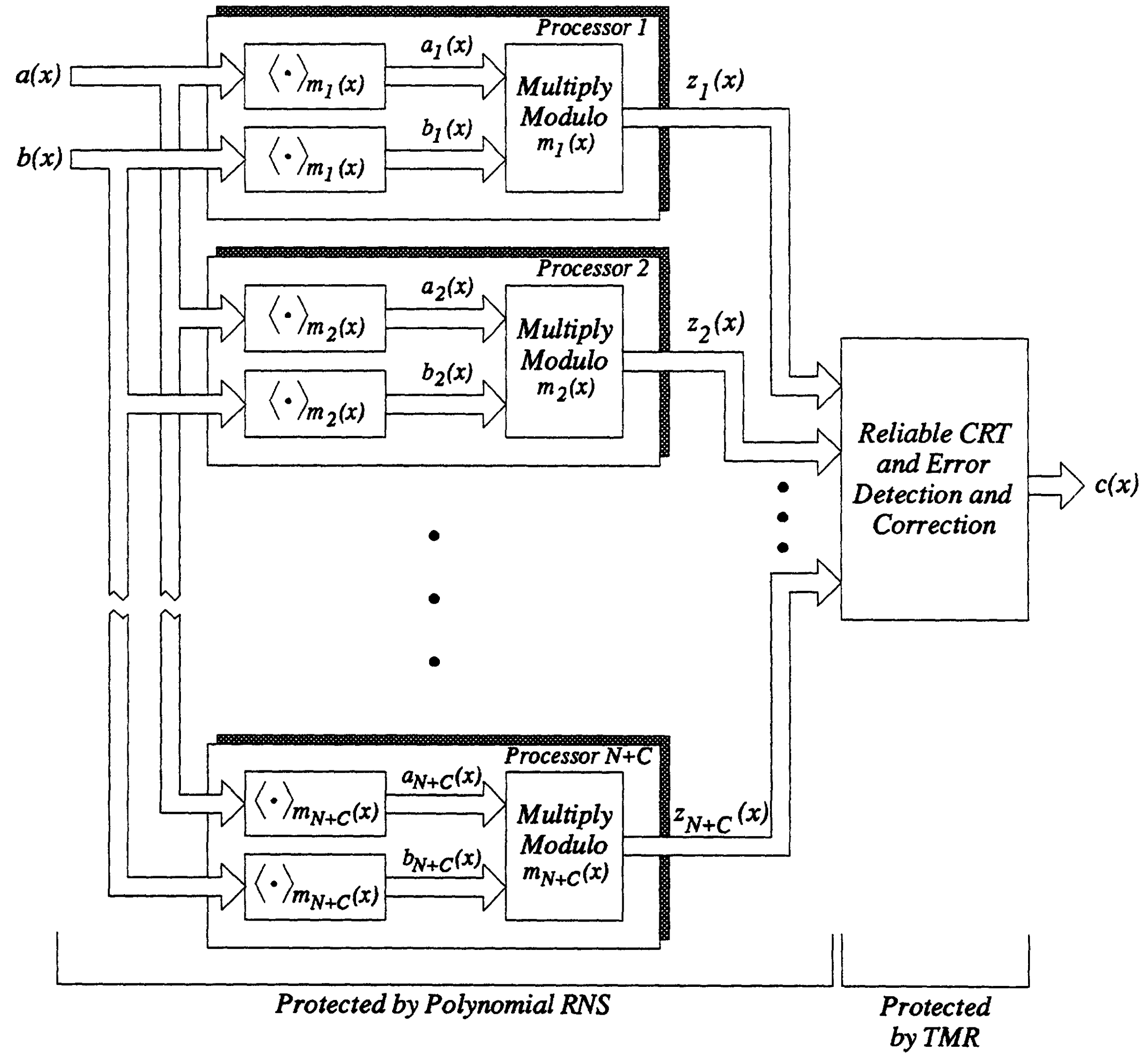
as well as the $k^{\underline{th}}$ residue product

$$c_k(x) = \langle a_k(x) b_k(x) \rangle_{m_k(x)} \,. \tag{6.14}$$

Our approach can only protect against errors in the first two steps of the WCA. To ensure proper computation of the CRT reconstruction and error detection and correction, we assume that Triple Modular Redundancy (TMR) is used in these steps. This hybrid approach is practical since the bulk of computation occurs during the first two steps of the WCA. A diagram of the overall system architecture is shown in Figure 6-1.

Our chief concern is guarding against failures in the $N+C$ residue processors, and so we assume that data I/O and interprocessor communication are reliable. These functions can be protected using standard techniques such as binary error-correcting codes or triplicated buses. We declare that a processor has failed when it does not compute the correct result given its input. This model covers a wide range of possible processor failures including transient single bit arithmetic errors as well as complete processor failure. We also assume that when a processor fails, it corrupts only the computation assigned to it, and does not affect the communication network or any other processor. With this model, one residue will be corrupted per processor failure.

We denote the outputs of the processors by $z_k(x)$, and assume that λ failures occur in



a(x)
b(x)
Processor 1
$\langle\cdot\rangle_{m_1(x)}$
$a_1(x)$
$b_1(x)$
Multiply Modulo $m_1(x)$
$z_1(x)$
Processor 2
$\langle\cdot\rangle_{m_2(x)}$
$a_2(x)$
$b_2(x)$
Multiply Modulo $m_2(x)$
$z_2(x)$
Processor N+C
$\langle\cdot\rangle_{m_{N+C}(x)}$
$a_{N+C}(x)$
$b_{N+C}(x)$
Multiply Modulo $m_{N+C}(x)$
$z_{N+C}(x)$
Reliable CRT and Error Detection and Correction
c(x)
Protected by Polynomial RNS
Protected by TMR

Figure 6-1: Robust multiprocessor architecture used to compute convolutions. Computation is first divided into $N+C$ residue operations, each of which is computed by an independent processor. The outputs of these processors are fed to a reliable system which computes the CRT reconstruction and performs error detection and correction.

processors $\{k_1, \ldots, k_\lambda\}$. The processor outputs will have value

$$z_k(x) = \begin{cases} c_k(x) + \phi_k(x) & \text{for} \quad k = k_1, \ldots, k_\lambda \\ c_k(x) & \text{else} \end{cases} \tag{6.15}$$

where $\phi_{k_i}(x)$ is the net effect of the failure in channel k_i, and $\deg \phi_{k_i}(x) < \deg m_{k_i}(x)$. Using this set of residues, the output of the reliable CRT step is

$$z(x) = \sum_{k=1}^{N+C} \left\langle z_k(x) D_k^+(x) \right\rangle_{m_k(x)} M_k^+(x) \tag{6.16}$$

where now

$$M_k^+(x) = \frac{M^+(x)}{m_k(x)} \tag{6.17}$$

and $D_k^+(x)$ is chosen such that

$$\left\langle D_k^+(x) M_k^+(x) \right\rangle_{m_k(x)} = 1. \tag{6.18}$$

Substituting (6.15) into (6.16) gives

$$z(x) = c(x) + \sum_{i=1}^{\lambda} \left\langle \phi_{k_i}(x) D_{k_i}^+(x) \right\rangle_{m_{k_i}(x)} M_{k_i}^+(x) \tag{6.19}$$

and we see that processor failures affect the result in an additive manner.

6.3.2 Relationship to Group-Theoretic Framework

The redundant polynomial residue number system code that we use is related to the group-theoretic framework of Chapter 4, and satisfies the postulates of a partial homomorphism. Computation occurs in a polynomial ring, $F[x]/M(x)$, redundancy is added by embedding computation in a larger ring, $F[x]/M^+(x)$, and an additive error model (6.19) is assumed.

The importance of using a polynomial residue number system to compute convolution is two-fold. First, it allows convolution to be computed efficiently by decomposing computation into smaller independent convolutions. Second, the fault mechanism (6.19) generates errors which may be protected against by the redundancy. Any convolution algorithm which generates errors of the form shown in (6.19) may be used instead of the WCA.

The mapping ϕ from $F[x]/M(x)$ to $F[x]/M^+(x)$ which adds redundancy is very simple, $\phi(a(x)) = a(x)$. We treat the polynomial $a(x)$ as an element of $F[x]/M^+(x)$ and perform arithmetic modulo $M^+(x)$. The desired convolution is computed as long as the result has length $Q \leq \sum_{k=1}^{N} \deg m_k(x)$. We will constrain the lengths of the input sequences such that this is true. The subset of valid results is given by

$$\mathcal{H}_V = \{c(x) \in F[x]/M^+(x) \mid \deg c(x) < Q\}. \tag{6.20}$$

Over this set of outputs, it is seen that the mapping $\phi(a(x)) = a(x)$ satisfies (4.50) and is thus a partial homomorphism.

It is assumed that errors affect the result in an additive manner as shown in (6.19). We will assume that $\phi_{k_i}(x)$ is an arbitrary element of $F[x]/m_{k_i}(x)$, and therefore the errors are symmetric. We are thus able to measure the redundancy in this code via its minimum distance. We will pursue this in the following section and will also present an algorithm for detecting and correcting multiple errors.

6.3.3 Fault Detection and Correction

We now develop an algorithm that detects and corrects multiple processor failures by examining the result $z(x)$. The algorithm is complicated by the fact that we must determine the exact number and location of failed processors. Let D be the largest nonnegative integer satisfying

$$\deg\left[\frac{M^+(x)}{m_{l_1}(x)\cdots m_{l_D}(x)}\right] \geq Q \text{ for every set of } D \text{ unique moduli } \{m_{l_1}(x),\ldots,m_{l_D}(x)\}. \tag{6.21}$$

D measures the amount of redundancy present in the polynomial RNS. It serves the same purpose as the minimum distance of a binary error-correcting code. To change a valid set of residues into another valid set, at least $D+1$ residues must be modified. The specific value of D depends on the redundant moduli $m_{N+1}(x),\ldots,m_{N+C}(x)$. If the degrees of the redundant moduli are all greater than or equal to the degrees of the original N moduli,

$$\deg m_{N+i}(x) \geq \deg m_j(x) \qquad \text{for} \qquad \begin{cases} i = 1, \ldots, C \\ j = 1, \ldots, N \end{cases} \tag{6.22}$$

then $D = C$.

The properties of a polynomial RNS with a given value of D are described in the following two theorems which are proved in Appendix 6.A.

Theorem 1 – Fault Detection

Let D satisfy (6.21). Then

a) If no failures occur ($\lambda = 0$), then $\deg z(x) < Q$ and the correct convolution is $c(x) = z(x)$.

b) If between 1 and D failures occur ($1 \leq \lambda \leq D$), then $\deg z(x) \geq Q$.

Theorem 2 – Fault Correction

Decompose D as $D = \alpha + \beta$ for some integers $\alpha \geq \beta \geq 0$. Assume that no more than α processors can fail at any time. Then:

a) We can reliably distinguish when β or fewer errors occur from the case when between $\beta + 1$ and α failures occur.

b) If β or fewer failures occur, we can correct them by examining $z(x)$.

The decomposition $D = \alpha + \beta$ presented in Theorem 2 is not arbitrary, but determines the error detecting and correcting ability of the code. α is the maximum number of simultaneous processor failures which we will attempt to detect, while β is the maximum number we will attempt to correct. Since a failure must be detected in order to be corrected, α must be greater than or equal to β. A key issue is that we must make an assumption about the maximum number of simultaneous processor failures which can occur. Then we add sufficient redundancy to attain a desired level of protection. For example, suppose that we anticipate at most 2 processors failing at one time. Then we could do any of the following:

1. Detect up to 2 processor failures, but correct none of them ($D = 2$, $\alpha = 2$, $\beta = 0$).

2. Detect up to 2 processor failures. If 1 processor failed, we can determine this and correct the result ($D = 3$, $\alpha = 2$, $\beta = 1$).

3. Detect and correct up to 2 processor failures ($D = 4$, $\alpha = 2$, $\beta = 2$).

The minimum redundancy needed for single fault detection is $D = 1$, and this can be accomplished by $C = 1$ extra modulus which satisfies

$$\deg m_{N+1}(x) \geq \deg m_i(x) \qquad \text{for} \qquad i = 1, \ldots, N. \tag{6.23}$$

For single fault detection and correction we need $D = 2$, and this can be satisfied by $C = 2$ extra moduli satisfying

$$\deg m_{N+1}(x), \deg m_{N+2}(x) \geq \deg m_i(x) \qquad \text{for} \qquad i = 1, \ldots, N. \tag{6.24}$$

In general, with C extra moduli satisfying (6.22), we can simultaneously detect and correct at most $\lfloor C/2 \rfloor$ faults, where $\lfloor C/2 \rfloor$ is the largest integer not greater than $C/2$. This result is equivalent to the error detecting and correcting ability of a distance $C + 1$ binary error-correcting code [14].

Theorem 2 ensures that errors may be corrected but does not state how to perform the actual error correction. This is given in the following theorem which determines the exact number and location of the faulty processors, and during this process, corrects the output. The proof of this theorem is also given in Appendix 6.A.

Theorem 3 – Fault Correction Algorithm

Suppose the moduli satisfy the conditions (6.21) and assume that $\lambda \leq \alpha$ failures occurred in processors $\{k_1, \ldots, k_\lambda\}$. Then:

```
1) Given the (possibly faulty) residues $z_k(x)$, use a reliably implemented CRT
   to reconstruct the corresponding sequence $z(x)$.
2) If $\deg z(x) < Q$ then no fault has occurred, so $c(x) = z(x)$. STOP.
3) Otherwise a fault has occurred.
       For $p = 1, \ldots, \beta$
           For all possible sets of $p$ processors $1 \leq j_1 < j_2 \cdots < j_p \leq N + C$
               Compute $r_{j_1,\ldots,j_p}(x) = \langle z(x) \rangle_{M^+_{j_1,\ldots,j_p}(x)}$ where
                   $M^+_{j_1,\ldots,j_p}(x) = \frac{M^+(x)}{m_{j_1}(x) \cdots m_{j_p}(x)}$.
```

If $\deg r_{j_1,\ldots,j_p}(x) < Q$, then the p processors $\{j_1, \ldots, j_p\}$ have failed, and $c(x) = r_{j_1,\ldots,j_p}(x)$ is the correct convolution. STOP.

4) If all the polynomials $r_{j_1,\ldots,j_\beta}(x)$ have degree $\geq Q$, then $\beta + 1$ to α faults occurred, and this cannot be corrected. STOP.

This algorithm essentially does an exhaustive search of all possible combinations of failed processors. It begins by checking if no processors failed by testing if $\deg z(x) < Q$. If so, then $c(x) = z(x)$ is the correct result. Otherwise, it begins with $p = 1$ and checks if the error was caused by a single processor failure. If not, then all possible two processor failures are checked ($p = 2$). This continues until $p = \beta$. If no set of residues which explains the fault can be found, then by Theorem 2 we know that between $\beta + 1$ and α faults occurred, and this cannot be corrected. Testing for multiple failures in this manner guarantees that only faulty processors will be corrected.

The algorithm can be implemented quickly if we recognize that only the high order coefficients of the remainder (those of degree $\geq Q$) must be computed before testing if $\deg r_{j_1,\ldots,j_p}(x) < Q$. If any of these coefficients are nonzero, then we abort the division and test the next set of processors. Once we find $\{j_1, \ldots, j_p\}$ such that the high order coefficients are all zero, we know that processors $\{j_1, \ldots, j_p\}$ have failed. To obtain the correct result, we complete the division. This procedure requires roughly $(Q^+ - Q)/Q$ as much computation as performing all divisions completely.

Even with this fast fault correction algorithm, checking for multiple processor failures can be computationally expensive. The procedure in Theorem 3 essentially does an exhaustive search of all possible combinations of failed processors. Checking for β or fewer failures requires a total of

$$\sum_{i=1}^{\beta} \frac{(N+C)!}{i!(N+C-i)!} \tag{6.25}$$

separate polynomial divisions. This is reasonable only for small values of β.

If exact arithmetic is used, then the above procedure is sufficient. However, if any rounding or truncation occurs during computation, the high order coefficients of $r_{j_1,\ldots,j_p}(x)$ will never be exactly zero, and our fault test needs to be modified. This is done in Section 6.5.

The error detection and correction techniques that are described in this section are similar to those used in integer RNS [52]. However, we perform arithmetic in polynomial rings, rather than in integer rings. Encoding entire sequences with a polynomial RNS has several advantages over low-level single sample encoding using an integer RNS. First, off-the-shelf fixed or floating point arithmetic units may be used since computation is performed in the complex field. Integer residue number systems, on the other hand, require nonstandard finite ring arithmetic units. They have great difficulty with rounding operations, and have limited dynamic range. Second, and most importantly, in polynomial rings the choice of moduli constrains the *length* of convolution that can be performed, but not the *dynamic range* of the sample values. Since the length is specified in advance, overflow can be avoided.

6.4 Fast FFT-Based Algorithm

In this section we present a specific set of moduli which allows each step of the algorithm to be implemented efficiently. We show that when mapped onto a 2-D array, our algorithm is computationally equivalent to computing convolution using a Cooley-Tukey FFT with two additional rows.

6.4.1 FFT Moduli

Computation is reduced if we use sparse polynomials as moduli. This simplifies computing the residues and performing the residue multiplications. Also, $M^{+}_{j_1,\ldots,j_p}(x)$ will be sparse, simplifying error detection and correction.

Suppose the samples to be convolved are elements of the field of complex numbers. Also assume that $a[n]$ and $b[n]$ have lengths such that the maximum length of the convolution is a composite $Q = NR$ for integers N and R. Then a particularly elegant choice for the moduli is as follows:

$$m_k(x) = x^R - W_{N+C}^{-(k-1)} \qquad \text{for} \qquad k = 1, \ldots, N+C \tag{6.26}$$

where $W_{N+C} = e^{j\frac{2\pi}{N+C}}$ is the $(N+C)^{\underline{th}}$ root of unity. Note that $m_k(x)$ can be written as the product of R first order factors,

$$m_k(x) = \prod_{i=0}^{R-1} \left(x - W_S^{-(k-1)-i(N+C)}\right) \tag{6.27}$$

where $S = (N+C)R$. From this expansion it can be seen that the $m_k(x)$ have no roots in common, and are thus co-prime. Also note that

$$M^+(x) = \prod_{k=1}^{N+C} m_k(x) = x^S - 1. \tag{6.28}$$

Our method computes $c(x)$ modulo $M^+(x)$ which corresponds to S-point circular convolution. Hence, to achieve fault-tolerance, we have embedded a $Q = NR$ point linear convolution in an $S = (N+C)R$ point circular convolution.

We will assume that at most a single processor can fail at any one time, and add redundancy such that this failure can be reliably detected and corrected. We thus require $\beta \geq 1$ and $\alpha \geq \beta$. To minimize redundancy, we choose $\alpha = \beta = 1$ and therefore $D = 2$. Since our moduli are all of the same degree, we need two extra moduli to achieve this level of redundancy. *We will assume that $C = 2$ throughout the rest of this section.*

6.4.2 Algorithm Description

We now describe in detail the steps involved in a fault-tolerant convolution algorithm which uses the moduli shown in (6.26). We describe the steps in terms of polynomial operations and as operations on two dimensional arrays. The latter description reveals the relationship between our fault-tolerant algorithm and standard convolution algorithms. This relationship will be discussed in Section 6.4.3

We map the sequence $a[n]$ onto a 2-D array as follows:

$$a[n_1, n_2] = a[n_1 R + n_2] \qquad \text{for} \qquad \begin{array}{l} 0 \leq n_1 \leq N+1 \\ 0 \leq n_2 \leq R-1 \end{array} \tag{6.29}$$

where n_1 is the row index and n_2 is the column index. Note that since $a[n]$ does not occupy the entire array, we zero pad it to length S. We compute the residues $a_k[n]$ via operations on $a[n_1, n_2]$ and place the residues in the 2-D array,

$$A[k,n] = a_k[n] \qquad \text{for} \qquad \begin{array}{l} 1 \le k \le N+2 \\ 0 \le n \le R-1. \end{array} \tag{6.30}$$

The arrays $b[n_1, n_2]$, $c[n_1, n_2]$, $z[n_1, n_2]$ and $B[k,n]$, $C[k,n]$, $Z[k,n]$ are similarly defined. We also use the following array representation of the remainder $r_j(x)$ which is used during error detection and correction:

$$r_j[n_1, n_2] = r_j[n_1 R + n_2] \qquad \text{for} \qquad \begin{array}{l} 0 \le n_1 \le N \\ 0 \le n_2 \le R-1 \\ 1 \le j \le N+2. \end{array} \tag{6.31}$$

Note that $r_j[n_1, n_2]$ has $N+1$ rows while the other arrays have $N+2$ rows.

Computation of Residues

The first step of the algorithm is to compute the residues $a_k(x)$. Since the moduli $m_k(x)$ are sparse polynomials, each coefficient of a residue polynomial is the sum of only $N+2$ terms,

$$a_k(x) = \sum_{n=0}^{R-1} x^n \sum_{l=0}^{N+1} W_{N+2}^{-(k-1)l} a[n+lR] \qquad \text{for} \qquad 1 \le k \le N+2 \tag{6.32}$$

or in array notation,

$$A[k,n] = \sum_{l=0}^{N+1} W_{N+2}^{-(k-1)l} a[l,n] \qquad \text{for} \qquad 1 \le k \le N+2. \tag{6.33}$$

For each n, this equation is recognized as an $N+2$ point DFT along the column $a[\cdot, n]$. Similar operations are used to compute the residue array $B[k,n]$.

Residue Multiplications

The second step of the algorithm is to compute the products $z_k(x) = \langle a_k(x) b_k(x) \rangle_{m_k(x)}$ for $k = 1, 2, \ldots, N+2$. In general, it is difficult to efficiently compute convolution modulo an arbitrary polynomial. However, the special moduli (6.26) allow the products to be computed by circular convolutions through a simple transformation as follows. First pre-multiply $a_k[n]$ and $b_k[n]$ by a phase shift,

$$\begin{aligned} \tilde{a}_k[n] &= a_k[n]W_S^{-(k-1)n} \iff \tilde{A}[k,n] = A[k,n]W_S^{-(k-1)n} \\ \tilde{b}_k[n] &= b_k[n]W_S^{-(k-1)n} \iff \tilde{B}[k,n] = B[k,n]W_S^{-(k-1)n}. \end{aligned}$$

Then convolve these new residues using an R-point circular convolution,

$$\tilde{z}_k(x) = \left\langle \tilde{a}_k(x)\tilde{b}_k(x) \right\rangle_{(x^R-1)} \iff \tilde{Z}[k,n] = \sum_{i=0}^{R-1} \tilde{A}[k,i]\tilde{B}[k,\langle n-i\rangle_R]. \tag{6.34}$$

Finally post-multiply to obtain the desired product residues,

$$z_k[n] = \tilde{z}_k[n]W_S^{(k-1)n} \iff Z[k,n] = \tilde{Z}[k,n]W_S^{(k-1)n}.$$

Many efficient circular convolution algorithms exist [53] and any one could be used to compute (6.34). A logical choice would be to use an FFT-based algorithm by taking R-point FFTs of each row, multiplying point-by-point, and then taking inverse R-point FFTs,

$$\tilde{Z}[k,\cdot] = \mathrm{FFT}_R^{-1}\left[\mathrm{FFT}_R\left[\tilde{A}[k,\cdot]\right]\cdot \mathrm{FFT}_R\left[\tilde{B}[k,\cdot]\right]\right]. \tag{6.35}$$

CRT Reconstruction

The next step in our algorithm is to reconstruct the polynomial $z(x)$ using all $N+2$ residues. By assumption, this operation is computed reliably. It can be shown that for moduli (6.26), each polynomial $M_k^+(x)$ is sparse, with only one out of every R coefficients being non-zero,

$$M_k^+(x) = \frac{M^+(x)}{m_k(x)} = \sum_{i=0}^{N+1} W_{N+2}^{(k-1)(i+1)}x^{iR}. \tag{6.36}$$

Also, the $D_k^+(x)$ are constants,

$$D_k^+(x) = \frac{1}{N+2}W_{N+2}^{-(k-1)}. \tag{6.37}$$

Since $\deg z_k(x) < R$, the CRT reconstruction is formed from non-overlapped, shifted, and scaled combinations of the $z_k(x)$. We find that

$$z(x) = \frac{1}{N+2} \sum_{k=1}^{N+2} \sum_{i=0}^{N+1} z_k(x) x^{iR} W_{N+2}^{i(k-1)} \Longleftrightarrow z[n_1, n_2] = \frac{1}{N+2} \sum_{k=1}^{N+2} Z[k, n_2] W_{N+2}^{(k-1)n_1}. \tag{6.38}$$

This is similar to (6.33) and is recognized as $N+2$ point inverse DFTs along each column $Z[\cdot, n_2]$.

Fast Fault Detection and Correction

If $\deg z(x) < Q$ then no fault has occurred, and we are done, $c(x) = z(x)$. This corresponds to the last 2 rows of $z[n_1, n_2]$ being zero. If there are nonzero entries in the last two rows, then a fault $\phi_q(x)$ has occurred in some processor q. To locate the fault we must divide $z(x)$ by each $M_j^+(x)$ in turn, and check the leading R coefficients of the remainder $r_j(x) = \langle z(x) \rangle_{M_j^+(x)}$.

For this special choice of moduli, there is an even faster fault detection technique. It can be shown that an error $\phi_q[n_2]$ linearly perturbs each row of the output,

$$z[n_1, n_2] = \begin{cases} c[n_1, n_2] + \dfrac{W_{N+2}^{(q-1)n_1}}{N+2} \phi_q[n_2] & \text{for} \quad n_1 = 0, \ldots, N-1 \\ \\ \dfrac{W_{N+2}^{(q-1)n_1}}{N+2} \phi_q[n_2] & \text{for} \quad n_1 = N, N+1. \end{cases} \tag{6.39}$$

This reveals an alternative, simpler method of locating the fault. Instead of computing and testing $N+C$ residues, we may use the correlation between non-zero samples of $z[N, n_2]$ and $z[N+1, n_2]$ to quickly identify q,

$$\hat{q} = \left\langle \text{round} \left[\frac{N+2}{2\pi} \text{ARG} \left(z[N+1, n_2] \, z^*[N, n_2] \right) \right] \right\rangle_{N+2} + 1 \tag{6.40}$$

where ARG(x) refers to the principal value, or angle, of the complex quantity x. Note, this approach is sensitive to computational noise which may corrupt the correlation and lead to incorrect fault diagnoses. This problem becomes more severe as $|\phi_q[n_2]|$ decreases since small perturbations greatly affect $\text{ARG}\left(z\left[N+1, n_2\right] z^*\left[N, n_2\right]\right)$. A more intelligent approach to estimating $\hat{q}$ is to use all the samples in rows $z[N, \cdot]$ and $z[N+1, \cdot]$ rather than just 2 samples. This approach is pursued in detail in Section 6.5.

6.4.3 Algorithm Summary

The computational steps involved in the FFT-based algorithm are summarized in Figure 6-2. Close examination reveals that this procedure is similar to computing convolution using Cooley-Tukey FFTs [53] of length $(N+2)R$. If Cooley-Tukey FFTs were used, we would first arrange the data into rows of 2-D matrices. Then compute the FFT of each column, multiply the array by twiddle factors, take an FFT of each row, and multiply these row FFTs. Then inverse FFT each row, multiply by twiddle factors, and inverse FFT each column. The only difference between a Cooley-Tukey FFT and our algorithm is that the initial column FFTs must be replaced with DFTs which compute each sample independently. This is necessary because the computation in each row must be done independently by a separate processor, and column FFTs would violate this partitioning of computation. Thus each processor must be loaded with the input sequences $a[n]$ and $b[n]$, and will evaluate only the single sample of each column DFT that it needs. FFTs are not efficient under these circumstances. In the CRT, however, the distribution of computation is not critical to the functioning of the algorithm since the CRT is computed reliably using TMR. Thus the $N+2$ residue processors output all their data to the CRT processors, which then use column FFTs to compute $z(x)$.

Added insight can be gained by studying two extreme cases. First, when $N = 1$, operation is analogous to TMR. We compute an R-point convolution using three independent R-point convolutions. Another interesting case occurs when $R = 1$. Then $m_k(x) = \left(x - W_{N+2}^{-(k-1)}\right)$ and $a_k(x)$ and $b_k(x)$ are constants equal to the value of the Fourier transforms of $a[n]$ and $b[n]$ at frequency $\frac{2\pi(k-1)}{N+2}$. This is equivalent to computing convolution by having each processor multiply a single DFT component. Adding two samples to the length of the convolution allows an error in any DFT component to be detected and corrected. This is similar to a method proposed by Wolf to protect communication channels from impulse noise [46]. He encodes sequences using DFTs and adds extra DFT coefficients

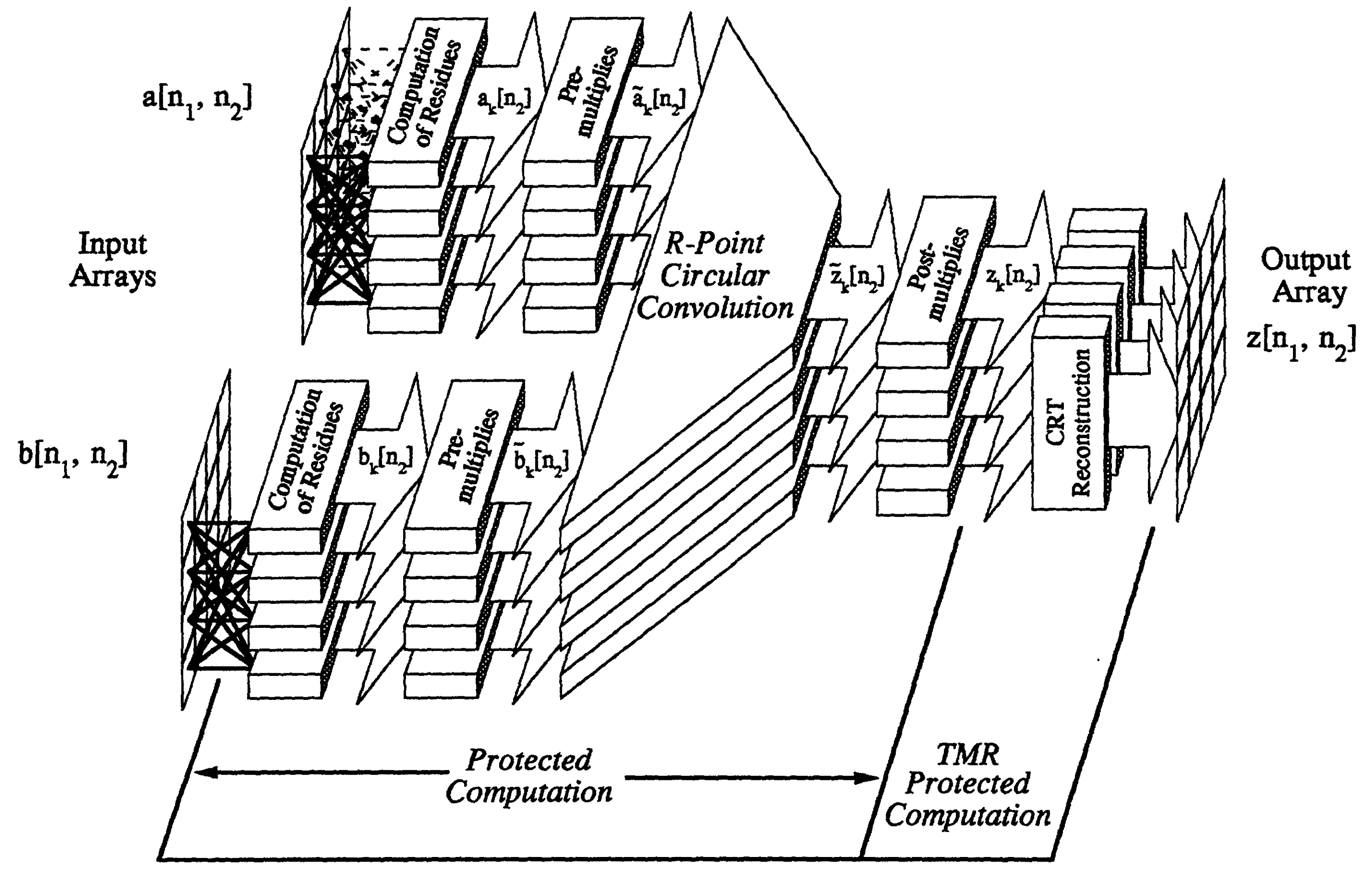


Figure 6-2: Computational steps involved in the fault-tolerant FFT-based algorithm. The example shown is of an 8-point convolution computed by four 4-point convolutions.

for redundancy.

6.5 Generalized Likelihood Ratio Test

If infinite precision arithmetic is used with no rounding, then all computation will be exact and the fault detection/correction procedure discussed in Section 6.4.2 would be sufficient. Unfortunately, processors must use fixed or floating point approximations to the complex numbers, and rounding and truncation errors occur. In this section we discuss how to distinguish between these small deviations and actual processor failures in an optimal manner.

We begin with (6.39) and add an extra term $\epsilon[n_1, n_2]$ to model the net effect of computational noise on each sample of the result,

$$z[n_1,n_2] = \begin{cases} c[n_1,n_2] + \frac{W_{N+2}^{(q-1)n_1}}{N+2}\phi_q[n_2] + \epsilon[n_1,n_2] & \text{for} \quad n_1 = 0,\ldots,N-1 \\ \\ \frac{W_{N+2}^{(q-1)n_1}}{N+2}\phi_q[n_2] + \epsilon[n_1,n_2] & \text{for} \quad n_1 = N, N+1. \end{cases} \tag{6.41}$$

Let $\boldsymbol{z}$, $\boldsymbol{c}$, and $\underline{\phi}_q$ represent the arrays $z[n_1,n_2]$, $c[n_1,n_2]$, and $\phi_q[n_2]$ respectively. We choose a set of hypotheses to model the behavior of our system. Let H_* represent the hypothesis that all processors are functioning properly, and let H_q represent the hypothesis of a failure in the $q^{\underline{th}}$ processor, where $q = 1,\ldots,N+2$. Also define P_* and P_q as the *a priori* probabilities of these events and assume that they are independent of data and fault.

Our basic approach is to compute the likelihood (probability) of observing the output $\boldsymbol{z}$ assuming that each hypothesis is true. The hypothesis with largest likelihood is most probable, and serves as our fault diagnosis. Unfortunately, the likelihoods depend on the correct output $\boldsymbol{c}$ and on the fault $\underline{\phi}_q$, if any, which are unknown. We therefore use a GLRT to jointly estimate the likelihoods and unknown parameters.

Define L_* to be the log likelihood of $\boldsymbol{z}$ and H_* conditioned on $\boldsymbol{c}$, maximized over all possible correct outputs $\boldsymbol{c}$,

$$L_* = \max_{\boldsymbol{c}} \log p(\boldsymbol{z}, H_* \mid \boldsymbol{c}) = \max_{\boldsymbol{c}} \log p(\boldsymbol{z} \mid H_*, \boldsymbol{c}) + \log P_*. \tag{6.42}$$

We employ log likelihoods rather than probabilities since they can be solved more easily. Both methods are equivalent since the logarithm is a monotonically increasing function. Similarly, define L_q as the log likelihood of $\boldsymbol{z}$ and H_q conditioned on $\boldsymbol{c}$ and $\underline{\phi}_q$, with $\boldsymbol{c}$ and $\underline{\phi}_q$ set to their most likely values,

$$L_q = \max_{\boldsymbol{C},\underline{\phi}_q} \log p\left(\boldsymbol{z}, H_q \mid \boldsymbol{c}, \underline{\phi}_q\right) = \max_{\boldsymbol{C},\underline{\phi}_q} \log p\left(\boldsymbol{z} \mid H_q, \boldsymbol{c}, \underline{\phi}_q\right) + \log P_q. \tag{6.43}$$

We will compute L_* and L_q for $q = 1, \ldots, N+C$ and pick the largest. The largest likelihood corresponds to the most likely failure hypothesis given the observed data $\boldsymbol{z}$. Also, the values $\hat{\boldsymbol{c}}$ and $\hat{\underline{\phi}}_{\hat{q}}$ which maximize the likelihoods serve as estimates of the correct output and error.

The solution of the likelihood equations depends heavily on the probability distribution of the computational noise and cannot be solved in general. However, since $\epsilon[n_1, n_2]$ is the accumulated effect of many rounding operations, it is reasonable to model it as white, zero mean Gaussian noise with variance σ_ϵ^2. We will also assume that $\epsilon[n_1, n_2]$ is independent of signal and fault,

$$p\left(\epsilon[n_1, n_2] \mid \boldsymbol{c}, \underline{\phi}_q\right) = p(\epsilon[n_1, n_2]) = N\left(0, \sigma_\epsilon^2\right). \tag{6.44}$$

We assume that the processors are uniform and have equal probability of failure. The GLRT is solved in Appendix 6.B and it reduces to the following simple form. First compute the constant

$$L'_* = 4\sigma_\epsilon^2 \log\left(\frac{P_*}{P_q}\right). \tag{6.45}$$

This serves as a threshold which decides between H_* and all other hypotheses. Next, estimate the most likely failed processor given the observed data,

$$\hat{q} = \left\langle \text{round}\left[\frac{N+2}{2\pi} \text{ARG}\left(\rho_{N+1,N}\right)\right]\right\rangle_{N+2} + 1 \tag{6.46}$$

where $\rho_{i,j}$ is the total correlation between the $i^{\underline{th}}$ and $j^{\underline{th}}$ rows of $z[n_1, n_2]$,

$$\rho_{i,j} = \sum_{n_2=0}^{R-1} z[i, n_2]\, z^*[j, n_2]. \tag{6.47}$$

Then compute the log likelihood of a failure in processor $\hat{q}$,

$$L'_{\hat{q}} = \rho_{N,N} + \rho_{N+1,N+1} + 2\mathrm{Re}\left(\rho_{N,N+1} W_{N+2}^{(\hat{q}-1)}\right). \tag{6.48}$$

Compare $L'_{\hat{q}}$ with the threshold L'_*. If $L'_{\hat{q}}$ is smaller, we declare that no processor has failed and ascribe the deviation to computational noise. The estimate of the output, $\hat{c}[n_1, n_2]$, then equals

$$\hat{c}[n_1, n_2] = \begin{cases} z[n_1, n_2] & \text{for } n_1 = 0, \ldots, N-1 \\ 0 & \text{for } n_1 = N, N+1. \end{cases} \tag{6.49}$$

Otherwise, if $L'_{\hat{q}}$ is greater than L'_*, we declare that processor $\hat{q}$ has failed. We correct the output by first estimating the error

$$\hat{\phi}_{\hat{q}}[n_2] = \frac{N+2}{2}\left[z[N, n_2] W_{N+2}^{-(\hat{q}-1)N} + z[N+1, n_2] W_{N+2}^{-(\hat{q}-1)(N+1)}\right] \tag{6.50}$$

and then subtracting a phase shifted copy of this estimate from $z[n_1, n_2]$,

$$\hat{c}[n_1, n_2] = \begin{cases} z[n_1, n_2] - \dfrac{W_{N+2}^{(q-1)n_1}}{N+2}\hat{\phi}_{\hat{q}}[n_2] & \text{for } n_1 = 0, \ldots, N-1 \\ 0 & \text{for } n_1 = N, N+1. \end{cases} \tag{6.51}$$

This method improves performance over the simple method (6.40) because a total of $2R$ samples are used to identify the faulty processor instead of only 2 samples. It yields a more accurate estimate of $\phi_q[n_2]$ by reducing computational noise through averaging. The GLRT requires little additional computation; roughly $3R$ multiplications are needed to compute the correlations and roughly $3R + NR$ to correct the fault.

With a GLRT, error detection depends on the value of random computational noise, and therefore errors may not always be reliably detected. High noise levels can cause false alarms and small errors can go undetected. In practice, however, this is not a serious problem since all large errors are properly detected, and during a false alarm, the error estimate $\hat{\phi}_{\hat{q}}[n_2]$ is generally quite small, and the improperly applied error correction does not corrupt the output significantly.

The numerical value of the decision threshold L'_* depends on the statistics of the computational noise which we modeled as Gaussian random variables. In practice, the distribution of the computational noise depends heavily on details of the implementation (arithmetic precision, values of N and R, FFT routines used) and a Gaussian model may be inappropriate. Also note, if floating point arithmetic is used, then the actual computational noise may be correlated with the data, and it may be necessary to scale the threshold according to the magnitude of the input sequences [54]. In practice, it may be best to use computer simulations to choose a threshold that achieves the desired false alarm probability.

When a faulty processor is detected, the fault is corrected and then several alternative courses of action may be taken. The processor may be monitored to see if the error disappears. If so, then the fault was only transient and normal operation can continue. If the error persists, the faulty processor can be shutdown and the system reconfigured as an $N+1$ processor error detecting system. Alternatively, a standby processor might be switched in to replace the faulty processor.

6.6 Fault-Tolerance Overhead

In this section we discuss the efficiency and fault coverage of the single error-correcting FFT-based system discussed in the previous section. We compare an unprotected NR-point convolution computed by a standard FFT-based algorithm, with a fault-tolerant NR-point convolution which is embedded in an $(N+2)R$-point circular convolution. Our analysis focuses on how the polynomial RNS protects the computation involved in convolution and does not take into account the additional hardware and processing required for reliable interprocessor communication and I/O. We examine two quantities: *overhead* and *coverage*. Overhead is defined as the percentage of extra computation needed for fault-tolerance relative to the unprotected algorithm. Coverage is the percentage of total computation protected by the polynomial RNS (the remaining computation is protected via TMR).

We divide the FFT-based algorithm into 4 steps. During step 1, we compute the residue arrays $A[k,n_2]$ and $B[k,n_2]$. Processor k computes row k of these arrays using DFTs as discussed in Section 6.4.3. In step 2, the residues are convolved using R-point FFTs. Step 3 is the CRT reconstruction using $N+2$ point column FFTs. During step 4, error detection and correction are performed. Note that steps 3 and 4 are protected via TMR, and we will weight the computation required for these steps accordingly.

For simplicity, when considering the computational complexity of an algorithm, we count only the number of multiplications involved. This is a reasonable approximation, since for most FFT algorithms, the total number of operations is proportional to the number of multiplications. Let M_1, M_2, M_3, and M_4 be the number of multiplications in each step. Also, let M_u be the total number of multiplications in a standard unprotected NR-point convolution computed with FFTs. Using these definitions, our performance measures may be written as:

$$\text{overhead} = \frac{M_1 + M_2 + 3\,(M_3 + M_4)}{M_u} - 1 \tag{6.52}$$

$$\text{coverage} = \frac{M_1 + M_2}{M_1 + M_2 + M_3 + M_4}. \tag{6.53}$$

As part of our calculations we must compare the number of multiplications in N and $N+2$ point FFTs. This is difficult to do for arbitrary values of N and so we make a rough approximation. We assume that an L-point FFT requires $2L\log_2 L$ multiplications [53], even though L may not be a power of 2. The majority of multiplications occur in the FFTs, and we ignore twiddle and transform coefficient multiplications. With these assumptions we obtain

$$M_1 \approx 2R(N+2)^2 \tag{6.54}$$

$$M_2 \approx 6R(N+2)\log_2 R \tag{6.55}$$

$$M_3 \approx 2R(N+2)\log_2(N+2) \tag{6.56}$$

$$M_4 \approx 6R + RN \tag{6.57}$$

$$M_u \approx 6RN\log_2(RN). \tag{6.58}$$

Using these approximations, we evaluated our measures for several values of N and R and the results are shown in Table 6.1. A good reference to compare the calculated overheads with is that required by a MR system offering a similar level of fault protection. A single error-correcting MR system requires triplication, that is, 200% overhead. Our method,

Overhead		R		
		64	256	1024
$N+2$	4	195%	174%	161%
	8	93%	82%	74%
	16	86%	74%	65%
	32	117%	100%	88%
	64	192%	165%	145%

Coverage		R		
		64	256	1024
$N+2$	4	88%	90%	92%
	8	87%	90%	91%
	16	88%	90%	91%
	32	90%	91%	92%
	64	93%	93%	94%

Table 6.1: Overhead and fault coverage of the robust FFT-based convolution algorithm.

on the other hand, achieves this level of fault tolerance with as little as 65% overhead, a substantial savings. We find that overhead varies strongly with N and reaches a minimum at $N+2=16$. Two separate factors contribute to this behavior. First, if N is small, the two additional row convolutions make up a sizable portion of the total computation, and thus the overhead is high. Second, when N is large, the DFTs in step 1 require significant amounts of computation since the number of operations involved grows as $(N+2)^2$. Efficient operation occurs between these two extremes.

We also find that the polynomial RNS protects the majority of computation, as demonstrated by the high coverage values. In most instances, over 90% of the computation occurs in the residue channels. Thus, the bulk of computation is covered by the low cost arithmetic code, while only the remaining 10% need be protected by more expensive MR.

6.7 Conclusion

In this chapter we presented a new approach to protecting linear convolution which was considerably cheaper than traditional methods based on modular redundancy. Our algorithm used a polynomial residue number system (RNS), which is the underlying structure of the Winograd Convolution Algorithm. Computation is decomposed into independent, parallel residue channels, and redundancy incorporated by adding extra residue channels. Analogous to integer RNS fault-tolerance schemes, single errors can be detected by adding one extra modulus, and corrected using two extra moduli. However, we do not encounter many of the problems associated with integer RNS.

We derived conditions on the redundant moduli such that a desired level of fault-tolerance may be achieved, and presented an algorithm for detecting and correcting multiple processor failures. Importantly, we presented a specific set of moduli polynomials which yielded an efficient FFT-based algorithm. The effects of computational noise were handled using a generalized likelihood ratio test, and the resulting fault detection/correction algorithm is both fast and accurate.

The parallel nature of our algorithm makes it ideal for implementation on a multiprocessor system. We distribute computation such that each residue channel is computed by a separate processor. The low cost polynomial RNS coding scheme is able to protect the bulk of computation, roughly 90%, while the remaining 10% is protected via more expensive triple modular redundancy. This hybrid approach yields a system fully protected against any single failure at a cost significantly lower than a fully redundant implementation. We are able to achieve this level of reliability with only 65% overhead, compared with 200% needed for triple modular redundancy.

Variations of our scheme are possible which allow a wider range of operations to be protected. Addition and subtraction of polynomials can also be protected, and we are not limited to protecting individual ring operations, but can protect several with a single residue encoding and a single error test. The only requirement is that the length of the final result be constrained. Errors may be detected and corrected in the same manner.

6.A Proof of Theorems

This appendix contains proofs of the theorems which were presented in Section 6.3.3. We assume throughout that λ failures occur in processors $\{k_1, \ldots, k_\lambda\}$ and that the outputs of the residue processors have value (6.15). The result of the reliable CRT reconstruction will then be (6.19). We also assume that the moduli satisfy (6.21) for some value of D, and that $D = \alpha + \beta$ for $\alpha \geq \beta \geq 0$.

If no errors have occurred, $\lambda = 0$, then $z(x) = c(x)$ and thus $\deg z(x) < Q$. This proves Theorem 1a. To prove 1b, we use the following lemma:

<u>Lemma 1:</u> Let $\phi_{k_1}(x) \neq 0, \ldots, \phi_{k_\lambda}(x) \neq 0$ be any set of $1 \leq \lambda \leq D$ polynomials

with $\deg \phi_{k_i}(x) < \deg m_{k_i}(x)$. Then:

$$\deg \left[\sum_{i=1}^{\lambda} \left\langle \phi_{k_i}(x) D_{k_i}^{+}(x) \right\rangle_{m_{k_i}(x)} M_{k_i}^{+}(x) \right] \geq Q. \tag{6.59}$$

Proof of Lemma 1: Let $\tilde{\phi}_{k_i}(x) = \left\langle \phi_{k_i}(x) D_{k_i}^{+}(x) \right\rangle_{m_{k_i}(x)}$. Since $\deg \phi_{k_i}(x) < \deg m_{k_i}(x)$ and $\phi_{k_i}(x) \neq 0$, and since $D_{k_i}^{+}(x)$ and $m_{k_i}(x)$ are co-prime, we know that $\tilde{\phi}_{k_i}(x) \neq 0$. Then:

$$\begin{aligned} \sum_{i=1}^{\lambda} \tilde{\phi}_{k_i}(x) M_{k_i}^{+}(x) = \left[\frac{M^{+}(x)}{m_{k_1}(x) \cdots m_{k_\lambda}(x)} \right] & \left[\tilde{\phi}_{k_1}(x) m_{k_2}(x) \cdots m_{k_\lambda}(x) + \right. \\ & \left. \tilde{\phi}_{k_2}(x) m_{k_1}(x) m_{k_3}(x) \cdots m_{k_\lambda}(x) + \cdots + \tilde{\phi}_{k_\lambda}(x) m_{k_1}(x) \cdots m_{k_{\lambda-1}}(x) \right]. \end{aligned} \tag{6.60}$$

The first term on the right hand side has degree $\geq Q$ by (6.21). The second term cannot be zero because it cannot be evenly divided by any of the polynomials $m_{k_1}(x), \ldots, m_{k_\lambda}(x)$. Thus the degree of the right hand side is $\geq Q$ and Lemma 1 is true.

Since Lemma 1 is true, we know that the error term in (6.19) will always have degree $\geq Q$. Since $\deg c(x) < Q$, $z(x)$ must have degree $\geq Q$ if between 1 and D failures occur. This proves Theorem 1.

Theorem 2 is contained in Theorem 3, and thus proving Theorem 3 is sufficient. We rely upon the following lemma:

Lemma 2: Assume that $\lambda \leq \alpha$ errors occur in processors $\{k_1, \ldots, k_\lambda\}$. Let $\{j_1, \ldots, j_p\}$ be any set of p processors with $p \leq \beta$. Then:

$$\deg \left[\langle z(x) \rangle_{M_{j_1,\ldots,j_p}^{+}(x)} \right] < Q \text{ if and only if } \{k_1, \ldots, k_\lambda\} \subset \{j_1, \ldots, j_p\}$$

where

$$M_{j_1,\ldots,j_p}^{+}(x) = \frac{M^{+}(x)}{m_{j_1}(x) \cdots m_{j_p}(x)}.$$

Proof of Lemma 2: Let $r_{j_1,\ldots,j_p}(x) = \langle z(x) \rangle_{M_{j_1,\ldots,j_p}^{+}(x)}$. We prove this lemma in two parts. First, we show that $\deg r_{j_1,\ldots,j_p}(x) \geq Q$ if $\{k_1, \ldots, k_\lambda\} \not\subset \{j_1, \ldots, j_p\}$. Then, we show that $\deg r_{j_1,\ldots,j_p}(x) < Q$ if $\{k_1, \ldots, k_\lambda\} \subset \{j_1, \ldots, j_p\}$.

Using (6.60), write the result of the reliable CRT reconstruction in the form

$$z(x) = c(x) + \frac{M^+(x)}{m_{k_1}(x)\cdots m_{k_\lambda}(x)}\Phi(x) \tag{6.61}$$

where $\Phi(x) \neq 0$ and $\Phi(x)$ is co-prime to the moduli $m_{k_1}(x), \ldots, m_{k_\lambda}(x)$. Then, using the division algorithm for polynomials, write this as

$$z(x) = r_{j_1,\ldots,j_p}(x) + \frac{M^+(x)}{m_{j_1}(x)\cdots m_{j_p}(x)}Q(x) \tag{6.62}$$

where $Q(x)$ and $r_{j_1,\ldots,j_p}(x)$ are the quotient and remainder when $z(x)$ is divided by $M^+_{j_1,\ldots,j_p}(x)$. Suppose $\{j_1, \ldots, j_p\} \cap \{k_1, \ldots, k_\lambda\} = \{m_1, \ldots, m_r\}$ where $r \leq \min(\lambda, p)$, and suppose $\{j_1, \ldots, j_p\} \cup \{k_1, \ldots, k_\lambda\} = \{n_1, \ldots, n_s\}$ where $s \leq \lambda + p$. Let $\{\tilde{j}_1, \ldots, \tilde{j}_{p-r}\} = \{j_1, \ldots, j_p\} - \{m_1, \ldots, m_r\}$ represent the indices of $\{j_1, \ldots, j_p\}$ which do not appear in $\{k_1, \ldots, k_\lambda\}$. Similarly let $\left\{\tilde{k}_1, \ldots, \tilde{k}_{\lambda-r}\right\} = \{k_1, \ldots, k_\lambda\} - \{m_1, \ldots, m_r\}$. Equating (6.61) and (6.62) gives:

$$\begin{aligned} c(x) - r_{j_1,\ldots,j_p}(x) = & \left[\frac{M^+(x)}{m_{n_1}(x)\cdots m_{n_s}(x)}\right]\left[Q(x)m_{\tilde{k}_1}(x)\cdots m_{\tilde{k}_{\lambda-r}}(x) - \right. \\ & \left. \Phi(x)m_{\tilde{j}_1}(x)\cdots m_{\tilde{j}_{p-r}}(x)\right]. \end{aligned} \tag{6.63}$$

Since $\lambda \leq \alpha$ and $p \leq \beta$, we know that $s \leq D$. Therefore by (6.21), the first term on the right hand side has degree $\geq Q$. When $\{k_1, \ldots, k_\lambda\} \not\subset \{j_1, \ldots, j_p\}$, then $\left\{\tilde{k}_1, \ldots, \tilde{k}_{\lambda-r}\right\}$ will be nonempty. Then, since $\Phi(x)$ is co-prime to the moduli $m_{k_1}(x), \ldots, m_{k_\lambda}(x)$, the second term on the right hand side cannot be zero. Thus the right hand side has degree $\geq Q$ and $\deg r_{j_1,\ldots,j_p}(x) \geq Q$ if $\{k_1, \ldots, k_\lambda\} \not\subset \{j_1, \ldots, j_p\}$.

Now assume that $\{k_1, \ldots, k_\lambda\} \subset \{j_1, \ldots, j_p\}$. Begin with (6.61) and expand the second term,

$$z(x) = c(x) + \sum_{i=1}^{\lambda} \tilde{\phi}_{k_i}(x)M^+_{k_i}(x). \tag{6.64}$$

Now take the residue when this is divided by $M^+_{j_1,\ldots,j_p}(x)$,

$$r_{j_1,\ldots,j_p}(x) = \langle c(x)\rangle_{M^+_{j_1,\ldots,j_p}(x)} + \sum_{i=1}^{\lambda}\left\langle \tilde{\phi}_{k_i}(x)M^+_{k_i}(x)\right\rangle_{M^+_{j_1,\ldots,j_p}(x)}. \tag{6.65}$$

Since $\deg M^{+}_{j_1,\ldots,j_p}(x) > \deg c(x)$, $\langle c(x)\rangle_{M^{+}_{j_1,\ldots,j_p}(x)} = c(x)$. Also, since $M^{+}_{j_1,\ldots,j_p}(x)$ divides $M^{+}_{k_i}(x)$, $\left\langle \tilde{\phi}_{k_i}(x) M^{+}_{k_i}(x)\right\rangle_{M^{+}_{j_1,\ldots,j_p}(x)} = 0$ for $i = 1,\ldots,\lambda$. The above equation then reduces to

$$r_{j_1,\ldots,j_p}(x) = c(x). \tag{6.66}$$

Thus $r_{j_1,\ldots,j_p}(x) = c(x)$ and $\deg r_{j_1,\ldots,j_p}(x) < Q$ when $\{k_1,\ldots,k_\lambda\} \subset \{j_1,\ldots,j_p\}$. This proves Lemma 2.

To show that the procedure in Theorem 3 works properly, we consider two cases. First assume that no failures occurred, $\lambda = 0$. Then by Theorem 1, $z(x) = c(x)$ and $\deg z(x) < Q$, and the procedure would stop in step 1. Second assume that $\lambda \leq \alpha$ errors occurred. Lemma 2 guarantees that $\deg r_{j_1,\ldots,j_p}(x) \geq Q$ if $\{k_1,\ldots,k_\lambda\} \not\subset \{j_1,\ldots,j_p\}$. Since we are checking all possible combinations of p processors, starting with $p = 1$ and continuing until $p = \beta$, $\deg r_{j_1,\ldots,j_p}(x)$ will be less than Q if and only if $\lambda \leq \beta$, $p = \lambda$, and $\{j_1,\ldots,j_p\} = \{k_1,\ldots,k_\lambda\}$. Then by Lemma 2, $c(x) = r_{j_1,\ldots,j_p}(x)$ is the correct solution. Otherwise, if $\beta + 1 \leq \lambda \leq \alpha$ failures occurred, then $\{k_1,\ldots,k_\lambda\} \not\subset \{j_1,\ldots,j_p\}$ and we continue to step 4. This set of faults is uncorrectable.

6.B Solution of Likelihood Equations

In this appendix we solve the likelihood equations discussed in Section 6.5. We begin with (6.42) and (6.43) and model computational noise as a zero mean Gaussian random process (6.44). With this assumption, the likelihoods become

$$L_* = \max_{\boldsymbol{c}} \left[\eta_* - \frac{1}{2\sigma_\epsilon^2} \sum_{n_1=0}^{N+1} \sum_{n_2=0}^{R-1} |z[n_1,n_2] - c[n_1,n_2]|^2\right] \tag{6.67}$$

$$L_q = \max_{\boldsymbol{c},\underline{\phi}_q} \left[\eta_q - \frac{1}{2\sigma_\epsilon^2} \sum_{n_1=0}^{N+1} \sum_{n_2=0}^{R-1} \left| z[n_1,n_2] - c[n_1,n_2] - \frac{W_{N+2}^{(q-1)n_1}}{N+2}\phi_q[n_2]\right|^2\right] \tag{6.68}$$

where η_* and η_q are constants,

$$\eta_* = \log P_* - \frac{1}{2} S \log\left(2\pi\sigma_\epsilon^2\right) \tag{6.69}$$

$$\eta_q \quad = \quad \log P_q - \frac{1}{2} S \log\left(2\pi\sigma_\epsilon^2\right). \tag{6.70}$$

We start by maximizing L_* over $\boldsymbol{c}$ to obtain $\hat{c}[n_1, n_2]$, an estimate of the output given that H_* is true. Since $c[n_1, n_2]$ is zero for rows N and $N+1$, (6.67) can be rewritten as,

$$L_* = \max_{\boldsymbol{c}} \left[\eta_* - \frac{1}{2\sigma_\epsilon^2} \left\{ \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{R-1} |z[n_1, n_2] - c[n_1, n_2]|^2 + \sum_{n_1=N}^{N+1} \sum_{n_2=0}^{R-1} |z[n_1, n_2]|^2 \right\} \right]. \tag{6.71}$$

Maximizing over $\boldsymbol{c}$ we obtain

$$\hat{c}[n_1, n_2] = \begin{cases} z[n_1, n_2] & \text{for} \quad n_1 = 0, \ldots, N-1 \\ 0 & \text{for} \quad n_1 = N, N+1. \end{cases} \tag{6.72}$$

Substituting $\hat{c}[n_1, n_2]$ into (6.71) yields the likelihood that hypothesis H_* is true,

$$L_* = \eta_* - \frac{1}{2\sigma_\epsilon^2} \sum_{n_1=N}^{N+1} \sum_{n_2=0}^{R-1} |z[n_1, n_2]|^2. \tag{6.73}$$

Computing L_q is more difficult since we must estimate both $\boldsymbol{c}$ and $\underline{\phi}_q$. Maximizing (6.68) over $\boldsymbol{c}$ we obtain

$$\hat{c}_I[n_1, n_2] = \begin{cases} z[n_1, n_2] - \dfrac{W_{N+2}^{(q-1)n_1}}{N+2} \phi_q[n_2] & \text{for} \quad n_1 = 1, \ldots, N-1 \\ 0 & \text{for} \quad n_1 = N, N+1. \end{cases} \tag{6.74}$$

(Note, we call this $\hat{c}_I[n_1, n_2]$ to emphasize that it is an intermediate step in the maximization process, and not the final estimate of the correct output.) Substituting $\hat{c}_I[n_1, n_2]$ into (6.68), the log likelihood function becomes

$$L_q = \max_{\underline{\phi}_q} \left[\eta_q - \frac{1}{2\sigma_\epsilon^2} \sum_{n_1=N}^{N+1} \sum_{n_2=0}^{R-1} \left| z[n_1, n_2] - \frac{W_{N+2}^{(q-1)n_1}}{N+2} \phi_q[n_2] \right|^2 \right]. \tag{6.75}$$

Maximizing over $\underline{\phi}_q$ and keeping in mind that complex quantities are involved, we obtain

$$\hat{\phi}_q[n_2] = \frac{N+2}{2} \left[z[N, n_2] W_{N+2}^{-(q-1)N} + z[N+1, n_2] W_{N+2}^{-(q-1)(N+1)} \right]. \tag{6.76}$$

The error estimate $\hat{\phi}_q[n_2]$ is formed by averaging the last two rows of $z[n_1, n_2]$ with appropriate phase shifts and a scale factor of $N+2$. Substituting $\hat{\phi}_q[n_2]$ for $\phi_q[n_2]$ in (6.74) gives the estimate of the correct output,

$$\hat{c}[n_1, n_2] = \begin{cases} z[n_1, n_2] - \dfrac{W_{N+2}^{(q-1)n_1}}{N+2}\hat{\phi}_q[n_2] & \text{for} \quad n_1 = 1, \ldots, N-1 \\ 0 & \text{for} \quad n_1 = N, N+1. \end{cases} \tag{6.77}$$

Finally, substituting (6.77) and (6.76) into (6.68), and after some algebra, we obtain the likelihood of hypothesis H_q,

$$L_q = \eta_q - \frac{1}{2\sigma_\epsilon^2} \sum_{n_1=N}^{N+1} \sum_{n_2=0}^{R-1} |z[n_1, n_2]|^2 + \frac{1}{2\sigma_\epsilon^2} \frac{2}{(N+2)^2} \sum_{n_2=0}^{R-1} \left|\hat{\phi}_q[n_2]\right|^2. \tag{6.78}$$

The likelihood equations can be further simplified by assuming that the failure probabilities P_q are the same for $q = 1, \ldots, N+2$, and by using relative likelihoods defined by

$$L'_k = 4\sigma_\epsilon^2 \left[L_k - \eta + \frac{1}{2\sigma_\epsilon^2} \sum_{n_1=N}^{N+1} \sum_{n_2=0}^{R-1} |z[n_1, n_2]|^2 \right]. \tag{6.79}$$

The likelihoods then reduce to

$$L'_* = 4\sigma_\epsilon^2 \log \frac{P_*}{P_q} \tag{6.80}$$

and

$$L'_q = \rho_{N,N} + \rho_{N+1,N+1} + 2\mathrm{Re}\left(\rho_{N,N+1} W_{N+2}^{(q-1)}\right) \tag{6.81}$$

where $\rho_{i,j}$ is the total correlation between the $i^{\underline{th}}$ and $j^{\underline{th}}$ rows of $z[n_1, n_2]$ and is defined in (6.47).

We can simplify the hypothesis testing procedure by solving directly for the value of q which maximizes (6.81) rather than computing each L'_q. This yields

$$\hat{q} = \left\langle \text{round}\left[\frac{N+2}{2\pi} \text{ARG}\left(\rho_{N+1,N}\right)\right] \right\rangle_{N+2} + 1. \tag{6.82}$$

Our fault test then proceeds as follows. First, compute the constant L'_*. Then compute $\hat{q}$ and $L'_{\hat{q}}$ using (6.82) and (6.81). If $L'_* > L'_{\hat{q}}$, we declare that no processor has failed and ascribe the nonzero samples in rows $z\,[N, \cdot]$ and $z\,[N+1, \cdot]$ to computational noise. Otherwise, we declare that processor $\hat{q}$ has failed and correct the fault using (6.76) and (6.77).

Chapter 7

Fault-Tolerant A/D Conversion

This chapter examines an A/D converter system that provides a high sampling rate and that can tolerate converter failures. Such a system could be used in high stress environments where continuous operation is needed or in remote sensing applications where servicing faulty units is impractical or even impossible.

Unlike previous chapters, this chapter does not deal with computational fault-tolerance. Rather, redundancy is used to protect the conversion of continuous-time to discrete-time signals. Although a different application of fault-tolerance, we will show that the robust A/D converter system may be easily placed in the group-theoretic framework of Chapter 4.

We describe the basic round-robin architecture used by the A/D converter system in Section 7.1 and introduce linear redundancy through oversampling. Then, by examining the redundancy, we show in Section 7.2 how this system is related to the framework of Chapter 4. Section 7.3 develops the optimal error detection and correction algorithm utilizing a generalized likelihood ratio test. The algorithm reduces to a simple form, with a complexity comparable to that of an FIR filter. Section 7.4 considers an ideal unrealizable system, while Section 7.5 addresses problems encountered in a practical implementation. Both sections contain results from detailed computer simulations. We conclude in Section 7.6.

7.1 Round-Robin A/D Converter

A round-robin A/D converter system is shown in Figure 7-1. It contains N slow A/D

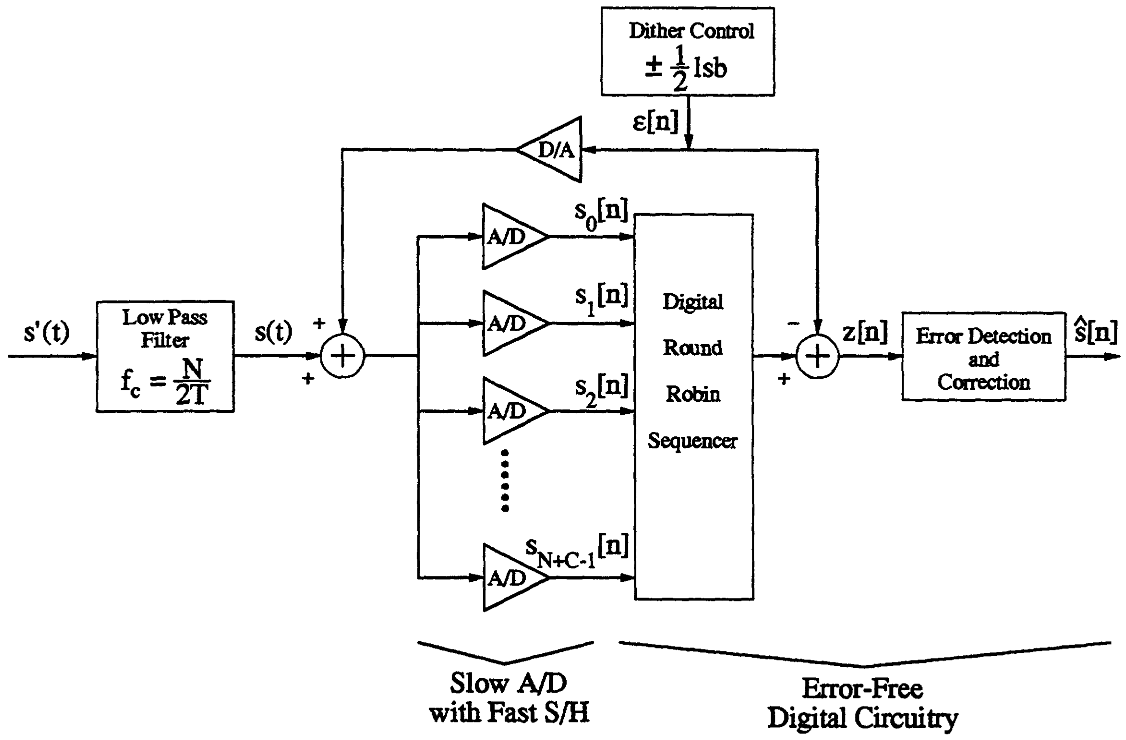


Figure 7-1: Round-robin A/D converter system.

converters each with fast sample and hold circuitry. The first converter samples and holds the analog input signal $s(t)$ and then begins a conversion. After a fixed delay, the second converter samples the signal and begins a conversion. This repeats for all N converters, and by the time the Nth converter starts, the first converter has finished and is ready to accept another sample. Operation continues in this circular fashion. If a conversion requires T seconds for a single converter, then the overall sampling rate for the round-robin system would be N/T samples/sec., and the input can contain frequencies up to $f_{max} = N/2T$ Hz.

To decorrelate the quantization noise from the signal, we use a small amount of dither. Dither circuitry adds a random analog voltage uniformly distributed between $\pm 1/2$lsb (least significant bit) to the sample. After conversion, it subtracts this same quantity from the digital signal. As a result of dither, each output sample contains white quantization noise uniformly distributed between $\pm 1/2$lsb, and which is uncorrelated with the signal.

We assume that the converters must operate in a stressed environment and that they are the only components subject to failure. We model converter failures as being independent and assume that when a converter fails, it corrupts only the samples assigned to it.

We assume that the dither circuitry, digital processing, and output buses always function properly. If necessary, these components could be protected against failure by modular redundancy or the digital processing may be performed remotely in a less stressful environment. Failures in the dither circuitry can be restricted to cause no more than a $\pm\frac{1}{2}$ lsb error in the samples.

This system is made robust by introducing redundant information. Keep the analog input signal bandlimited to $\pm f_{max}$ Hz, and add C extra converters to increase the sampling rate to $\frac{N+C}{T}$ samples/sec. The input signal is now somewhat oversampled.

7.2 Relationship to Framework

We now describe how the results of this chapter are related to the group-theoretic framework presented in Chapter 4. Let $s(t)$ and $z[n]$ denote the continuous-time input and discrete-time output of the A/D converter system. $s(t)$ is a real continuous-time signal that is bandlimited to $\pm f_{max}$ Hz. Let $\mathcal{G}$ denote the set of all continuous-time real signals that are bandlimited to $\pm f_{max}$ Hz. The set $\mathcal{G}$ together with the field $\mathbb{R}$ forms a vector space G. The group product in G is defined as the addition of functions and scalar multiplication corresponds to multiplying a function by a constant. Similarly, the set of all possible discrete-time sequences $z[n]$ forms a vector space H over the field of real numbers. Operations in H consist of the addition of sequences and scalar multiplication.

We can model the sampling process as a mapping from $s(t) \in G$ to $z[n] \in H$. We begin by ignoring quantization noise and assume that the converters have infinite precision. (Quantization noise does not play a role at this point, it only becomes important when we consider the optimal solution based on a generalized likelihood ratio test.) If no faults occur within the system, then the output may be written as

$$z[n] = s(nT). \tag{7.1}$$

It is easily shown that the sampling process satisfies the requirements of a vector space homomorphism (4.56). Since we are oversampling, $z[n]$ contains redundant information. The subset of valid error-free sequences in H corresponds to the subspace $\tilde{G}$ of low-pass discrete-time signals that are bandlimited to $\pm\frac{\pi N}{N+C}$ radians.

Now assume that λ failures occur in converters $k_1, k_2, \ldots, k_\lambda$. The output will then equal

$$z[n] = s(nT) + \phi_{k_1}[n] + \phi_{k_2}[n] + \cdots + \phi_{k_\lambda}[n] \tag{7.2}$$

where $\phi_k[n]$ denotes the net effect of a failure in the $k^{\underline{th}}$ converter on the result. $\phi_k[n]$ is zero except for samples that came from the $k^{\underline{th}}$ converter. The output may equivalently be written as

$$z[n] = s(nT) + e[n] \tag{7.3}$$

where $e[n] = \sum_{j=1}^{\lambda} \phi_{k_j}[n]$ equals the net error. Comparing (7.3) and (4.21), we see that the redundancy present in the output is of the same form as that studied in Chapter 4. Valid results lie in a subspace $\tilde{G}$ of H, and an additive error model is assumed. Thus we may use the results of Section 4.3 to analyze this system.

We assume that when converter k fails, it corrupts its output samples in an arbitrary manner, $\phi_k[n] \in \mathbb{R}$. This constitutes a symmetric error model, and we may therefore analyze the redundancy present in the system via its minimum distance. The minimum distance is most easily computed by application of (4.48). We must determine the minimum number of errors such that the net effect of all errors is a low-pass signal bandlimited to $\pm\frac{\pi N}{N+C}$ radians. This analysis is most easily performed in the frequency domain. Let

$$\Phi_k(\omega) = \sum_{n=-\infty}^{\infty} \phi_k[n] e^{-j\omega n} \tag{7.4}$$

denote the discrete-time Fourier transform of a single error $\phi_k[n]$. Because $\phi_k[n]$ equals zero except for samples that came from the $k^{\underline{th}}$ converter, $\Phi_k(\omega)$ has a periodic structure,

$$\Phi_k\left(\omega + \frac{2\pi l}{N+C}\right) = \Phi_k(\omega) e^{-\frac{j2\pi lk}{N+C}}. \tag{7.5}$$

$\Phi_k(\omega)$ contains $N + C$ complete copies of the fault spectrum in an interval of width 2π. N copies are contained in the low frequency region, and C copies are contained in the high frequency region where there is no energy from the low-pass input signal. This is illustrated in Figure 7-2.

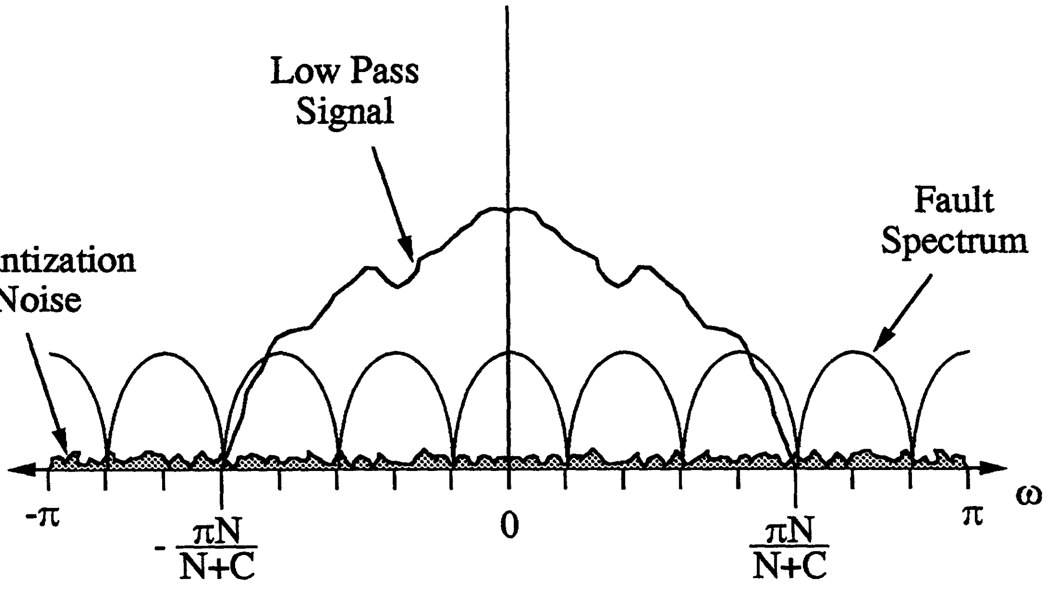


Figure 7-2: Output frequency spectrum of round-robin A/D converter system.

Using the linearity of the Fourier transform, we may write the transform of the net error $e[n]$ as

$$E(\omega) = \sum_{j=1}^{\lambda} \Phi_{k_j}(\omega). \tag{7.6}$$

In order to determine the minimum distance of this code, we must determine the smallest value of λ such that $E(\omega) = 0$ in the interval $\frac{\pi N}{N+C} < \omega < \frac{\pi(N+2C)}{N+C}$ for some nonzero errors $\Phi_{k_j}(\omega)$. Since $E(\omega)$ is composed of a sum of errors $\Phi_k(\omega)$ which have a periodic spectrum, we may evaluate this on a point-by-point basis. Let $\frac{\pi N}{N+C} < \omega_0 < \frac{\pi(N+2)}{N+C}$ be some frequency in the interval spanned by the first high frequency copy of the fault spectrum. C values of the high frequency portion of $\Phi_k(\omega)$ are linearly related to $\Phi_k(\omega_0)$,

$$\begin{aligned}
\Phi_k(\omega_0) &= \Phi_k(\omega_0) & (7.7)\\
\Phi_k\left(\omega_0 + \frac{2\pi}{N+C}\right) &= \Phi_k(\omega_0)W^k & (7.8)\\
&\vdots & (7.9)\\
\Phi_k\left(\omega_0 + \frac{2\pi(C-1)}{N+C}\right) &= \Phi_k(\omega_0)W^{k(C-1)}
\end{aligned}$$

where $W = e^{-\frac{j2\pi}{N+C}}$. Combining this with (7.6), we may write

$$\begin{aligned} E(\omega_0) &= \sum_{j=1}^{\lambda} \Phi_{k_j}(\omega_0) && (7.10) \\ E\left(\omega_0 + \frac{2\pi}{N+C}\right) &= \sum_{j=1}^{\lambda} \Phi_{k_j}(\omega_0) W^{k_j} && (7.11) \\ &\vdots && (7.12) \\ E\left(\omega_0 + \frac{2\pi(C-1)}{N+C}\right) &= \sum_{j=1}^{\lambda} \Phi_{k_j}(\omega_0) W^{k_j(C-1)}. \end{aligned}$$

We must determine the minimum value of λ such that the above equations all equal 0 for some set of nonzero errors. Express these equations in matrix form as $\boldsymbol{E} = \boldsymbol{W\Phi}$ where

$$\boldsymbol{E} = \begin{bmatrix} E(\omega_0) \\ E\left(\omega_0 + \frac{2\pi}{N+C}\right) \\ \vdots \\ E\left(\omega_0 + \frac{2\pi(C-1)}{N+C}\right) \end{bmatrix} \qquad \boldsymbol{W} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ W^{k_1} & W^{k_2} & \cdots & W^{k_\lambda} \\ \vdots & \vdots & & \vdots \\ W^{k_1(C-1)} & W^{k_2(C-1)} & \cdots & W^{k_\lambda(C-1)} \end{bmatrix}$$

$$\boldsymbol{\Phi} = \begin{bmatrix} \Phi_{k_1}(\omega_0) \\ \Phi_{k_2}(\omega_0) \\ \vdots \\ \Phi_{k_\lambda}(\omega_0) \end{bmatrix}.$$

If $\boldsymbol{\Phi} \neq 0$, then $\boldsymbol{E}$ will be zero only if $\boldsymbol{W}$ is not full rank. We must therefore determine the smallest value of λ such that the columns of $\boldsymbol{W}$ are linearly dependent. The ith and jth columns of $\boldsymbol{W}$ are linearly independent if $k_i \neq k_j$. Thus the columns of $\boldsymbol{W}$ become linearly dependent only when $\lambda = C + 1$. Therefore, this oversampling technique generates a code with a minimum distance of $C + 1$.

With $C = 1$ extra converter, we can detect single failures. With $C = 2$ extra converters, we can detect and correct single failures. (We will show that due to the nonidealness of realizable filters, at least $C = 3$ extra converters are needed to correct single errors.)

We can detect and correct errors using a syndrome homomorphism ψ. Since $\tilde{G}$ equals the subspace of low-pass signals bandlimited to $\pm\frac{\pi N}{N+C}$ radians, the quotient space $H/\tilde{G}$ can be shown to be isomorphic to the vector space of discrete-time high-pass signals containing frequencies above $\pm\frac{\pi N}{N+C}$ radians. Denote this quotient space by T. The syndrome homomorphism from H onto T contains $\tilde{G}$ in its kernel. An obvious homomorphism which satisfies this constraint is a high-pass filter with cutoff frequency $\pm\frac{\pi N}{N+C}$ radians,

$$\psi(z[n]) = z[n] * h_{hp}[n] \tag{7.13}$$

where $h_{hp}[n]$ denotes the impulse response of the ideal high-pass filter. If the output of the filter equals zero, then $z[n] \in \tilde{G}$. Otherwise, if the output is nonzero, then an error has occurred. The broken converter is identified from the phase difference between high frequency copies of the fault spectrum. Then the fault is reconstructed by averaging the C copies in the high frequency region, and it is subtracted from the observations to estimate the fault-free signal.

The coding scheme we use is similar to one developed by Wolf [46]. He shows that under certain conditions, discrete-time sequences carry redundant information which allows detection and correction of errors. Specifically, sequences whose discrete Fourier transforms contain zeros can be protected against impulse noise. Wolf's error detection and correction scheme is based on coding theory, while ours utilizes a generalized likelihood ratio test. Both methods use out-of-band energy to detect errors.

A/D converters inherently quantize the input signal and, together with the dither circuitry, introduce additive white noise in $z[n]$. This radically alters the manner in which error detection and correction must be performed. Errors can no longer be detected by testing if $z[n] \in \tilde{G}$ since error-free signals are not strictly bandlimited. We will instead pursue an error detection and correction strategy that specifically takes into account the effects of quantization noise.

7.3 Algorithm Development

This section develops the optimal fault detection/correction algorithm using a generalized likelihood ratio test. We include the effects of quantization noise in our model and write the output of the round-robin system as

$$z[n] = s[n] + \epsilon[n] + \phi_k[n] \tag{7.14}$$

where $s[n]$ is the desired low-pass signal and $\epsilon[n]$ is white quantization noise which is uncorrelated with $s[n]$. Let $\underline{s}$ be a vector of all samples $s[n]$, and define $\underline{z}$, $\underline{\phi}_k$, and $\underline{\epsilon}$ similarly.

7.3.1 Generalized Likelihood Ratio Test

We will use a generalized likelihood ratio test to determine the most likely fault hypothesis and to correct the fault if necessary. Let H_* represent the hypothesis that no converter has failed, and let H_k represent a failure in the $k^{\underline{th}}$ converter where $0 \leq k \leq N + C - 1$. Define $p(H_*)$ and $p(H_k)$ as the a priori probabilities of these events, which we assume are independent of the signal or fault,

$$p(H_k) = p\left(H_k | \underline{s}, \underline{\phi}_k\right). \tag{7.15}$$

We must compute the likelihood L_k of each hypothesis H_k given the observed data $\underline{z}$. For hypothesis H_*, the likelihood unfortunately depends on the unknown signal $\underline{s}$. Therefore, we will maximize the likelihood over $\underline{s}$ to determine the likelihood L_* of hypothesis H_*,

$$L_* = \max_{\underline{s}} \log [p(\underline{z} \mid H_*, \underline{s}) p(H_*)]. \tag{7.16}$$

For hypotheses H_k, $k = 0, \ldots, N + C - 1$, the likelihoods depend on both the unknown signal $\underline{s}$ and the unknown fault $\underline{\phi}_k$. We therefore maximize over both $\underline{s}$ and $\underline{\phi}_k$ to determine the likelihood L_k of H_k,

$$L_k = \max_{\underline{s}, \underline{\phi}_k} \log \left[p\left(\underline{z} \mid H_k, \underline{s}, \underline{\phi}_k\right) p(H_k)\right]. \tag{7.17}$$

The most likely failure hypothesis is chosen by finding the largest likelihood. The failure estimate $\hat{\underline{\phi}}_k$ (if any) and a clean signal estimate $\hat{\underline{s}}$ are the values at which the likelihood is maximized.

The bulk of our derivation will be performed in the frequency domain where the distribution of samples of the noise transform is approximately white Gaussian noise. We will exploit this by approximating the noise as being zero mean Gaussian in both time and frequency:

$$p(\epsilon[n]) = N\left(0, \sigma_\epsilon^2\right). \tag{7.18}$$

where the variance $\sigma_\epsilon^2 = lsb^2/12$.

We begin by solving for L_*. Using the distribution of the quantization noise, (7.16) can be written as

$$L_* = \max_{\underline{s}} \left[\log p(H_*) - \frac{1}{2}\sum_{n=0}^{M-1} \log\left(2\pi\sigma_\epsilon^2\right) - \frac{1}{2\sigma_\epsilon^2}\sum_{n=0}^{M-1} (z[n] - s[n])^2\right] \tag{7.19}$$

where M is the number of samples available. The first two terms are constants and will be denoted as η_*. Define $Z(\omega_r)$, $S(\omega_r)$, and $\Phi_k(\omega_r)$ to be the M point DFTs of $z[n]$, $s[n]$, and $\phi_k[n]$. For long time intervals, Parseval's theorem can be applied to the third term in (7.19), giving

$$L_* = \max_{\underline{s}} \left[\eta_* - \frac{1}{2\sigma_\epsilon^2}\frac{1}{M}\sum_{r=0}^{M-1} |Z(\omega_r) - S(\omega_r)|^2\right] \tag{7.20}$$

where ω_r is a frequency index with value $\omega_r = 2\pi r/M$.

We will work with frequencies in the range of 0 to 2π, and divide the frequency samples into a low frequency region, Ω_L, which contains signal energy, and a high frequency region, Ω_H, which does not. The regions will be divided as follows:

$$\begin{aligned} \Omega_L &= \{\omega_r \,|\, r = 0, \ldots, M_L - 1 \text{ and } r = M_H, \ldots, M-1\} \\ \Omega_H &= \{\omega_r \,|\, r = M_L, \ldots, M_H - 1\} \end{aligned} \tag{7.21}$$

where

$$M_L = \left(\tfrac{M}{N+C}\right)\tfrac{N}{2} \quad \text{and} \quad M_H = M - \left(\tfrac{M}{N+C}\right)\tfrac{N}{2}. \tag{7.22}$$

Assume that $M/2$ is a multiple of $N+C$ so that the frequency samples can be easily divided into the two regions.

Since $S(\omega_r)$ is bandlimited, (7.20) can be rewritten as

$$L_* = \max_{\underline{s}} \left[\eta_* - \frac{1}{2\sigma_\epsilon^2} \frac{1}{M} \left\{ \sum_{\omega_r \in \Omega_L} |Z(\omega_r) - S(\omega_r)|^2 + \sum_{\omega_r \in \Omega_H} |Z(\omega_r)|^2 \right\} \right]. \tag{7.23}$$

Now maximize with respect to $S(\omega_r)$ to obtain,

$$\hat{S}(\omega_r) = \begin{cases} Z(\omega_r) & \text{for } \omega_r \in \Omega_L \\ 0 & \text{else.} \end{cases} \tag{7.24}$$

This can also be written as,

$$\hat{S}(\omega_r) = H_{LP}(\omega_r) Z(\omega_r) \tag{7.25}$$

where $H_{LP}(\omega_r)$ is an ideal low pass filter with frequency response

$$H_{LP}(\omega_r) = \begin{cases} 1 & \text{for } \omega_r \in \Omega_L \\ 0 & \text{else.} \end{cases} \tag{7.26}$$

If there are no faults, so that hypothesis H_* is true, then this optimally "clean" signal estimate $\hat{S}(\omega_r)$ is found by low pass filtering $Z(\omega_r)$ to remove high frequency quantization noise. Substituting (7.24) into (7.23), the likelihood of H_* becomes,

$$L_* = \eta_* - \frac{1}{2\sigma_\epsilon^2} \frac{1}{M} \sum_{\omega_r \in \Omega_H} |Z(\omega_r)|^2 . \tag{7.27}$$

It is convenient to define $Z_H(\omega_r)$ as a high pass version of $Z(\omega_r)$,

$$Z_H(\omega_r) = H_{HP}(\omega_r) Z(\omega_r) \tag{7.28}$$

where $H_{HP}(\omega_r)$ is an ideal high pass filter with,

$$H_{HP}(\omega_r) = \begin{cases} 1 & \text{for } \omega_r \in \Omega_H \\ 0 & \text{else.} \end{cases} \tag{7.29}$$

Then since (7.27) only depends on the high frequency samples of $Z(\omega_r)$, we can write

$$L_* = \eta_* - \frac{1}{2\sigma_\epsilon^2}\frac{1}{M}\sum_{r=0}^{M-1} |Z_H(\omega_r)|^2 . \tag{7.30}$$

Applying Parseval's theorem and returning to the time domain results in,

$$L_* = \eta_* - \frac{1}{2\sigma_\epsilon^2}\sum_{n=0}^{M-1} z_H^2[n]. \tag{7.31}$$

Now consider the case of a failure in the $k^{\underline{th}}$ converter. As before, we apply Parseval's theorem to the time domain likelihood expression for L_k and obtain

$$L_k = \max_{\underline{s},\underline{\phi}_k} \left[\log p(H_k) - \frac{1}{2}\sum_{n=0}^{M-1} \log\left(2\pi\sigma_\epsilon^2\right) - \frac{1}{2\sigma_\epsilon^2}\frac{1}{M}\sum_{r=0}^{M-1} |Z(\omega_r) - S(\omega_r) - \Phi_k(\omega_r)|^2\right] . \tag{7.32}$$

The first two terms of this expression are constants and we will denote them as η_k. Divide the summation into low and high frequency regions and maximize with respect to $S(\omega_r)$ to obtain

$$\hat{S}(\omega_r) = H_{LP}(\omega_r)\left[Z(\omega_r) - \Phi_k(\omega_r)\right] . \tag{7.33}$$

Substituting this back into (7.32), the log likelihood function becomes

$$L_k = \max_{\underline{\phi}_k} \left[\eta_k - \frac{1}{2\sigma_\epsilon^2}\frac{1}{M}\sum_{\omega_r\in\Omega_H} |Z(\omega_r) - \Phi_k(\omega_r)|^2\right] . \tag{7.34}$$

Since the summation is only over high frequencies, substitute $Z_H(\omega_r)$ for $Z(\omega_r)$. Next, extend the summation over all frequencies and subtract any newly introduced terms,

$$L_k = \max_{\underline{\phi}_k} \left[\eta_k - \frac{1}{2\sigma_\epsilon^2}\frac{1}{M}\left\{\sum_{r=0}^{M-1} |Z_H(\omega_r) - \Phi_k(\omega_r)|^2 - \sum_{\omega_r\in\Omega_L} |\Phi_k(\omega_r)|^2\right\}\right] . \tag{7.35}$$

In order to reduce this expression further, we must exploit the structure of $\Phi_k(\omega_r)$. $\phi_k[n]$ only contains samples from the faulty converter, and it is zero except for one out of every $N + C$ samples. Its transform $\Phi_k(\omega_r)$ consists of $N + C$ copies of the fault spectrum, each with a phase shift that depends on which converter failed,

$$\Phi_k\left(\omega_r + \frac{2\pi l}{N+C}\right) = \Phi_k(\omega_r)e^{-\frac{j2\pi l}{N+C}k}. \tag{7.36}$$

Therefore we may write

$$\left|\Phi_k\left(\omega_r + \frac{2\pi l}{N+C}\right)\right| = |\Phi_k(\omega_r)| \tag{7.37}$$

for all ω_r and all integers l. We use this to extend the second summation in (7.35) to include all frequencies,

$$L_k = \max_{\underline{\phi}_k}\left[\eta_k - \frac{1}{2\sigma_\epsilon^2}\frac{1}{M}\left\{\sum_{r=0}^{M-1}|Z_H(\omega_r) - \Phi_k(\omega_r)|^2 - \frac{N}{N+C}\sum_{r=0}^{M-1}|\Phi_k(\omega_r)|^2\right\}\right]. \tag{7.38}$$

Applying Parseval's theorem and combining terms we obtain,

$$L_k = \max_{\underline{\phi}_k}\left[\eta_k - \frac{1}{2\sigma_\epsilon^2}\sum_{n=0}^{M-1}\left(z_H^2[n] - 2\phi_k[n]z_H[n] + \frac{C}{N+C}\phi_k^2[n]\right)\right]. \tag{7.39}$$

We can now maximize this expression and solve for the fault estimate. For a failure in the $k^{\underline{th}}$ converter, the only nonzero samples of $\phi_k[n]$ are those for which $n \equiv k$ (we will use this notation as shorthand for n mod $(N+C) = k$.) Therefore maximizing (7.39) yields,

$$\hat{\phi}_k[n] = \begin{cases} \frac{N+C}{C}z_H[n] & \text{for } n \equiv k \\ 0 & \text{else} \end{cases} \tag{7.40}$$

as the optimal fault estimate.

Now, to obtain the likelihood of H_k, substitute (7.40) back into (7.39). After some algebra we are left with,

$$L_k = \eta_k - \frac{1}{2\sigma_\epsilon^2}\sum_{n=0}^{M-1} z_H^2[n] + \frac{1}{2\sigma_\epsilon^2}\frac{N+C}{C}\sum_{n\equiv k} z_H^2[n] \tag{7.41}$$

Assume that all a priori failure probabilities $p(H_k)$ are the same for $k = 0, \ldots, N+C-1$, so that the constants η_k are all equal,

$$\eta_k = \eta \text{ for } k = 0, \ldots, N+C-1. \tag{7.42}$$

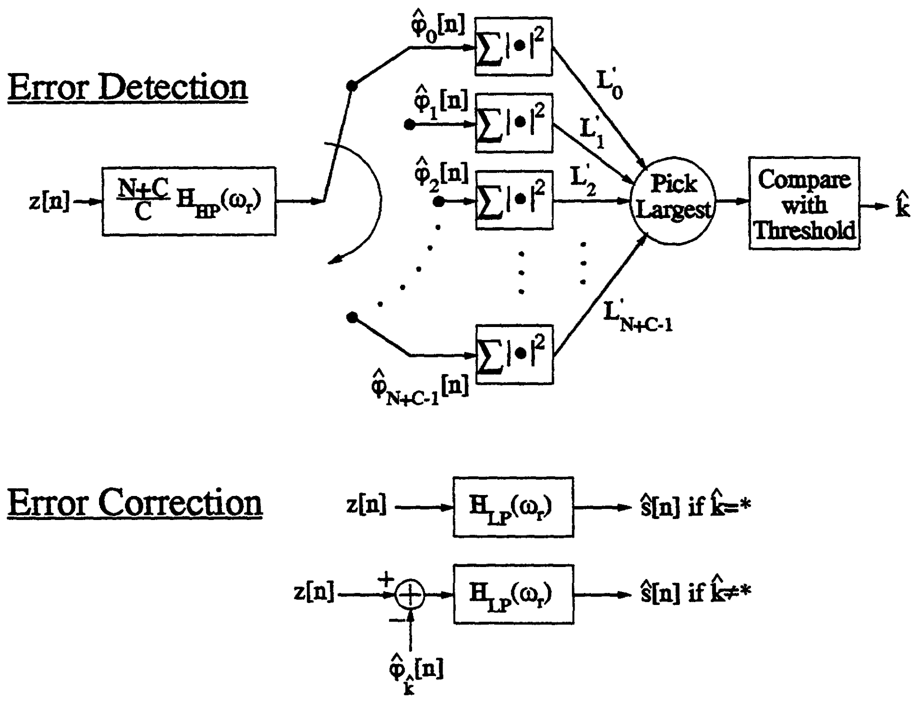


Figure 7-3: Ideal error detection and correction system.

Then it is convenient to use scaled relative likelihoods defined by,

$$L'_k = 2\sigma_\epsilon^2 \left(\frac{N+C}{C}\right)[L_k - L_* - \eta + \eta_*] \text{ for } k = * \text{ and } 0,\ldots,N+C-1. \tag{7.43}$$

The scaled relative likelihoods reduce to,

$$L'_* = \gamma \tag{7.44}$$

$$L'_k = \left(\frac{N+C}{C}\right)^2 \sum_{n\equiv k} z_H^2[n] = \sum_{n\equiv k} \hat{\phi}_k^2[n] \tag{7.45}$$

where

$$\gamma = 2\sigma_\epsilon^2 \left(\frac{N+C}{C}\right)\left[\log \frac{p(H_*)}{p(H_k)}\right] \tag{7.46}$$

is a constant. For hypothesis $k = 0,\ldots,N+C-1$, the best estimate of the original signal is then,

$$\hat{S}(\omega_r) = H_{LP}(\omega_r)\left[Z(\omega_r) - \hat{\Phi}_k(\omega_r)\right]. \tag{7.47}$$

The complete generalized likelihood ratio test fault correction system is shown in Figure 7-3. The dithered sampled signal $z[n]$ is first scaled by $\frac{N+C}{C}$ and filtered by $H_{HP}(\omega_r)$. The output of this filter, $\frac{N+C}{C}z_H[n]$, is sorted into $N+C$ interleaved streams, with the $k^{\underline{th}}$ stream receiving every $(N+C)^{\underline{th}}$ sample starting with sample k. Each of these streams is simply $\hat{\phi}_k[n]$, the best least squares estimate of the fault in the $k^{\underline{th}}$ converter, assuming that converter k is faulty. We measure the energy in each of these fault estimates, and if any energy is greater than the threshold γ, we declare that a converter has failed. The largest energy indicates the best guess $\hat{k}$ of which converter has failed. The signal is then reconstructed by fixing the incorrect samples (if any) and low pass filtering.

7.3.2 Required Number of Extra Converters

Insight into choosing an appropriate number of extra converters C can be gained by writing (7.40) in the frequency domain,

$$\hat{\Phi}_k(\omega_r) = \frac{1}{C}\sum_{l=0}^{N+C-1} Z_H\left(\omega_r + \frac{2\pi l}{N+C}\right) e^{\frac{j2\pi k}{N+C}l}. \tag{7.48}$$

The magnitude of this function is periodic in ω_r with period $\frac{2\pi}{N+C}$. Since $Z_H(\omega_r)$ is high pass, for a given ω_r only C terms in this sum are nonzero. Each $\frac{M}{N+C}$ point "period" of $\hat{\Phi}_k(\omega_r)$ is thus formed by averaging C sections of $Z_H(\omega_r)$ with appropriate phase shifts.

If $C = 1$, no averaging takes place. The fault estimates, $\hat{\Phi}_k(\omega_r)$ for $k = 0, \ldots, N+C-1$ will differ only by a phase shift, and the energy in each will be the same. The system will only be able to decide if a fault has occurred, but will not be able to determine the specific faulty converter. If $C \geq 2$, then the energy in $\hat{\Phi}_k(\omega_r)$ is maximized when the proper phase shift is applied so that all C non-zero terms add coherently. Thus, with $C \geq 2$ we can achieve single fault correction.

7.3.3 Probability of False Alarm

The hypothesis testing procedure can make several errors. A "false alarm" occurs when H_* is true but the algorithm selects H_k as most likely, for some $k \neq *$. "Detection" occurs if converter q has failed and the method chooses any hypothesis except H_*. "Misdiagnosis" occurs when converter q has failed, hypothesis H_k is diagnosed, and $k \neq q$. Let P_F, P_D, and P_M represent the probabilities of these events.

We can develop a Neyman–Pearson test which adjusts the threshold γ to achieve a given probability of false alarm, P_F. Suppose H_* is true and that the quantization noise $\epsilon[n]$ is white zero mean Gaussian with variance σ_ϵ^2. Now

$$z_H[n] = \sum_{l=0}^{M-1} h_{HP}[n-l]\epsilon[l]. \tag{7.49}$$

Since the samples $\epsilon[n]$ are Gaussian, so are the samples $z_H[n]$ with mean and covariance given by,

$$\begin{aligned} \mathrm{E}\left[z_H[n] | H_*\right] &= 0 && (7.50) \\ \mathrm{Cov}\left[z_H[n], z_H[m] | H_*\right] &= \sum_{l=0}^{M-1} h_{HP}[n-l]h_{HP}[m-l]\sigma_\epsilon^2 \\ &= h_{HP}[n-m]\sigma_\epsilon^2 && (7.51) \end{aligned}$$

where the last line follows because $h_{HP}[n] * h_{HP}[n] = h_{HP}[n]$, where '$*$' represents circular convolution with period $N+C$. Since $h_{HP}[n-m]$ has zeroes spaced $(N+C)/C$ apart, samples of $z_H[n]$ spaced by multiples of $(N+C)/C$ are statistically independent. Equation (7.40) thus implies that the non-zero samples of $\hat{\phi}_k[n]$ are independent Gaussian random variables with zero mean and variance $\left(\frac{N+C}{C}\right)\sigma_\epsilon^2$. Equation (7.45) implies that under hypothesis H_*, each L_k' is the sum of the squares of $M/(N+C)$ independent Gaussian random variables, $\hat{\phi}_k[n]$. Each L_k' is thus Chi-square distributed with $D = \frac{M}{N+C}$ degrees of freedom, and

$$P\left(L_k' \geq \gamma | H_*\right) = \frac{1}{\Gamma(D/2)} \int_{\frac{\gamma C}{2\sigma_\epsilon^2(N+C)}}^{\infty} t^{D/2-1} e^{-t} dt \tag{7.52}$$

where $\Gamma(x) = (x-1)!$ is the normalization factor.

A false alarm occurs when H_* is true, but one or more likelihoods are greater than γ. Thus

$$P_F = 1 - P\left(L_0' \leq \gamma, L_1' \leq \gamma, \ldots, L_{N+C-1}' \leq \gamma | H_*\right). \tag{7.53}$$

Since $z_H[n]$ is a bandpass signal with bandwidth $\pm\pi C/(N+C)$, the L_k' are highly correlated with each other. Samples of $z_H[n]$ spaced by $(N+C)/C$ will be independent of each other, and the others are determined by linear interpolation. This implies that likelihoods spaced by $(N+C)/C$ are independent of each other. Since there are C of these,

$$\begin{aligned} P_F &\approx 1 - P\left(L'_0 \leq \gamma, L'_{\frac{N+C}{C}} \leq \gamma, L'_{2\left(\frac{N+C}{C}\right)} \leq \gamma, \ldots, L'_{(C-1)\left(\frac{N+C}{C}\right)} \leq \gamma \middle| H_*\right) \\ &= 1 - P\left(L'_0 \leq \gamma \middle| H_*\right)^C \\ &= 1 - \left[1 - P\left(L'_0 \geq \gamma \middle| H_*\right)\right]^C \\ &\approx CP\left(L'_0 \geq \gamma \middle| H_*\right) \end{aligned} \tag{7.54}$$

where the approximation in the last line is valid for small P_F. We can thus use the chi-squared formula in (7.52) to set γ to achieve any desired level for P_F.

If the integration interval M is much larger than the number of converters $N+C$, then the number of degrees of freedom D will be large, and the distribution of L'_k can be approximated as Gaussian. In this case, a good approximation for P_F is in terms of an error function:

$$P_F \approx \frac{C}{2}\operatorname{erfc}\left(\frac{\gamma - \mathrm{E}\left[L'_0 \middle| H_*\right]}{\sqrt{2\mathrm{Var}\left[L'_0 \middle| H_*\right]}}\right) \tag{7.55}$$

where:

$$\operatorname{erfc}(x) = \frac{2}{\sqrt{\pi}}\int_x^\infty e^{-t^2}\,dt \tag{7.56}$$

and where the mean and variance of the likelihoods are given in Appendix 7.B:

$$\mathrm{E}\left[L'_0 \middle| H_*\right] = \frac{M}{N+C}\left(\frac{N+C}{C}\right)\sigma_\epsilon^2 \tag{7.57}$$

$$\mathrm{Var}\left[L'_0 \middle| H_*\right] = 2\frac{M}{N+C}\left(\frac{N+C}{C}\right)^2\sigma_\epsilon^4. \tag{7.58}$$

In other words, γ should be set to:

$$\gamma = \mathrm{E}\left[L'_0 \middle| H_*\right] + \beta\sqrt{\mathrm{Var}\left[L'_0 \middle| H_*\right]} \tag{7.59}$$

where:

$$\beta = \sqrt{2}\operatorname{erfc}^{-1}\left(\frac{2P_F}{C}\right) \tag{7.60}$$

A reasonable approximation to erfc(x) is $\frac{1}{x+\sqrt{x^2+2}}e^{-x^2}$. Therefore P_F falls rapidly as γ increases. It is usually better to set γ somewhat too high, rather than too low.

7.3.4 Probability of Detection

The probability of detection P_D is defined as the probability that a fault is declared on some converter given that a fault has occurred on converter q. In practice, this is approximately equal to the probability that likelihood $L'_q \geq \gamma$:

$$\begin{aligned} P_D &= 1 - P\left(L'_0 \leq \gamma, \ldots, L'_{N+C-1} \leq \gamma \middle| H_q, \underline{\phi}_q\right) \\ &\approx P\left(L'_q \geq \gamma \middle| H_q, \underline{\phi}_q\right) \end{aligned} \tag{7.61}$$

If the number of terms summed in the likelihoods is large, $M/(N+C) \gg 1$, then we can approximate L'_q as Gaussian. Appendix 7.B shows that:

$$P_D \approx 1 - \frac{1}{2}\text{erfc}\left(\frac{\mathrm{E}\left[L'_q \middle| H_q, \underline{\phi}_q\right] - \gamma}{\sqrt{2\mathrm{Var}\left[L'_q \middle| H_q, \underline{\phi}_q\right]}}\right) \tag{7.62}$$

where:

$$\mathrm{E}\left[L'_q \middle| H_q, \underline{\phi}_q\right] = \frac{M}{N+C}\left(\frac{N+C}{C}\right)\sigma_\epsilon^2\left[1 + \mathrm{FNR}_q\left(\frac{C}{N+C}\right)\right] \tag{7.63}$$

$$\mathrm{Var}\left[L'_q \middle| H_q, \underline{\phi}_q\right] = 2\frac{M}{N+C}\left(\frac{N+C}{C}\right)^2\sigma_\epsilon^4\left[1 + 2\mathrm{FNR}_q\left(\frac{C}{N+C}\right)\right] \tag{7.64}$$

and where FNR_q is the fault-to-noise ratio on converter q:

$$\mathrm{FNR}_q = \frac{N+C}{M}\sum_{l=0}^{M-1}\frac{\phi_q^2[l]}{\sigma_\epsilon^2} \tag{7.65}$$

Since erfc(x) decays faster than rate e^{-x^2}, we expect P_D to approach 1 exponentially as FNR_q increases or as the integration length $M/(N+C)$ increases.

7.3.5 Probability of Misdiagnosis

A misdiagnosis occurs when a fault occurs on converter q, a fault is declared, but the wrong converter is identified. This will cause the algorithm to "correct" the wrong converter. The probability of converter misdiagnosis is difficult to compute analytically, but it is possible to gain some insight by examining a simpler measure. We can compute instead the probability that some likelihood L'_k is greater than L'_q given that H_q is true and that the fault is $\underline{\phi}_q$. Assuming that the number of terms summed in the likelihoods is large, $M/(N+C) \gg 1$, then the likelihoods L'_k and L'_q can be approximated as jointly Gaussian. Then Appendix 7.B shows that:

$$P\left(L'_k \geq L'_q \middle| H_q, \underline{\phi}_q\right) \approx \frac{1}{2}\operatorname{erfc}\left(\frac{\mathrm{E}\left[L'_q - L'_k \middle| H_q, \underline{\phi}_q\right]}{\sqrt{2\mathrm{Var}\left[L'_q - L'_k \middle| H_q, \underline{\phi}_q\right]}}\right) \tag{7.66}$$

where

$$\mathrm{E}\left[L'_q - L'_k \middle| H_q, \underline{\phi}_q\right] = \frac{M}{N+C}\sigma_\epsilon^2\left(1 - S^2(k-q)\right)\mathrm{FNR}_q \tag{7.67}$$

$$\begin{aligned}\mathrm{Var}\left[L'_q - L'_k \middle| H_q, \underline{\phi}_q\right] &= \\ & 4\frac{M}{N+C}\left(\frac{N+C}{C}\right)^2\sigma_\epsilon^4\left(1 - S^2(k-q)\right)\left(1 + \mathrm{FNR}_q\left(\frac{C}{N+C}\right)\right)\end{aligned} \tag{7.68}$$

where

$$S(l) = \frac{\sin\left(\frac{\pi C}{N+C}l\right)}{C\sin\left(\frac{\pi}{N+C}l\right)} \tag{7.69}$$

is a circular sinc function with $S(0) = 1$, and FNR_q is defined in (7.65).

For fixed N, C, and M, and for $k \neq q$, $S(k-q)$ is maximized for $k = q \pm 1$. Thus, the most common misdiagnosis declares an adjacent neighbor of the faulty converter to be faulty. Also note that increasing the number of samples contributing to the likelihood, $M/(N+C)$, increasing the fraction of extra converters $C/(N+C)$, or increasing the size of the fault, FNR_q, all decrease the misdiagnosis probability.

7.3.6 Variance of Low Pass Signal Estimate

In this section we consider the accuracy of our system in estimating the original low pass signal. The equations presented are derived in Appendix 7.A. First, consider the case when all converters are functioning properly and hypothesis H_* is chosen. We can show that the expected value and variance of our estimator equals:

$$\begin{aligned} \mathrm{E}\left[\hat{s}[n] | H_*\right] &= s[n] && (7.70) \\ \mathrm{Var}\left[\hat{s}[n] | H_*\right] &= \frac{N}{N+C}\sigma_\epsilon^2. && (7.71) \end{aligned}$$

Now suppose that the $q^{\underline{th}}$ converter is broken with actual fault $\phi_q[n]$, and that hypothesis q is correctly chosen as the most likely. Under these conditions, we can show that:

$$\mathrm{E}\left[\hat{s}[n] | H_q, \underline{\phi}_q\right] = s[n] \quad (7.72)$$

$$\mathrm{Var}\left[\hat{s}[n] | H_q, \underline{\phi}_q\right] = \begin{cases} \frac{N}{C}\sigma_\epsilon^2 & \text{for } n \equiv q \\ \frac{\sigma_\epsilon^2}{N+C}\left[N + CS^2(k-q)\right] & \text{else} \end{cases} \quad (7.73)$$

Thus we see that our estimator $\hat{s}[n]$ is unbiased. Signal estimate samples for the faulty converter have variance N/C times larger than the quantization noise of a working converter, σ_ϵ^2. All the other signal estimate samples, however, have variance below σ_ϵ^2. The average signal estimate variance is given by

$$\frac{1}{M}\sum_{n=0}^{M-1} \mathrm{Var}\left[\hat{s}[n] | H_q, \underline{\phi}_q\right] = \left(\frac{N}{N+C}\right)\left(\frac{C+1}{C}\right)\sigma_\epsilon^2. \quad (7.74)$$

7.4 Simulation of Ideal System

In this section we present computer simulation results for the ideal generalized likelihood ratio test. The system studied had $N = 5$, $C = 3$, and $M = 1024$. The A/D converters were modeled as having a dynamic range of $\pm\frac{1}{2}$ volt and $B = 12$ bit resolution. Quantization noise was modeled as uniformly distributed between $\pm\frac{1}{2}lsb$, with variance $\sigma_\epsilon^2 = lsb^2/12$. Despite this non-Gaussian noise, the likelihoods are formed from so many independent terms, $M/(N + C) = 128$, that they can be accurately modeled as having a Gaussian

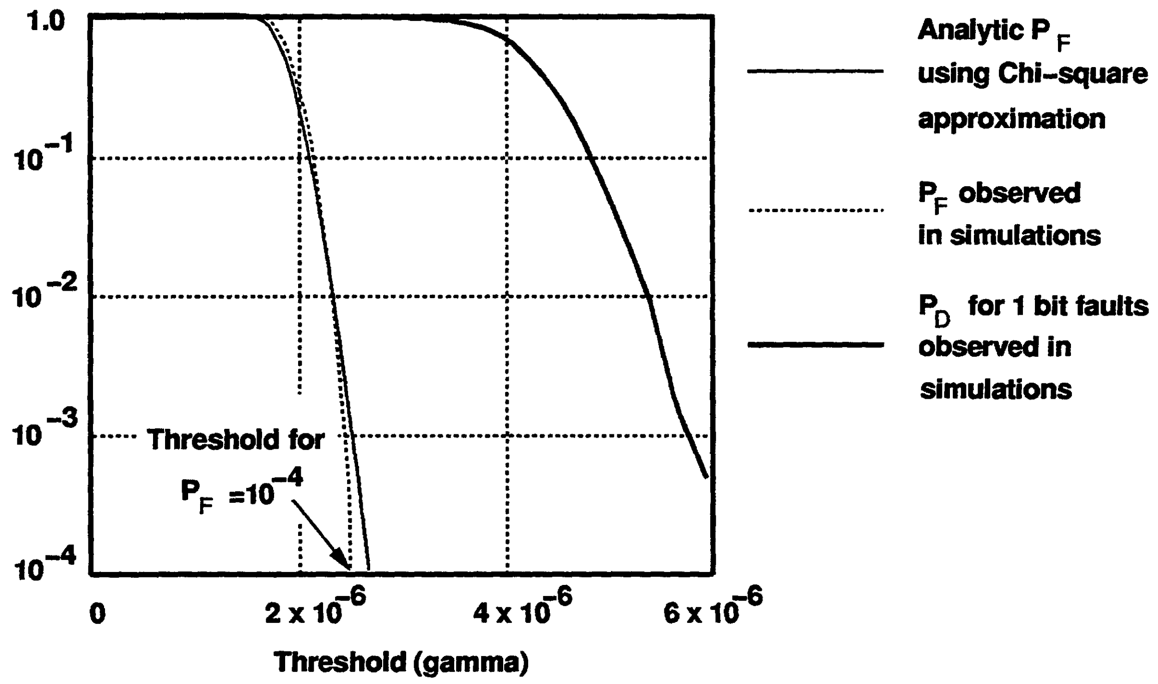


Figure 7-4: Comparison of analytic and measured likelihoods for the ideal system.

distribution.

Substituting into our formula for P_F in (7.55) gives:

$$P_F \approx 1.5\text{erfc}\left(\frac{\gamma - 341\sigma_\epsilon^2}{60.3\sigma_\epsilon^2}\right) \tag{7.75}$$

For example, a value of $\beta = 3.85$ in (7.59) yields $\gamma = 505\sigma_\epsilon^2$ and $P_F = 10^{-4}$. To test this formula, we performed 10,000 simulations without faults and recorded the likelihoods. A graph of P_F vs. γ as predicted by (7.54) is compared with these computer simulation results in Figure 7-4. As expected, the simulation and analytic results are very close, and thus either our chi-squared formula (7.54) or Gaussian approximation formula (7.55) can be used to set γ.

Next, in order to measure P_D, we simulated the system with a faulty converter. A fault of F bits corresponds to adding random noise uniformly distributed over a range $\pm 2^{F-B-1}$ to the input before quantizing. When $F = B$ the A/D converter has completely failed; when $F = 1$ only the lsb is broken. Approximating the likelihoods as Gaussian, formula (7.62), and recognizing that a 1 bit fault corresponds to $\mathrm{FNR}_q = 4$, we find that

$$P_D \approx 1 - 0.5\mathrm{erfc}\left(\frac{853\sigma_\epsilon^2 - \gamma}{121\sigma_\epsilon^2}\right) \tag{7.76}$$

for a 1 bit fault. We performed 10,000 simulations with 1 bit faults and a graph of the ideal versus the experimental P_D vs. γ is shown in Figure 7-4. Fault detection is highly reliable. For example, the same value of γ which gives $P_F = 10^{-4}$ in our example yields $P_D = 1 - 10^{-5}$. In fact, with this γ, every fault was detected in our 10,000 tests.

The probability of misdiagnosis is also low. During 10,000 simulations with 1 bit faults, we found that converter misdiagnosis occurred only 5 times, giving $P_M \approx 5 \times 10^{-4}$. When a mistake did occur, one of the faulty converters' immediate neighbors was always identified as being broken. These results are consistent with formula (7.66), which gives for our example:

$$P\left(L'_{q+1} \geq L'_q \middle| H_q, \underline{\phi}_q\right) \approx 4 \times 10^{-4} \tag{7.77}$$

The simulation results also matched the predictions of signal variance made in section 7.3.6. When H_* is true, the average variance of $\hat{s}[n]$ was found to be $0.62\sigma_\epsilon^2$, and this agrees well with (7.71). When any size fault occurred, the variance of $\hat{s}[n]$ was found to be $0.832\sigma_\epsilon^2$, which agrees with (7.74).

7.5 Realistic Systems

In practice, the data stream to be processed is infinite in length, $M = \infty$, and so our batch oriented ideal system is unrealizable. In this section we consider several modifications to the ideal scheme in order to make it practical. First, we remove the low pass filter on the signal estimate, since it reduces the error by less than 1 bit. Second, we remove the dither system. Third, we replace the unrealizable ideal high pass filter with an FIR filter designed to minimize the variance of the signal estimate in case of failure. Fourth, we replace the infinite length integration to compute the likelihoods with finite length boxcar or IIR filters. Finally, we attempt to develop a robust real-time, causal strategy for declaring and correcting faults which works well for both continuous and transient faults.

7.5.1 Eliminate the Low Pass Filter

We can substantially reduce the computational complexity of estimating the signal in (7.25) or (7.47) by omitting the low pass filter on the final signal estimate, and instead using the approximate signal estimate $\hat{s}_A[n]$ defined by:

$$\hat{s}_A[n] = \begin{cases} z[n] & \text{if } H_* \text{ appears to be true} \\ z[n] - \hat{\phi}_k[n] & \text{if } H_k \text{ appears to be true.} \end{cases} \tag{7.78}$$

Appendix 7.A shows that the low pass filtering operation leaves the sample estimates from the faulty converter unchanged, $\hat{s}[n] = \hat{s}_A[n]$ for $n \equiv q$, and only affects samples from working converters. It also derives the mean and variance of this unfiltered estimator, assuming the fault is correctly diagnosed:

$$\mathrm{E}\left[\hat{s}_A[n] \middle| H_q, \underline{\phi}_q\right] = s[n] \tag{7.79}$$

$$\mathrm{Var}\left[\hat{s}_A[n] \middle| H_q, \underline{\phi}_q\right] = \begin{cases} \frac{N}{C}\sigma_\epsilon^2 & \text{if } n \equiv q \\ \sigma_\epsilon^2 & \text{else.} \end{cases} \tag{7.80}$$

The estimator is unbiased, and for typical values of N and C, the average variance is only slightly greater than that of the low pass filtered estimator. Thus, eliminating the low pass filter cuts the computational load by almost half, with little loss in accuracy.

We repeated our previous simulation and estimated the variance of the unfiltered signal estimate. We found that regardless of the size of the fault, the average variance was $1.086\sigma_\epsilon^2$ without the low pass filter, compared to $0.832\sigma_\epsilon^2$ with the low pass. This matches our theoretical predictions, and is a strong argument for eliminating the low pass.

7.5.2 Eliminate the Dither System

The dither system at the front end of the A/D system was used to ensure that the quantization noise was white and uncorrelated with the signal. In practice, if the signal is sufficiently random, without long slow ramps or constant portions, then the noise will be nearly white even without dither. In our simulations with quantized Gaussian signals, we could detect little difference between systems using dither and those without. We can thus eliminate the dither system, simplifying the hardware and eliminating a potential failure without severely degrading performance.

7.5.3 Finite Length Filter

In a practical system with an infinite data stream it is convenient to replace the ideal high pass filter $H_{HP}(\omega_r)$ in (7.29) with a finite order FIR filter. This has an important impact on the choice of C. Unlike the ideal high pass, an FIR filter will have a finite width transition region. To avoid signal leakage, the filter's stopband will have to fill the low frequency region Ω_L. The transition band will have to be inside the high frequency region Ω_H, which leaves only a portion of the high frequency region for the passband. At least two complete copies of the fault spectrum are needed in the filter's passband in order to be able to correct any faulty converter. To achieve this with an FIR filter will require at least $C = 3$ extra converters.

There are many possible ways to design the FIR high pass filter; we could window the ideal high pass, we could use Parks-McClellan, and so forth. In the following, we consider an alternative design strategy based on minimizing the variance of the fault estimate.

Let $h[n]$ be the impulse response of a high pass filter. Let us assume it has odd length $2K+1$, and that it is centered about the origin, $h[n] = 0$ for $|n| > K$. Let us assume that a fault occurred on the $q^{\underline{th}}$ converter, H_q, with unknown value $\phi_q[n]$, and that the faulty converter has been properly identified. As before, let us assume the quantization noise $\epsilon[n]$ is zero mean and white, with sample variance σ_ϵ^2. Unlike the previous sections, let us also assume that the signal is described as a wide sense stationary stochastic process with zero mean and covariance $R[n] = \mathrm{E}\left[s[j]s[n+j]\right]$. We assume that the signal and noise are uncorrelated, and that the signal power spectrum $P(\omega)$ (the Fourier transform of $R[n]$) is low pass, $P(\omega) = 0$ for $\omega \in \Omega_H$. Now suppose we generate a fault estimate $\hat{\phi}_q[n]$ by high pass filtering the observations,

$$\hat{\phi}_q[n] \quad = \quad \begin{cases} \sum_{l=-L}^{L} h[l]z[n-l] & \text{for } n \equiv q \\ 0 & \text{else} \end{cases}$$

$$= \begin{cases} \sum_{l=-K}^{K} h[l]\left(s[n-l] + \phi_q[n-l] + \epsilon[n-l]\right) & \text{for } n \equiv q \\ 0 & \text{else} \end{cases} \tag{7.81}$$

(Note that we have absorbed the scaling factor $\frac{N+C}{C}$ shown in (7.40) into $h[n]$.) Now for samples $n \equiv q$,

$$\mathrm{E}\left[\hat{\phi}_q[n] \middle| H_q, \underline{\phi}_q\right] = \sum_{l=-K}^{K} h[l]\phi_q[n-l]. \tag{7.82}$$

To eliminate any bias, design $h[n]$ so that

$$h\left[p(N+C)\right] = \begin{cases} 1 & \text{for } p = 0 \\ 0 & \text{for } p = \pm 1, \pm 2, \ldots \end{cases} \tag{7.83}$$

Note that this implies that a filter of length $l(N+C)-1$ will do just as well as a filter of length $l(N+C)+1$. A "natural" length for the FIR filter, therefore, is $2K+1 = l(N+C)-1$ for some l.

Subject to the above constraint, let us now choose the remaining coefficients of $h[n]$ in order to minimize the variance of the fault estimate. For $n \equiv q$:

$$\begin{aligned} \mathrm{Var}\left[\hat{\phi}_q[n] \middle| H_q, \underline{\phi}_q\right] &= \mathrm{Var}\left[\sum_{l=-K}^{K} h[l]s[n-l] + \sum_{l=-K}^{K} h[l]\epsilon[n-l] \middle| H_q, \underline{\phi}_q\right] \\ &= \sum_{l=-K}^{K} \sum_{p=-K}^{K} h[l]h[p]R[p-l] + \sum_{l=-K}^{K} h^2[l]\sigma_\epsilon^2 \\ &= \underline{h}^T\left(\mathbf{R} + \sigma_\epsilon^2 \mathbf{I}\right)\underline{h} \end{aligned} \tag{7.84}$$

where $\underline{h} = (h[-K] \cdots h[K])^T$ and $\mathbf{R}$ is a $(2K+1) \times (2K+1)$ Toeplitz correlation matrix with entries $\mathbf{R}_{n,m} = R[m-n]$. For convenience, number the rows and columns of $\underline{h}$ and $\mathbf{R}$ from $-K$ through $+K$. We now optimize this variance over all possible sets of filter coefficients $\underline{h}$ subject to the constraints in (7.83). Since the samples of $h[n]$ for $n \equiv 0$ are already specified, we will remove them from $\underline{h}$ and include them explicitly in the equation. Form the reduced vector $\tilde{\underline{h}}$ from $\underline{h}$ by removing rows $0, \pm(N+C), \pm 2(N+C), \ldots$. Similarly, form the reduced matrix $\tilde{\mathbf{R}}$ from $\mathbf{R}$ by removing rows $0, \pm(N+C), \pm 2(N+C), \ldots$ and columns $0, \pm(N+C), \pm 2(N+C), \ldots$. The variance then becomes,

$$= R[0] + \sigma_\epsilon^2 + 2\underline{\tilde{h}}^T \underline{\tilde{r}} + \underline{\tilde{h}}^T \left(\tilde{\mathbf{R}} + \sigma_\epsilon^2 \mathbf{I}\right) \underline{\tilde{h}} \tag{7.85}$$

where $\underline{\tilde{r}} = (R[-K] \cdots R[K])^T$ is column 0 of matrix $\mathbf{R}$ with rows $0, \pm(N+C), \pm 2(N+C), \ldots$ removed. Set the derivative with respect to $\underline{\tilde{h}}$ to zero to find the minimum variance filter:

$$\underline{\hat{\tilde{h}}} = -\left(\tilde{\mathbf{R}} + \sigma_\epsilon^2 \mathbf{I}\right)^{-1} \underline{\tilde{r}} \tag{7.86}$$

To obtain the optimal FIR high pass filter $\underline{\hat{h}}$, reinsert the samples specified by (7.83). This method is guaranteed to have a unique minimum solution since $\tilde{\mathbf{R}} + \sigma_\epsilon^2 \mathbf{I}$ is positive definite. Also, the optimal filter $\hat{h}[n]$ will have even symmetry because of the symmetry of $\tilde{\mathbf{R}} + \sigma_\epsilon^2 \mathbf{I}$ and $\underline{\tilde{r}}$.

A different γ threshold is needed for the FIR high pass than for the ideal high pass. Including contributions from both signal leakage and quantization error, equation (7.84) gives the variance of $\hat{\phi}_q[n]$, with the optimal filter $\underline{\hat{h}}$ substituted for $\underline{h}$. If we approximate samples $\hat{\phi}_q[n]$ and $\hat{\phi}_q[n + l(N+C)]$ as being uncorrelated for any $l \neq 0$ (this is strictly true only for an ideal high pass), then formula (7.55) for P_F continues to apply, but with:

$$\begin{aligned} \mathrm{E}\left[L_k' \middle| H_*\right] &= \frac{M}{N+C} \left[\underline{\hat{h}}^T \left(\mathbf{R} + \sigma_\epsilon^2 \mathbf{I}\right) \underline{\hat{h}}\right] \\ \mathrm{Var}\left[L_k' \middle| H_*\right] &= 2\frac{M}{N+C} \left[\underline{\hat{h}}^T \left(\mathbf{R} + \sigma_\epsilon^2 \mathbf{I}\right) \underline{\hat{h}}\right]^2 \end{aligned} \tag{7.87}$$

We reconstruct the signal as in (7.78), without using a low pass filter. We can then show that:

$$\begin{aligned} \mathrm{E}\left[\hat{s}_A[n] - s[n] \middle| H_q, \underline{\phi}_q\right] &= 0 \\ \mathrm{Var}\left[\hat{s}_A[n] - s[n] \middle| H_q, \underline{\phi}_q\right] &= \begin{cases} \underline{\tilde{r}}^T \left(\tilde{\mathbf{R}} + \sigma_\epsilon^2 \mathbf{I}\right)^{-1} \underline{\tilde{r}} & \text{for } n \equiv q \\ \sigma_\epsilon^2 & \text{else.} \end{cases} \end{aligned} \tag{7.88}$$

A particularly useful case is where the signal power spectrum is low pass and flat in the pass band, $P(\omega) = \sigma_s^2$ for $\omega \in \Omega_L$ and $P(\omega) = 0$ for $\omega \in \Omega_H$. (This is the signal model used

		Filter			
		A 23 points	B 31 points	C 63 points	Ideal ∞ points
$\mathrm{Var}[\hat{s}_A[n]]$		$1.39\sigma_\epsilon^2$	$1.22\sigma_\epsilon^2$	$1.15\sigma_\epsilon^2$	$1.08\sigma_\epsilon^2$
Calculated γ		$968\sigma_\epsilon^2$	$716\sigma_\epsilon^2$	$612\sigma_\epsilon^2$	$505\sigma_\epsilon^2$
Measured P_F		2.5×10^{-4}	2.2×10^{-4}	9.6×10^{-5}	1×10^{-4}
1 bit faults	$1 - P_D$	5.89×10^{-2}	2.80×10^{-3}	3.8×10^{-4}	10^{-5}
	P_M	1.58×10^{-1}	5.42×10^{-2}	6.39×10^{-3}	5×10^{-4}
2 bit faults	$1 - P_D$	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$
	P_M	4.3×10^{-4}	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$
3 bit faults	$1 - P_D$	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$
	P_M	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$

Table 7.1: Results for the three optimal filters tested.

in all our simulations.) The covariance of $s[n]$ is then

$$R[n] = \sigma_s^2 \frac{\sin\left(\frac{\pi N}{N+C} n\right)}{\pi n}. \tag{7.89}$$

where $\sigma_s^2 = 2^{2B}\sigma_\epsilon^2$ is the variance of a signal filling the available dynamic range of B bits. Using the same system arrangement described earlier with $N = 5$, $C = 3$, and $B = 12$, we computed the optimal filter solution (7.86) for lengths 23, 31, and 63 samples respectively. Graphs of the frequency responses for the filters are shown in Figure 7-5.

Once again, we collected $M = 1024$ samples so that each likelihood was formed from $M/(N + C) = 128$ terms. 10^5 independent trials were performed, and we used (7.55) with (7.59) and (7.87) to set the threshold γ. We found that a value of $\beta = 3.85$ in (7.59) gave a false alarm probability of $P_F \approx 10^{-4}$. Results for the three filters tested are shown in Table 7.1.

All filters tested were able to accurately detect and diagnosis a faulty converter. The longer the filter, the more accurate its estimate $\hat{\phi}_k[n]$, and thus the better its performance. Although none of the FIR filters were as sensitive as the ideal filter in detecting errors in the least significant bit, larger faults could be detected and diagnosed accurately. The variances of the signal estimates are also shown in Table 7.1. The variances are quite low,

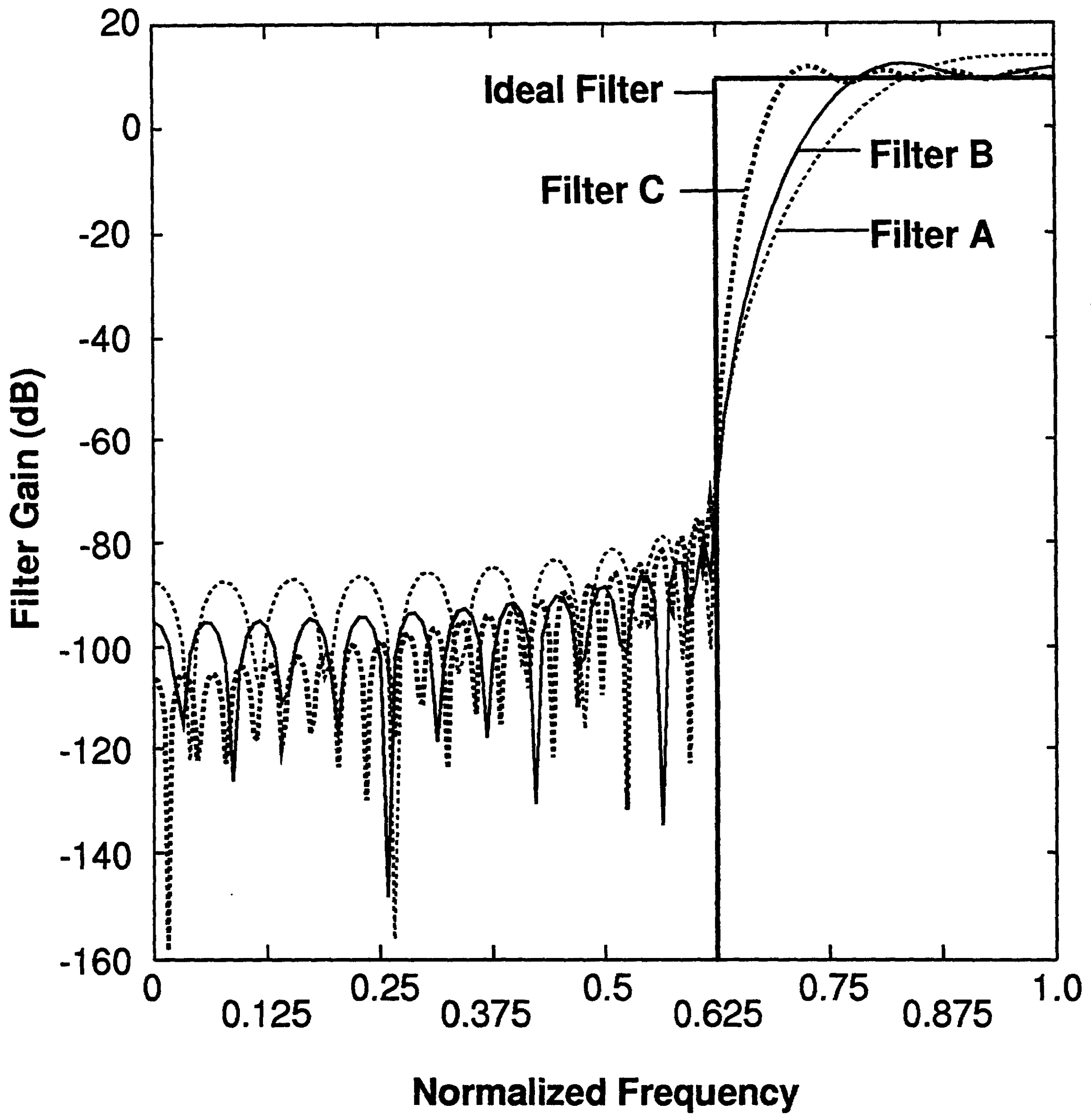


Figure 7-5: Graphs of the frequency responses of filters tested. A:23 pt., B:31 pt., C:63 pt., optimal filters. D:ideal filter.

and their values match match those predicted earlier. The false alarms which occur do not significantly raise the average signal estimate variance for these filter lengths.

7.5.4 Finite Order Integrators

The exact likelihood algorithm computes the likelihoods by integrating the energy in the fault estimates over all time, and then makes a single fault diagnosis for the entire batch of data. It would be useful to develop a real-time scheme which makes fault diagnoses and corrections based only on the most recently received data. One change needed is to compute the likelihoods using finite order integrators: we consider rectangular windows and single pole IIR filters. At time n, we use the high pass filter output $\hat{\phi}_k[n]$ to update the k^{th} converter's likelihood $L'[n]$ where $n \equiv k$:

$$\begin{aligned} &\text{Rectangular:} \quad L'[n] = \sum_{l=0}^{K_{int}-1} \hat{\phi}_k^2[n - l(N+C)] \\ &\text{IIR update:} \quad L'[n] = \alpha L'[n-(N+C)] + \hat{\phi}_k^2[n] = \sum_{l=0}^{\infty} \alpha^l \hat{\phi}_k[n - l(N+C)] \end{aligned} \qquad (7.90)$$

where K_{int} is the length of the rectangular window integrator, and $0 < \alpha < 1$ is the decay rate of the IIR integrator. The IIR approach has the advantage of requiring only $\mathbf{O}(1)$ storage, while the rectangular filter requires $\mathbf{O}(K_{int})$ storage. Appendix 7.C shows that for long integration windows (i.e. large K_{int} or $\alpha \approx 1$), our formulas for P_F, P_D, and P_M are still valid, but the likelihood means and variances are different. The expected values of the likelihoods have the same forms as in (7.58), (7.64), (7.67), and (7.87), except that the factor $M/(N+C)$ is replaced by K_{int} for the rectangular window likelihood, and by $1/(1-\alpha)$ for the IIR likelihood. Similarly, the variances of the likelihoods have the same forms as in (7.58), (7.64), (7.68), and (7.87), except that the factor $M/(N+C)$ is replaced by K_{int} for the rectangular window likelihood, and by $1/(1-\alpha^2)$ for the IIR likelihood. Careful study of the formulas suggests that similar performance for these two integrators should result if we choose $K_{int} = (1+\alpha)/(1-\alpha)$, and pick thresholds $\gamma_{rect} = \gamma_{IIR}(1+\alpha)$.

We ran the same simulation as in the previous section to compare the performance of several rectangular integrators and IIR integrators, using the 31 point FIR high pass filter B described earlier. The thresholds γ were chosen using (7.55), (7.59), and (7.87) to achieve

		Integration Length K_{int} (samples)		
		128	64	32
Calculated γ		$716\sigma_\epsilon^2$	$406\sigma_\epsilon^2$	$237\sigma_\epsilon^2$
Measured P_F		2.2×10^{-4}	1.21×10^{-3}	1.89×10^{-3}
1 bit faults	$1 - P_D$	2.80×10^{-3}	1.00×10^{-1}	3.96×10^{-1}
	P_M	5.40×10^{-2}	1.82×10^{-1}	3.48×10^{-1}
2 bit faults	$1 - P_D$	$< 10^{-5}$	$< 10^{-5}$	2×10^{-5}
	P_M	$< 10^{-5}$	1.58×10^{-3}	2.25×10^{-2}
3 bit faults	$1 - P_D$	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$
	P_M	$< 10^{-5}$	$< 10^{-5}$	2×10^{-5}
4 bit faults	$1 - P_D$	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$
	P_M	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$

Table 7.2: Results for filter B using rectangular windows.

$P_F \approx 10^{-4}$. The decays α and thresholds γ_{IIR} of the IIR integrators were chosen to match the performance of the rectangular integrators. The results for rectangular windows are shown in Table 7.2, and those for IIR integrators are shown in Table 7.3. As expected, the rectangular and IIR integrators have similar performance, and the results shown in the tables behave in the expected way. Although the observed P_F is much larger than predicted, especially for short integration lengths and IIR integrators, raising γ by 10-20% would drop the observed P_F down to about 10^{-4}. Performance is much worse than expected for integration lengths shorter than those shown in the tables; the reason for this is discussed in the next section.

7.5.5 Real-Time Fault Detection Algorithm

The system sketched earlier is batch oriented; it collects all the outputs of the high pass filter, sums up the likelihoods for the entire batch, then makes one fault decision for all the data. Several modifications are necessary to achieve a causal, real-time fault tolerant A/D system capable of responding correctly to both transient and continuous faults. The simplest approach would update one likelihood with each new high pass filter output, then if this likelihood is greater than γ and also greater than all the other likelihoods, a fault would be declared in the corresponding converter. Unfortunately, this strategy doesn't work

		Effective Integration Length (samples)		
		128	64	32
α		0.9845	0.9692	0.9394
Calculated γ		$361\sigma_\epsilon^2$	$206\sigma_\epsilon^2$	$122\sigma_\epsilon^2$
Measured P_F		2.13×10^{-3}	2.13×10^{-3}	4.07×10^{-3}
1 bit	$1 - P_D$	1.95×10^{-3}	9.88×10^{-2}	3.97×10^{-1}
faults	P_M	6.63×10^{-2}	2.06×10^{-1}	3.66×10^{-1}
2 bit	$1 - P_D$	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$
faults	P_M	$< 10^{-5}$	2.18×10^{-3}	2.75×10^{-2}
3 bit	$1 - P_D$	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$
faults	P_M	$< 10^{-5}$	$< 10^{-5}$	3×10^{-5}
4 bit	$1 - P_D$	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$
faults	P_M	$< 10^{-5}$	$< 10^{-5}$	$< 10^{-5}$

Table 7.3: Simulation results for filter B using IIR integrators.

properly for transient failures. Figure 7-6 illustrates the problem, showing various failures $\phi_0[n]$ on converter 0, together with the sequence of likelihoods $L'[n]$ generated by the IIR update formula (7.90). The dotted lines on the graph identify the samples corresponding to converter 0.

For a continuous failure, the correct likelihood $L'[n]$ for $n \equiv 0$ is always larger than any of the other likelihoods. For a single sample failure on converter 0 at time n_0, however, the output of the high pass filter starts rising at time $n_0 - K$ and continues oscillating until time $n_0 + K$. In fact, because $h[n] = 0$ for n equal to multiples of $N + C$, the likelihoods corresponding to converter 0 are unaffected by the failure until time n_0. Thus for K samples before the failure, all the likelihoods except the correct one start building in energy and may cross the threshold γ. This implies that a fault decision algorithm may have to wait up to K samples from the time the first likelihood rises above threshold before it decides which converter is at fault.

Another problem shows up with faults whose amplitude increases rapidly over time. The figure illustrates a fault whose amplitude rises at an exponential rate. Notice that the correct likelihood is consistently smaller than some of the other likelihoods. Eventually, when the fault amplitude stops growing, the correct likelihood will become largest, but this may take much more than K samples from the time the first likelihood crosses threshold.

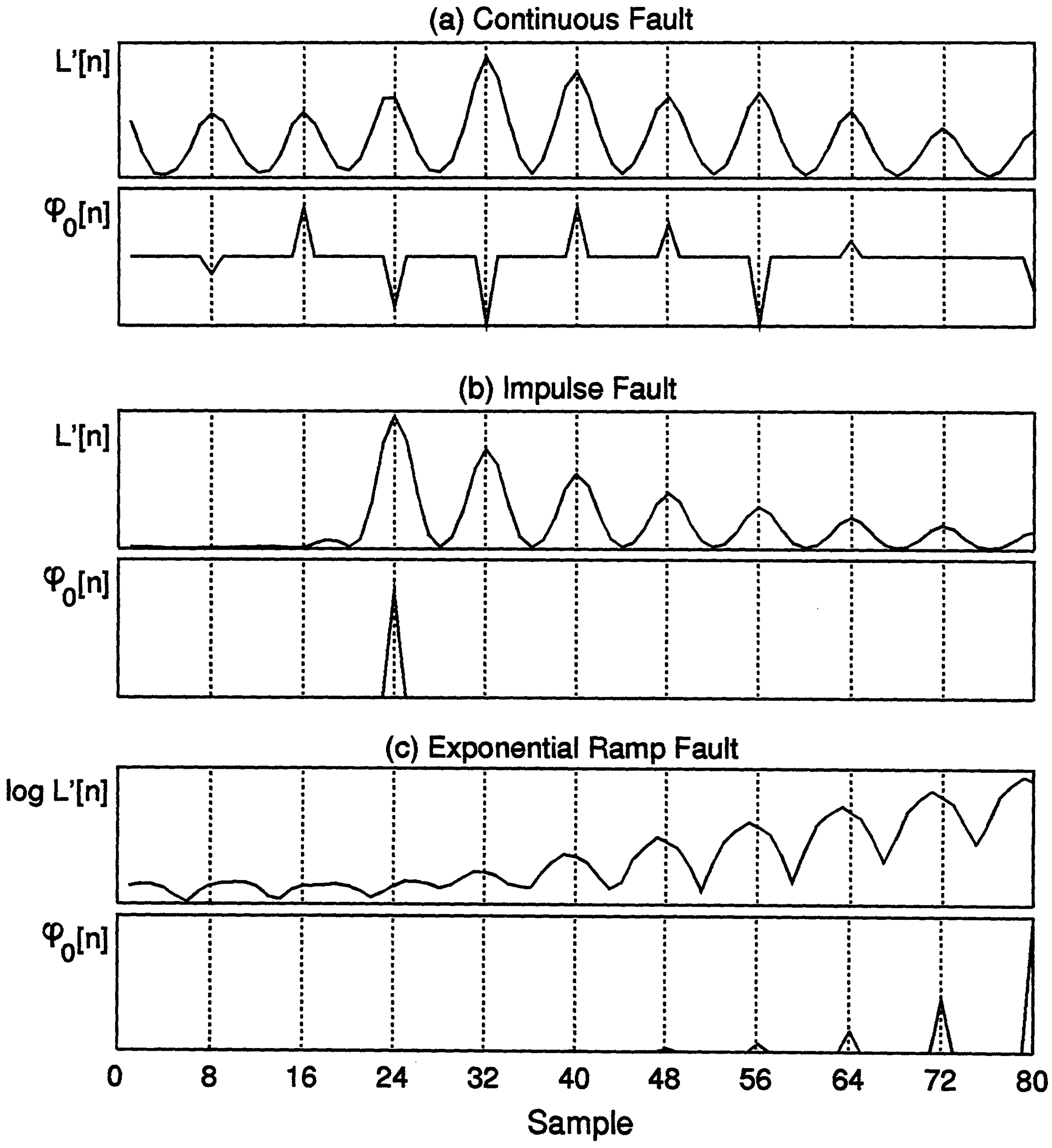


Figure 7-6: IIR Likelihoods $L'[n]$ versus fault $\phi_0[n]$: (a) Continuous fault (b) impulse fault (c) exponential ramp fault.

It seems difficult to design a single decision algorithm which produces the correct fault diagnoses in all these cases. One possible procedure might use a three state approach. At time n, the incoming sample $z[n]$ is processed through the high pass filter, and the output of the filter $\hat{\phi}_k[n-K]$ is used to update likelihood $L'[n-K]$. Initially the detector starts in state "OK", and it remains there as long as all likelihood values are below the threshold γ. If one goes above threshold, go to state "detected" and wait for at least K more samples. Then pick the largest likelihood from the last $N+C$, and if it is greater than threshold go to state "corrected" and declare that the corresponding converter has failed. Also start correcting the samples from that converter. Note that because we do not identify a failure until at least $2K$ samples after it has occurred, we will need to save at least the most recent $2K$ converter samples in a buffer so that these can be corrected in case we enter the "correction" state. This imposes a minimum latency of $2K$ in our fault correction system.

Further work is needed to refine this algorithm to incorporate realistic fault scenarios. If a small continuous fault starts, it may take awhile for the likelihood energies to build up enough to cross threshold, thus delaying the correction. If exponential ramp failures can occur, then it may be necessary to change which converter is identified as faulty while in the "correction" state. Rules on when to exit the "correction" state must be developed. It will also be necessary to compute the probabilities of false alarm, detection, and misdiagnosis for whatever algorithm is eventually chosen.

Choice of the threshold γ is complicated by the fact that P_D depends on the total energy in the fault, as reflected by the fault-to-noise ratio FNR. A continuous fault of low amplitude can give rise to a large FNR, since the likelihood integrates the fault energy over a large number of samples. A transient fault which affects only one sample, however, must be quite large to result in a comparable FNR. Setting a low threshold γ and using short integration lengths allows smaller transient errors to be detected and corrected. However, low γ leads to high probability of false alarm, and short integration lengths causes the likelihoods to have high variance, which causes high probability of misdiagnosis. Choosing the best parameters in the algorithm, therefore, involves some delicate performance tradeoffs.

7.6 Conclusion

In this chapter we described a robust oversampling round-robin A/D converter system which uses the redundancy inherent to low-pass signals to provide fault tolerance. The system was able to identify converter failures reliably and to correct the output accurately. The hardware needed to add robustness is minimal: a few extra converters and an amount of computation comparable to an FIR filter. A disadvantage of our approach is that we rely on a statistical test to detect and correct faults, and therefore have certain probabilities of missing or misdiagnosing a fault, or of declaring a fault where none exists. More fundamental concerns relate to our use of round-robin scheduling of multiple slow A/D converters, a technique which requires careful attention to calibration, sample/holds, and timing issues. Despite these potential problems, our approach is considerably cheaper than traditional approaches to fault tolerance such as modular redundancy, yet can achieve comparable protection against single faults.

This chapter explored A/D conversion, but it can be easily seen that the results may be extended to the design of fault-tolerant D/A converter systems. A/D and D/A converter systems are conceptually isomorphic mappings, and since we are able to protect A/D conversion, we must be able to protect D/A conversion as well. In fact, the same redundancy requirements would hold: we would need 1 extra converter to detect errors, and at least 2 extra converters to correct errors. Also, error detection and correction would be based on a syndrome computed using an ideal analog high-pass filter. The only remaining work is to derive the exact form of the error detection and correction procedures, and to determine an efficient implementation. The proper approach would be to explicitly take into account the effects of quantization noise and use a generalized likelihood ratio test to determine the optimal solution.

7.A Signal Estimates: Mean and Variance

Assume that fault H_q has occurred with value $\phi_q[n]$. From (7.40) we know that $\hat{\phi}_k[n]$ is formed from a high pass version of $z[n]$. We can expand this filtering operation as a convolution sum,

$$\hat{\phi}_k[n] = \left(\frac{N+C}{C} \sum_{l=0}^{M-1} h_{HP}[l] z[n-l] \right) u_k[n] \tag{7.91}$$

where we define

$$u_k[n] = \begin{cases} 1 & \text{for } n \equiv k \\ 0 & \text{else} \end{cases} \tag{7.92}$$

and where

$$h_{HP}[n] = \frac{(-1)^n \sin\left(\frac{\pi C}{N+C} n\right)}{M \sin\left(\frac{\pi}{M} n\right)} \tag{7.93}$$

is the impulse response of $H_{HP}(\omega_r)$. Substitute (7.14) into (7.91), and since $s[n]$ is a low pass signal, we are left with

$$\hat{\phi}_k[n] = \left(\overline{\phi}_q[n] + \overline{\epsilon}[n] \right) u_k[n] \tag{7.94}$$

where $\overline{\phi}_q[n]$ and $\overline{\epsilon}[n]$ are high pass versions of $\phi_q[n]$ and $\epsilon[n]$:

$$\begin{aligned} \overline{\phi}_q[n] &= \frac{N+C}{C} \sum_{l=0}^{M-1} h_{HP}[l] \phi_q[n-l] \\ \overline{\epsilon}[n] &= \frac{N+C}{C} \sum_{l=0}^{M-1} h_{HP}[l] \epsilon[n-l] \end{aligned} \tag{7.95}$$

It is important to recognize that $h_{HP}[n]$ contains zeros at all nonzero multiples of $N+C$ samples. Therefore, $\overline{\phi}_q[n] = \phi_q[n]$ for $n \equiv q$. Also, $\epsilon[n]$ is white Gaussian noise with mean and variance

$$\mathrm{E}[\epsilon[l]] = 0 \tag{7.96}$$

$$\mathrm{E}[\epsilon[l]\epsilon[r]] = \sigma_\epsilon^2 \delta[l-r] \tag{7.97}$$

Thus:

$$\begin{aligned} \mathrm{E}[\overline{\epsilon}[n]] &= 0 \\ \mathrm{Cov}[\overline{\epsilon}[n], \overline{\epsilon}[m]] &= \left(\frac{N+C}{C} \right)^2 \sum_{l=0}^{M-1} h_{HP}[n-l] h_{HP}[m-l] \sigma_\epsilon^2 \end{aligned}$$

$$= \left(\frac{N+C}{C}\right)^2 h_{HP}[n-m]\sigma_\epsilon^2 \tag{7.98}$$

where the last line follows because $h_{HP}[n] = h_{HP}[-n]$ and $h_{HP}[n] * h_{HP}[n] = h_{HP}[n]$, where '$*$' denotes circular convolution with period M. These equations imply that

$$\mathrm{E}\left[\hat{\phi}_k[n] \middle| H_q, \underline{\phi}_q\right] = \overline{\phi}_q[n] u_k[n] \tag{7.99}$$

and

$$\begin{aligned} \mathrm{Cov}\left[\hat{\phi}_{k_1}[n], \hat{\phi}_{k_2}[m] \middle| H_q, \underline{\phi}_q\right] &= \mathrm{Cov}\left[\overline{\epsilon}[n], \overline{\epsilon}[m] | H_q, \underline{\phi}_q\right] u_{k_1}[n] u_{k_2}[m] \\ &= \left(\frac{N+C}{C}\right)^2 h_{HP}[n-m]\sigma_\epsilon^2 u_{k_1}[n] u_{k_2}[m] \end{aligned} \tag{7.100}$$

Now for the mean and variance of the approximate, unfiltered signal estimate, which is formed by subtracting the fault estimate from the observations, (7.78). Assume that the fault H_q has been diagnosed properly. Then

$$\begin{aligned} \hat{s}_A[n] &= z[n] - \hat{\phi}_q[n] \\ &= s[n] + \phi_q[n] + \epsilon[n] - \left(\overline{\phi}_q[n] + \overline{\epsilon}[n]\right) u_q[n] \\ &= s[n] + \epsilon[n] - \overline{\epsilon}[n] u_q[n] \end{aligned} \tag{7.101}$$

Thus:

$$\mathrm{E}\left[\hat{s}_A[n] | H_q, \underline{\phi}_q\right] = s[n] \tag{7.102}$$

and

$$\begin{aligned} \mathrm{Cov}\left[\hat{s}_A[n], \hat{s}_A[m] | H_q, \underline{\phi}_q\right] &= \mathrm{E}\left[(\epsilon[n] - \overline{\epsilon}[n]u_q[n])(\epsilon[m] - \overline{\epsilon}[m]u_q[m]) | H_q, \underline{\phi}_q\right] \\ = \sigma_\epsilon^2 \Bigg\{ \delta[n-m] &- \left(\frac{N+C}{C}\right) h_{HP}[n-m]u_q[n] - \left(\frac{N+C}{C}\right) h_{HP}[m-n]u_q[m] \\ &+ \left(\frac{N+C}{C}\right)^2 h_{HP}[n-m]u_q[n]u_q[m] \Bigg\} \end{aligned} \tag{7.103}$$

In particular, since $h_{HP}[0] = C/(N+C)$,

$$\mathrm{Var}\left[\hat{s}_A[n] |\, H_q, \underline{\phi}_q\right] = \begin{cases} \frac{N}{C}\sigma_\epsilon^2 & \text{for } n \equiv q \\ \sigma_\epsilon^2 & \text{else} \end{cases} \tag{7.104}$$

From this we see that all samples of $\hat{s}_A[n]$ have the proper average value and that the corrected samples have $\frac{N}{C}$ times as much variance as the samples from the working converters. Also, the average variance in $\hat{s}_A[n]$ is,

$$\frac{1}{M}\sum_{n=0}^{M-1} \mathrm{Var}\left[\hat{s}_A[n] |\, H_k, \underline{\phi}_k\right] = \left[1 + \frac{N-C}{C(N+C)}\right]\sigma_\epsilon^2. \tag{7.105}$$

The ideal estimate under hypothesis H_q is formed by low pass filtering $\hat{s}_A[n]$,

$$\hat{s}[n] = \sum_{r=0}^{M-1} h_{LP}[n-r]\hat{s}_A[r] \tag{7.106}$$

Substituting (7.101) into this equation and recalling that $s[n]$ is low pass, yields,

$$\hat{s}[n] = s[n] + \sum_{r=0}^{M-1} h_{LP}[n-r]\left(\epsilon[r] - \bar{\epsilon}[r]u_q[r]\right). \tag{7.107}$$

Thus,

$$\mathrm{E}\left[\hat{s}[n] \mid H_q, \underline{\phi}_q\right] = s[n] \tag{7.108}$$

and we can compute the variance as follows,

$$\mathrm{Var}\left[\hat{s}[n] \mid H_q, \underline{\phi}_q\right] = \sum_{r=0}^{M-1}\sum_{t=0}^{M-1} h_{LP}[n-r]h_{LP}[n-t]\mathrm{Cov}\left[\hat{s}_A[r], \hat{s}_A[t] |\, H_q, \underline{\phi}_q\right] \tag{7.109}$$

$$= \sum_{r=0}^{M-1} h_{LP}^2[n-r]\sigma_\epsilon^2 - 2\left(\frac{N+C}{C}\right)\sum_{r\equiv q}\sum_{t=0}^{M-1} h_{LP}[n-r]h_{LP}[n-t]h_{HP}[r-t]\sigma_\epsilon^2$$
$$+ \left(\frac{N+C}{C}\right)^2 \sum_{r\equiv q}\sum_{t\equiv q} h_{LP}[n-r]h_{LP}[n-t]h_{HP}[r-t]\sigma_\epsilon^2.$$

This can be simplified in a few steps. Note that $h_{LP}[n] * h_{HP}[n] = 0$, and also:

$$h_{HP}[n(N+C)] = \begin{cases} \frac{C}{N+C} & \text{for } n = 0 \\ 0 & \text{else.} \end{cases} \tag{7.110}$$

Thus,

$$\mathrm{Var}\left[\hat{s}[n] \mid H_q, \underline{\phi}_q\right] = \sum_{r=0}^{M-1} h_{LP}^2[n-r]\sigma_\epsilon^2 + \left(\frac{N+C}{C}\right) \sum_{r \equiv q} h_{LP}^2[n-r]\sigma_\epsilon^2. \tag{7.111}$$

After a little more algebra, we find that

$$\mathrm{Var}\left[\hat{s}[n] \mid H_q, \underline{\phi}_q\right] = \begin{cases} \frac{N}{C}\sigma_\epsilon^2 & \text{for } n \equiv q \\ \frac{N}{N+C}\sigma_\epsilon^2 + \frac{C}{N+C}S^2(n-q)\sigma_\epsilon^2 & \text{else} \end{cases} \tag{7.112}$$

where $S(l)$ is the circular sinc function with period $N+C$ in (7.69). The variance is thus highest for those signal samples, $n \equiv q$, from the faulty converter which have been interpolated from the other converter samples, $\frac{N}{C}\sigma_\epsilon^2$. All other signal samples have variance below σ_ϵ^2, with lowest variance occurring for signal samples from converters far from the faulty one, about $\frac{N\sigma_\epsilon^2}{N+C}$. In fact, we can show that signal estimate samples for the faulty converter are unaffected by the low pass filtering operation. To see this, note that $H_{LP}(\omega_r) = 1 - H_{HP}(\omega_r)$, and thus $h_{LP}[n] = \delta[n] - h_{HP}[n]$. Substituting gives:

$$\begin{aligned} \hat{s}[n] &= s_A[n] - \sum_{l=0}^{M-1} h_{HP}[n-l]s_A[l] \\ &= s_A[n] - \sum_{l=0}^{M-1} h_{HP}[n-l]\left(z[l] - \hat{\phi}_q[l]\right) \end{aligned} \tag{7.113}$$

For $n \equiv q$:

$$\begin{aligned} \hat{s}[n] &= s_A[n] - \left(z_H[n] - \sum_{l \equiv q} h_{HP}[n-l]\hat{\phi}_q[l]\right) \\ &= s_A[n] - \left(z_H[n] - h_{HP}[0]\left(\frac{N+C}{C}\right) z_H[n]\right) \\ &= s_A[n] \end{aligned} \tag{7.114}$$

The average variance of the ideal estimator is,

$$\begin{aligned}
\frac{1}{M}\sum_{n=0}^{M-1} \operatorname{Var}\left[\hat{s}[n] \mid H_q, \underline{\phi}_q\right] &= \frac{N}{N+C}\sigma_\epsilon^2 + \left(\frac{N+C}{C}\right)\sum_{r\equiv q}\frac{1}{M}\sum_{n=0}^{M-1} h_{LP}^2[n-r]\sigma_\epsilon^2 \\
&= \frac{N(C+1)}{(N+C)C}\sigma_\epsilon^2. \qquad (7.115)
\end{aligned}$$

The statistics for $\hat{s}_A[n]$ and $\hat{s}[n]$ can also be derived for the case when all converters are working properly and hypothesis H_* is chosen. The mean and variance of the unfiltered signal estimator are:

$$\mathrm{E}\left[\hat{s}_A[n] \mid H_*\right] = s[n] \qquad (7.116)$$

$$\operatorname{Var}\left[\hat{s}_A[n] \mid H_*\right] = \sigma_\epsilon^2. \qquad (7.117)$$

The mean and variance of the filtered signal estimator are:

$$\mathrm{E}\left[\hat{s}[n] \mid H_*\right] = s[n] \qquad (7.118)$$

$$\operatorname{Var}\left[\hat{s}[n] \mid H_*\right] = \sum_{l=0}^{M-1} h_{LP}^2[n-l]\sigma_\epsilon^2 = \frac{N}{N+C}\sigma_\epsilon^2. \qquad (7.119)$$

7.B Ideal Likelihoods: Mean and Covariance

In this section we derive the formulas of section 7.3.5. First, define $\tau_l[n]$ as a shifted version of $h_{HP}[n]$ with all but every $(N+C)^{\underline{th}}$ sample set to zero:

$$\tau_l[n] = \begin{cases} h_{HP}[n+l] & \text{for } n \equiv 0 \\ 0 & \text{else.} \end{cases} \qquad (7.120)$$

Then

$$\tau_l[n] = h_{HP}[n+l]\frac{1}{N+C}\sum_{r=0}^{N+C-1} e^{-\frac{j2\pi r}{N+C}n} \qquad (7.121)$$

and Fourier transforming we obtain:

$$\begin{aligned} \tau_l(\omega_m) &= \sum_{n=0}^{M-1} \tau_l[n] e^{-j\omega_m n} && (7.122) \\ &= \frac{1}{N+C} \sum_{r=0}^{N+C-1} H_{HP}\left(\omega_m + \frac{2\pi r}{N+C}\right) e^{j\left(\omega_m + \frac{2\pi r}{N+C}\right)l}. && (7.123) \end{aligned}$$

Now $\tau_l(\omega_m)$ is periodic with period $2\pi/(N+C)$. For frequencies in the range $\pi N/(N+C) \leq \omega_m < \pi(N+2)/(N+C)$ we can calculate:

$$\begin{aligned} \tau_l(\omega_m) &= \frac{1}{N+C} \sum_{r=0}^{C-1} 1 \cdot e^{j\left(\omega_m + \frac{2\pi r}{N+C}\right)l} \\ &= \frac{C}{N+C} S(l) e^{j\left(\omega_m + \frac{\pi(C-1)}{N+C}\right)l} && (7.124) \end{aligned}$$

where $S(l)$ is the circular sinc function defined in (7.69). A formula valid for all ω_m can then be derived by exploiting the periodicity of $\tau_l(\omega_m)$:

$$\tau_l(\omega_m) = \frac{C}{N+C} S(l) e^{j\left(\omega_m + \frac{\pi(C-1-2p_m)}{N+C}\right)l} \qquad (7.125)$$

where p_m is the integer:

$$p_m = \left\lfloor m\left(\frac{N+C}{M}\right) - N/2 \right\rfloor \qquad (7.126)$$

and where $\lfloor x \rfloor$ represents the largest integer no greater than x. Note that the phase of $\tau_l(\omega_m)$ is a sawtooth ranging $\pm\frac{\pi l}{N+C}$ about πl in steps of $\frac{2\pi l}{M}$, while the magnitude is constant.

Now to compute the statistics of the L'_k. Assume that fault H_q has occurred with value $\phi_q[n]$. Combining (7.45) with (7.94), and using (7.98):

$$\begin{aligned} \mathrm{E}\left[L'_k \middle| H_q, \underline{\phi}_q\right] &= \sum_{n\equiv k} \mathrm{E}\left[\hat{\phi}_k^2[n] \middle| H_q, \underline{\phi}_q\right] \\ &= \sum_{n\equiv k} \overline{\phi}_q^2[n] + \sum_{n\equiv k} \mathrm{E}\left[\overline{\epsilon}^2[n] \middle| H_q, \underline{\phi}_q\right] \\ &= \sum_{n\equiv k} \overline{\phi}_q^2[n] + \left(\frac{M}{N+C}\right)\left(\frac{N+C}{C}\right)\sigma_\epsilon^2 && (7.127) \end{aligned}$$

The first term above can be evaluated by using (7.95), recognizing that $h_{HP}[n] = h_{HP}[-n]$, and substituting $\tau_{k-l}[n]$ for $h_{HP}[n+k-l]$ and $\tau_{l-p}[-n]$ for $h_{HP}[l-p-n]$:

$$\begin{aligned}
\sum_{n\equiv k} \overline{\phi}_q^2[n] &= \left(\frac{N+C}{C}\right)^2 \sum_{n\equiv k}\sum_{l\equiv q}\sum_{p\equiv q} h_{HP}[n-l]h_{HP}[n-p]\phi_q[l]\phi_q[p] \\
&= \left(\frac{N+C}{C}\right)^2 \sum_{l\equiv q}\sum_{p\equiv q} \phi_q[l]\phi_q[p]\left[\sum_{n\equiv 0} h_{HP}[k+n-l]h_{HP}[k+n-p]\right] \\
&= \left(\frac{N+C}{C}\right)^2 \sum_{l\equiv q}\sum_{p\equiv q} \phi_q[l]\phi_q[p]\left[\sum_{n=0}^{M-1} \tau_{k-l}[n]\tau_{p-k}[-n]\right] \qquad (7.128)
\end{aligned}$$

Apply Parseval's theorem and work in the frequency domain:

$$\begin{aligned}
\sum_{n\equiv k} \overline{\phi}_q^2[n] &= \left(\frac{N+C}{C}\right)^2 \sum_{l\equiv q}\sum_{p\equiv q} \phi_q[l]\phi_q[p]\left[\frac{1}{M}\sum_{m=0}^{M-1} \tau_{k-l}(\omega_m)\tau_{p-k}(\omega_m)\right] \qquad (7.129) \\
&= \sum_{l\equiv q}\sum_{p\equiv q} \phi_q[l]\phi_q[p]S(k-l)S(p-k)\left[\frac{1}{M}\sum_{m=0}^{M-1} e^{j\left(\omega_m+\frac{\pi(C-1-2pm)}{N+C}\right)(p-l)}\right]
\end{aligned}$$

Recall that the phase of the exponential in (7.129) is a sawtooth with range $\pm\pi(p-l)/(N+C)$. Therefore:

$$\frac{1}{M}\sum_{m=0}^{M-1} e^{j\left(\omega_m+\frac{\pi(C-1-2pm)}{N+C}\right)(p-l)} = \begin{cases} 1 & \text{if } p=l \\ 0 & \text{if } p-l=\pm(N+C), \pm 2(N+C), \ldots \end{cases} \qquad (7.130)$$

Also recognizing that $S(l)$ is symmetric and periodic with period $N+C$, equation (7.129) reduces to,

$$\sum_{n\equiv k} \overline{\phi}_q^2[n] = S^2(q-k)\sum_{l\equiv q} \phi_q^2[l] \qquad (7.131)$$

Combining (7.131) and (7.127) and using some algebra gives (7.64). For the no fault case, H_*, set $\text{FNR}_q = 0$ to get (7.58).

We now turn our attention to the covariance of two likelihoods, L'_{k_1} and L'_{k_2}, under hypothesis H_q and fault $\phi_q[n]$. Substituting (7.45) gives:

$$\begin{aligned}
\text{Cov}\left[L'_{k_1}, L'_{k_2} \,\middle|\, H_q, \underline{\phi}_q\right] &= \sum_{n_1\equiv k}\sum_{n_2\equiv k} \text{Cov}\left[\hat{\phi}_{k_1}^2[n_1], \hat{\phi}_{k_2}^2[n_2] \,\middle|\, H_q, \underline{\phi}_q\right] \\
&= \sum_{n_1\equiv k}\sum_{n_2\equiv k} \text{Cov}\left[\left(\overline{\phi}_q[n_1]+\overline{\epsilon}[n_1]\right)^2, \left(\overline{\phi}_q[n_2]+\overline{\epsilon}[n_2]\right)^2 \,\middle|\, H_q, \underline{\phi}_q\right] \qquad (7.132)
\end{aligned}$$

Now suppose a, b, c, and d are zero mean Gaussian random variables. Then it is well-known that $E[abc] = 0$ and

$$E[abcd] = E[ab]\,E[cd] + E[ac]\,E[bd] + E[ad]\,E[bc] \tag{7.133}$$

Using this in (7.132), plus the fact that $\operatorname{Cov}[a,b] = E[ab] - E[a]E[b]$, expanding terms and applying a lot of algebra gives:

$$\begin{aligned}\operatorname{Cov}\left[L'_{k_1}, L'_{k_2} \,\middle|\, H_q, \underline{\phi}_q\right] &= \sum_{n_1 \equiv k}\sum_{n_2 \equiv k} \left\{ 2\left(\frac{N+C}{C}\right)^4 h_{HP}^2[n_1 - n_2]\sigma_\epsilon^4 \right. \\ &\quad \left. + 4\overline{\phi}_q[n_1]\overline{\phi}_q[n_2]\left(\frac{N+C}{C}\right)^2 h_{HP}[n_1 - n_2]\sigma_\epsilon^2 \right\}\end{aligned} \tag{7.134}$$

Substituting (7.95) gives:

$$\begin{aligned} &= \left(\frac{N+C}{C}\right)^4 \sum_{n_1 \equiv k_1}\sum_{n_2 \equiv k_2} \left\{ h_{HP}^2[n_1 - n_2]\, 2\sigma_\epsilon^4 \right. \\ &\quad \left. + \sum_{l_1 \equiv q}\sum_{l_2 \equiv q} h_{HP}[n_1 - l_1]\, h_{HP}[n_2 - l_2]\, h_{HP}[n_1 - n_2]\, 4\sigma_\epsilon^2 \phi_q[l_1]\,\phi_q[l_2] \right\}\end{aligned} \tag{7.135}$$

Now substitute (7.120) for $h_{HP}[n]$ and adjust the summations,

$$\begin{aligned} &= \left(\frac{N+C}{C}\right)^4 \sum_{n_1 \equiv 0}\sum_{n_2 \equiv 0} \tau_{k_1-k_2}^2[n_1 - n_2]\, 2\sigma_\epsilon^4 \\ &\quad + \left(\frac{N+C}{C}\right)^4 \sum_{l_1 \equiv 0}\sum_{l_2 \equiv 0} \left[\sum_{n_1 \equiv 0}\sum_{n_2 \equiv 0} \tau_{q-k_1}[l_1 - n_1]\, \tau_{k_1-k_2}[n_1 - n_2]\, \tau_{k_2-q}[n_2 - l_2] \right] \\ &\quad \cdot \phi_q[l_1 + q]\,\phi_q[l_2 + q]\, 4\sigma_\epsilon^2. \end{aligned} \tag{7.136}$$

Because $\tau_l[n]$ is periodic with period M, we can further reduce the first term of the above equation,

$$\begin{aligned}
&= \left(\frac{N+C}{C}\right)^4 \left(\frac{M}{N+C}\right) \sum_{n\equiv 0} \tau_{k_1-k_2}^2[n] 2\sigma_\epsilon^4 \\
&\quad + \left(\frac{N+C}{C}\right)^4 \sum_{l_1\equiv 0} \sum_{l_2\equiv 0} \left[\sum_{n_1\equiv 0} \sum_{n_2\equiv 0} \tau_{q-k_1}[l_1-n_1]\, \tau_{k_1-k_2}[n_1-n_2]\, \tau_{k_2-q}[n_2-l_2] \right] \\
&\qquad \cdot \phi_q[l_1+q]\, \phi_q[l_2+q]\, 4\sigma_\epsilon^2
\end{aligned} \tag{7.137}$$

$$\begin{aligned}
&= \left(\frac{N+C}{C}\right)^4 \left(\frac{M}{N+C}\right) \frac{1}{M} \sum_{m=0}^{M-1} \left|\tau_{k_1-k_2}(\omega_m)\right|^2 2\sigma_\epsilon^4 \\
&\quad + \left(\frac{N+C}{C}\right)^4 \sum_{l_1\equiv 0} \sum_{l_2\equiv 0} \left[\frac{1}{M} \sum_{m=0}^{M-1} \tau_{q-k_1}(\omega_m)\, \tau_{k_1-k_2}(\omega_m)\, \tau_{k_2-q}(\omega_m)\, e^{j\omega_m(l_1-l_2)} \right] \\
&\qquad \cdot \phi_q[l_1+q]\, \phi_q[l_2+q]\, 4\sigma_\epsilon^2
\end{aligned} \tag{7.138}$$

$$\begin{aligned}
&\operatorname{Cov}\left[L'_{k_1}, L'_{k_2} \,\middle|\, H_q, \underline{\phi}_q\right] \qquad (7.139)\\
&= \frac{M}{C}\sigma_\epsilon^4 \left\{ 2\left(\frac{N+C}{C}\right) S^2(k_1-k_2) + 4S(q-k_1)\, S(k_1-k_2)\, S(k_2-q)\, \mathrm{FNR}_q \right\}
\end{aligned}$$

where $S(l)$ is the circular sinc function defined in (7.69), and FNR_q is the fault-to-quantization noise ratio defined in (7.65). Formula (7.64) results by substituting $k_1 = k_2 = q$ and noting that $S(0) = 1$. For the no fault case, H_*, formula (7.58) results by setting $\mathrm{FNR}_q = 0$.

The probabilities P_F, P_D, and P_M follow from Gaussian statistics. For any Gaussian variable $p(x) = N(m, \sigma^2)$,

$$\begin{aligned}
P(x \geq \gamma) &= \frac{1}{\sqrt{2\pi\sigma^2}} \int_\gamma^\infty \exp\left(-\frac{1}{2}\frac{(x-m)^2}{\sigma^2}\right) dx \\
&= \frac{1}{2}\left[\frac{2}{\sqrt{\pi}} \int_{\frac{\gamma-m}{\sqrt{2}\sigma}}^\infty \exp\left(-x^2\right) dx\right] \\
&= \frac{1}{2}\operatorname{erfc}\left(\frac{\gamma-m}{\sqrt{2}\sigma}\right)
\end{aligned} \tag{7.140}$$

Now if the number of terms $M/(N+C)$ summed to form each L'_k is large, then L'_k is approximately Gaussian. Formulas (7.55) and (7.62) follow directly from (7.140). Formula (7.66) follows from the observation that $L'_q - L'_k$ is Gaussian, while the mean (7.67) and variance (7.68) come from:

$$\begin{aligned}
\mathrm{E}\left[L'_q - L'_k \middle| H_q, \underline{\phi}_q\right] &= \mathrm{E}\left[L'_q \middle| H_q, \underline{\phi}_q\right] - \mathrm{E}\left[L'_k \middle| H_q, \underline{\phi}_q\right] \qquad (7.141)\\
\mathrm{Var}\left[L'_q - L'_k \middle| H_q, \underline{\phi}_q\right] &= \mathrm{Var}\left[L'_q \middle| H_q, \underline{\phi}_q\right] - 2\mathrm{Cov}\left[L'_k, L'_q \middle| H_q, \underline{\phi}_q\right] + \mathrm{Var}\left[L'_k \middle| H_q, \underline{\phi}_q\right]
\end{aligned}$$

7.C Finite Order Likelihoods

Suppose we replace the "correct" likelihood formulas with the windowed formula:

$$L'[n] = \sum_l w[l]\hat{\phi}_k^2[n - l(N+C)] \qquad (7.142)$$

where $n \equiv k$. If we assume that the fault is continuous, the window is long, and the average energy in the fault is independent of time, then:

$$\begin{aligned}
\mathrm{E}\left[L'[n] \middle| H_q, \underline{\phi}_q\right] &= \sum_l w[l]\mathrm{E}\left[\hat{\phi}_k^2[n - l(N+C)] \middle| H_q, \underline{\phi}_q\right]\\
&\approx \left(\sum_l w[l]\right)\frac{N+C}{M}\sum_{r=0}^{M-1}\mathrm{E}\left[\hat{\phi}_k^2[r] \middle| H_q, \underline{\phi}_q\right]\\
&= \left(\sum_l w[l]\right)\frac{N+C}{M}\mathrm{E}\left[L'_k \middle| H_q, \underline{\phi}_q\right] \qquad (7.143)
\end{aligned}$$

This implies that the expected value of the windowed likelihood is the same as that for the ideal likelihood, except that the factor $M/(N+C)$ is replaced by $\sum_l w[l]$. For our rectangular window, $w[l] = 1$ for $l = 0, \ldots, K_{int} - 1$ and $= 0$ otherwise. For the IIR update, $w[l] = \alpha^l$ for $l \geq 0$ and $= 0$ otherwise. Therefore the integration factor $M/(N+C)$ in (7.58), (7.64), (7.67), and (7.87) is replaced by K_{int} for the rectangular window and by $1/(1-\alpha)$ for the IIR window.

Similarly, under the same assumptions

$$\begin{aligned}
\mathrm{Var}\left[L'[n] \middle| H_q, \underline{\phi}_q\right] &= \sum_l \sum_p w[l]w[p]\mathrm{Cov}\left[\hat{\phi}_k^2[n - l(N+C)], \hat{\phi}_k^2[n - p(N+C)] \middle| H_q, \underline{\phi}_q\right]\\
&= \sum_l w^2[l]\mathrm{Var}\left[\hat{\phi}_k^2[n - l(N+C)] \middle| H_q, \underline{\phi}_q\right]
\end{aligned}$$

$$\approx \left(\sum_l w^2[l]\right)\frac{N+C}{M}\sum_{r=0}^{M-1}\mathrm{Var}\left[\hat{\phi}_k^2[r]\,\middle|H_q,\underline{\phi}_q\right]$$

$$= \left(\sum_l w^2[l]\right)\frac{N+C}{M}\mathrm{Var}\left[L_k'\,\middle|H_q,\underline{\phi}_q\right] \tag{7.144}$$

where the second line follows because equation (7.100) implies that samples $\hat{\phi}_k[n]$ separated by multiples of $N+C$ are statistically independent. Formula (7.144) implies that the integration factor $M/(N+C)$ in (7.58), (7.64), (7.68), and (7.87) is replaced by K_{int} for the rectangular window and by $1/(1-\alpha^2)$ for the IIR window.

For short windows, the fault energy can no longer be modeled as independent of time. Furthermore, the oscillations in the tails of the high pass filter cause energy from one converter to contribute to all the likelihoods. The combination of these effects causes P_F, P_D, and especially P_M to degrade rapidly as the integration length decreases.

Chapter 8

Conclusion and Suggestions for Future Research

This thesis dealt with the problem of designing an arithmetic code to protect a given computation. We applied group theory, and considered computation which could be modeled as operations in an algebraic group, ring, field, or vector space. We showed that arithmetic codes belong to a class of mappings known as algebraic homomorphisms, and this led to a procedure for constructing codes based on identifying suitable homomorphisms. Our results are important because they are a mathematically rigorous formulation of the problem of designing arithmetic codes. They unify the existing literature on arithmetic codes, and offer the first general procedure for determining arithmetic codes for a wide class of systems.

8.1 Summary and Contributions

This thesis progressed from a general study of fault-tolerance to an examination of a specific class of operations. As more assumptions were made about the underlying structure of computation, we lost much of the generality of our approach. This, however, was balanced by the fact that the additional structure allowed us to more accurately identify the form of the arithmetic codes. The end result of our study was a full characterization of systematic-separate arithmetic codes for protecting computation in a wide variety of algebraic systems.

We began very abstractly and used set-theory to model an arbitrary multiple-input multiple-output system. We presented a decomposition of a fault-tolerant system into a cascade of three subsystems: a redundant computation unit, an error corrector, and a result decoder. Though not a rigorous canonical decomposition, it was arrived at from a careful study of existing fault-tolerant systems, and embodied all of the pertinent features shared by these systems. Conditions on redundancy were derived such that errors could be detected and corrected. This was done in a comprehensive manner. We considered the effects of multiple simultaneous errors and allowed a tradeoff between the abilities to detect and correct errors. We then identified an important and often-encountered class of errors that allows the inherent redundancy of a system to be more easily quantified by a single integer. We referred to these as symmetric errors, and they are analogous to the minimum distance of an error-correcting code.

This set-theoretic framework is an important contribution to the study of fault-tolerant systems. It is a general description of fault-tolerance, and makes few assumptions about the underlying system being protected or of the computation performed. It has a strong theoretical basis and encompasses all known fault-tolerant systems. Although specifically constructed for the study of fault-tolerant computation, the framework is applicable to any system utilizing redundancy. It could be used as a starting point for any future study in the areas of fault-tolerant computation or error-correcting codes.

Upon this set-theoretic framework, we built a description of arithmetic codes for protecting computation in algebraic groups. We modeled a code as adding redundancy by mapping computation to a group of higher order. By making a few elementary assumptions, we showed that the structure of a group restricted arithmetic codes to the class of mappings known as algebraic homomorphisms. This key insight opened the door to an examination of arithmetic codes through the study of homomorphisms. We restated redundancy conditions and developed general error detection and correction procedures. We showed that these procedures could be reduced to functions of a syndrome, and then extended the results for groups to rings, fields, and vector spaces.

This group-theoretic framework is noteworthy because it links the practical aspects of an arithmetic code to the theoretical definition of an algebraic homomorphism. Homomorphisms are a well-understood and deeply studied mathematical construct possessing a rich structure, and we drew heavily upon group theory to determine the possible form of homomorphisms. Our results for groups are also general enough to encompass the vast majority of existing arithmetic codes.

We then narrowed our focus from arbitrary algebraic homomorphisms to those yielding systematic-separate codes. These codes are preferable because they protect computation using a parallel independent parity channel. We first identified the general form of these codes and then gave a constructive procedure, based on a quotient group isomorphism, for determining possible codes. We proved that by finding all possible subgroups, our approach is guaranteed to find, up to isomorphisms, all possible systematic-separate codes. This code construction technique is the first procedure, outside of modular redundancy, for constructing a large class of arithmetic codes. It is probably the most useful aspect of this thesis, and has immediate, practical application.

Many examples of codes for protecting computation were given. Several were unique to our thesis, and were briefly mentioned. These included the matrix ring codes of Section 4.7.3 and the codes for linear convolution presented in Section 5.6.5. In addition, we presented detailed analyses of two practical fault-tolerant systems. In the first, we applied a polynomial residue number system to protect the convolution of discrete-time sequences. This yielded an efficient FFT-based algorithm which was suitable for multiprocessor implementations. Although not the first fault-tolerant convolution algorithm proposed (Section 5.6.5 gives an alternative method), it is better able to protect against the types of errors observed in real systems. We showed that single failures may be concurrently detected and corrected with as little as 65% overhead.

Our second major application dealt with reliable A/D conversion. We used a round-robin system architecture and added redundancy by oversampling the input signal. This oversampling process was, in fact, an algebraic homomorphism, and this enabled us to examine this system using the group-theoretic framework. We studied single converter failure, and the redundancy present in the code was well-suited to the errors introduced by a converter failure. We optimally handled the effects of quantization noise via a generalized likelihood ratio test. The error detection and correction algorithms reduced to a simple form, requiring about as much additional computation as an FIR filter.

8.2 Future Research Directions

This thesis laid a theoretical foundation for the study of arithmetic codes, and we foresee three immediate branches down which this work may continue. The first deals with exploring more fully some of the techniques and examples that were uncovered. The second has to do with applying our results to a wider range of systems. The third branch considers other possible uses for algebraic homomorphisms.

8.2.1 Further Exploration

This thesis approached the problem of designing an arithmetic code by focusing on the constraint that redundancy be preserved throughout computation. This led to the characterization of arithmetic codes as algebraic homomorphisms. In order to be useful, an arithmetic code must in addition be easily encoded and decoded, and must protect against the expected set of errors. These issues were only cursorily examined and not fully explored in this thesis. We feel, however, that these issues are very important and that it is impossible to design a practical arithmetic codes without taking them into consideration. It is doubtful, though, that a general solution to this problem can be found. The set of expected errors is too closely linked to the architecture used to perform the processing, and varies greatly across systems. Also, in some cases, the application of our framework reduces to a class of error-correcting codes for which these issues have not yet been fully resolved.

An efficient method of performing error correction is also lacking in our framework. We showed that the syndrome is a condensation of all information relevant to fault correction, and used this as an index into a lookup table in order to identify the error. This approach is practical only when there is a small number of expected errors. In practice, one may encounter a large or even an infinite number of possible errors, and in these instances a different approach to error correction is needed. One possible solution is to perform error correction in stages by classifying the set of errors. First, determine if an error has occurred within the system. If so, then determine which class the error falls in. Finally, identify the specific error within the determined class and correct its effect on the result.

An efficient method of correcting multiple errors is also needed. Presently, the complexity of the error corrector increases exponentially with the number of simultaneous errors, and so only single errors can usually be protected against. It is likely that an efficient method of correcting multiple errors exists. The syndrome mapping is a homomorphism, and this preserves the additive structure of multiple errors within the syndrome. A reasonable idea is to try combining this approach with the error classification scheme mentioned above.

Another drawback to our framework is that certain parts of the system, not covered by the arithmetic code, are assumed to be inherently robust. These include the error corrector and result decoder, and we assumed that they were protected by modular redundancy. This reliability was needed because of the manner in which we partitioned the operation of a robust system. It would be preferable to have an arithmetic code that could protect the entire operation of the system, not just limited portions. Alternatively, the design of self-checking systems for performing error correction and result decoding could be studied.

Chapters 4 and 5 each contain a wide variety of examples of operations protected by arithmetic codes. Our emphasis was on determining the form of the redundancy, and we were not greatly concerned with specific implementations. Although many of the examples reduced to systems examined by others, some were unique to this thesis. These include the matrix ring codes of Section 4.7.3 and the codes for linear convolution presented in Section 5.6.5. We feel that most of these examples, if explored more deeply, will lead to practical fault-tolerant implementations. Analyses in the same spirit as the studies of fault-tolerant convolution and A/D conversion in Chapters 6 and 7 are needed.

8.2.2 Arithmetic Codes for Other Operations

This thesis dealt exclusively with protecting computation that can be modeled as operations in algebraic groups, rings, fields, and vector spaces. These systems have a well-understood structure and include many important and computationally intensive operations. Their universality is, in fact, one of the driving forces behind the study of group theory. Still, there are a great many operations that cannot be placed in our framework. The ultimate goal of any general approach to fault-tolerance is to be able to protect an arbitrary computational unit, such as a CPU, in an efficient manner. The solution to this problem lies very far in the future, and it is arguable if any code, besides modular redundancy, would be able to protect it.

A step towards expanding the scope of this thesis is to apply our framework to other algebraic systems such as monoids or associate groups, or to the more general case of a set with operators. Homomorphisms are a constantly recurring theme in group theory, and they exist in other algebraic systems as well. Most likely they can be used, as in this thesis, to identify possible arithmetic codes.

The tools for detecting and correcting errors developed in Chapter 4 were based on only a few basic premises: that the set of outputs forms an Abelian group, that an additive error model applies, and that valid results lie in a subgroup. These constraints were automatically satisfied by our assumptions of computation in algebraic groups and in the manner in which redundancy was added. Instead of focusing on binary operations to begin with (this is a major assumption), it might be possible to design robust systems by focusing on the three premises just mentioned. The main problem to overcome would be to develop a constructive approach for adding redundancy such that valid results lie in a subgroup of the output set.

8.2.3 Other Possible Uses of Homomorphisms

This thesis examined the use of algebraic homomorphisms as arithmetic codes. Other uses for homomorphisms exist as well.

The interaction between codewords and errors in an error-correcting code may be modeled as an additive operation in an algebraic group. When viewed in this light, an error-correcting code satisfies the postulates of an algebraic homomorphism, and when dealing with binary vector spaces, leads to the standard class of linear error-correcting codes. The additive error model is a good description of the interaction between codewords and additive noise in a communication channel.

A promising departure from this additive model is to consider designing codes which protect against convolutional interference. This type of interference models the distortion introduced by a nonuniform transfer characteristic in a communication channel. The traditional approach to solving this problem is to use an adaptive equalizer to correct the distortion. This is an expensive solution that often has difficulty correcting rapidly changing channels such as those encountered in mobile communication systems.

Codes that protect against this form of interference may be designed by beginning with a binary polynomial field rather than a vector space. The nonzero elements of the field form a group under the multiplicative operation, and this operation may be used to model the effects of errors. In this case, an error polynomial would be multiplied with the codewords. These codes may be placed in our framework because of the generality of group theory. We only require an additive model in some group, and this additive operation need not correspond to vector addition. Valid codewords would consist of the set of prime polynomials, and error detection and correction would reduce to polynomial factorization.

A radically different application of algebraic homomorphisms is in the processing of encrypted information [55]. In many applications processing is done on sensitive information. The current practice is to store data in encrypted form, and then to decrypt it prior to processing. This decryption step is undesirable since it increases the likelihood of unauthorized access. It may be possible, however, to process data in an encrypted form using an algebraic homomorphism. In this case, a nonsystematic homomorphism is desired that satisfies different design goals. The main constraint is that the codes be easily encrypted and decrypted if the proper key is known, and virtually impossible to decrypt without it. Number theory, a branch of group theory, is at the heart of many encrypting schemes, and thus it seems reasonable that many of our group-theoretic results may be used to process encrypted data.

Bibliography

[1] J. A. Abraham, "Fault-tolerance techniques for highly parallel signal processing architectures," *SPIE Highly Parallel Signal Processing Architectures*, vol. 614, pp. 49–65, 1986.

[2] G. R. Redinbo, "Signal processing architectures containing distributed fault-tolerance," in *Conference Record - Twentieth Asilomar Conference on Signals, Systems & Computers*, pp. 711–716, IEEE, November 1987.

[3] K. Fitzgerald, "As chip geometries continue to shrink, IC reliability grows ever more critical," *I.E.E.E. The Institute*, p. 10, June 1988.

[4] D. A. Rennels, "Fault-tolerant computing - concepts and examples," *IEEE Transactions on Computers*, vol. C-33, pp. 1116–1129, December 1984.

[5] J. von Neuman, *Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components*, pp. 43–98. Princeton University Press, 1956.

[6] T. R. N. Rao and E. Fujiwara, *Error-Control Coding for Computer Systems.* Englewood Cliffs, N.J.: Prentice-Hall, 1989.

[7] J. G. Tryon, *Redundancy Techniques for Computing Systems*, ch. Quadded–Logic, pp. 205–208. Washington, D.C.: Spartan Books, 1962.

[8] 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," in *Proceedings, 1st International Conference on Fault-Tolerant Computing*, pp. 1312–1321, November 1971.

[9] R. E. Harper, J. H. Lala, and J. J. Deyst, "Fault-tolerant parallel processor architecture overview," in *Eighteenth International Symposium on Fault-Tolerant Computing, Digest of Papers*, pp. 252–257, IEEE, June 1988.

[10] J. A. Katzman, "A fault-tolerant computing system," in *Proceedings 11th Hawaii International Conference on System Sciences, Pt. III*, pp. 85–102, 1978.

[11] R. Freiburghoue, "Making processing fail-safe," *Mini-Micro Systems*, pp. 255–264, May 1982.

[12] J. H. Patel and L. Y. Fung, "Concurrent error detection in ALU's by recomputing with shifted operands," *IEEE Transactions on Computers*, vol. C-31, pp. 589–595, July 1982.

[13] M. Namjoo and E. J. McCluskey, "Watchdog processors and capabililty checking," in *Proceedings, 12th International Symposium on Fault-Computing*, pp. 245–248, 1982.

[14] R. E. Blahut, *Theory and Practice of Error Control Codes.* Reading, MA.: Addison-Wesley Publishing Company, 1984.

[15] N. S. Szabo, *Residue Arithmetic and its Applications to Computer Technology.* New York, N.Y.: McGraw-Hill, 1967.

[16] M. H. Etzel and W. K. Jenkins, "Redundant residue number systems for error detection and correction in digital filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-28, pp. 538–544, October 1980.

[17] S. S.-S. Yau and Y.-C. Liu, "Error correction in redundant residue number systems," *IEEE Transactions on Computers*, vol. C-22, pp. 5–11, January 1973.

[18] W. K. Jenkins, "Residue number system error checking using expanded projection," *Electronic Letters*, vol. 18, pp. 927–928, October 1982.

[19] C.-C. Su and H.-Y. Lo, "An algorithm for scaling and single residue error correction in residue number systems," *IEEE Transactions on Computers*, vol. 39, pp. 1053–1064, August 1990.

[20] H. Krishna and K. Y. Lin, "A coding theory approach to error control in redundant residue number systems – Part I: Theory and single error correction." Received for review, October 1990.

[21] W. K. Jenkins and E. J. Altman, "Self-checking properties of residue number error checkers based on mixed radix conversions," *IEEE Transactions on Circuits and System*, vol. 35, pp. 159–167, February 1988.

[22] Marshall, Jr., Thomas G., "Coding of real number sequences for error correction: A digital signal processing problem," *IEEE Journal on Selected Areas in Communications*, vol. SAC-2, pp. 381–392, March 1984.

[23] K.-H. Huang, *Fault-Tolerant Algorithms for Multiple Processor Systems.* PhD thesis, University of Illinois, Urbana-Champaign, November 1983.

[24] K.-H. Huang and J. A. Abraham, "Algorithm-based fault-tolerance for matrix operations," *IEEE Transactions on Computers*, vol. C-33, pp. 518–528, June 1984.

[25] J.-Y. Jou, *Fault-Tolerant Matrix Arithmetic and Signal Processing on Highly Concurrent VLSI Systems.* PhD thesis, University of Illinois, Urbana-Champaign, IL., 1985.

[26] J.-Y. Jou and J. A. Abraham, "Fault-tolerant matrix arithmetic and signal processing on highly concurrent computing structures," *Proceedings of the IEEE*, vol. 74, pp. 732–741, May 1986.

[27] J.-Y. Jou and J. A. Abraham, "Fault-tolerant algorithms and architectures for real time signal processing," in *Proceedings – International Conference on Parallel Processing*, pp. 359–362, IEEE, August 1988.

[28] V. S. S. Nair, "General linear codes for fault-tolerant matrix operations on processor arrays," Master's thesis, University of Illinois, Urbana, IL., August 1988.

[29] V. S. S. Nair and J. A. Abraham, "General linear codes for fault-tolerant matrix operations on processor arrays," in *Proceedings - International Symposium on Fault-Tolerant Computing*, pp. 180–185, IEEE, June 1988.

[30] V. S. S. Nair and J. A. Abraham, "Real-number codes for fault-tolerant matrix operations on processor arrays," *IEEE Transactions on Computers*, vol. 39, pp. 426–435, April 1990.

[31] C. J. Anfinson and F. T. Luk, "Linear algebraic model of algorithm-based fault-tolerance," *IEEE Transactions on Computers*, vol. 37, pp. 1599–1604, December 1988.

[32] P. Banerjee, J. T. Rahmeh, C. Stunkel, V. S. Nair, K. Roy, V. Balasubramanian, and J. A. Abraham, "Algorithm-based fault-tolerance on a hypercube multiprocessor," *IEEE Transactions on Computers*, vol. 39, pp. 1132–1145, September 1990.

[33] G. M. Megson and E. D. J., "Algorithmic fault-tolerance for matrix operations on triangular arrays," *Parallel Computing*, vol. 10, pp. 207–219, April 1989.

[34] C. J. Anfinson, A. W. Bojanczyk, F. T. Luk, and E. K. Torng, "Algorithm-based fault-tolerant techniques for MVDR beamforming," in *Proceedings – International Conference on Acoustics, Speech, and Signal Processing*, pp. 2417–2420, IEEE, May 1989.

[35] F. T. Luk and H. Park, "Analysis of algorithm-based fault-tolerance techniques," *Journal of Parallel and Distributed Computing*, vol. 5, pp. 172–184, April 1988.

[36] W. G. Bliss and A. W. Jullien, "Efficient and reliable VLSI algorithms and architectures for the discrete Fourier transform," in *Proceedings – International Conference on Acoustics, Speech, and Signal Processing*, pp. 901–904, IEEE, 1990.

[37] W. G. Bliss and M. R. Lightner, "The reliability of large arrays for matrix multiplication with algorithm-based fault-tolerance," in *Wafer Scale Integration III* (M. Sami and F. Distante, eds.), pp. 305–316, Elsevier Science Publishers B. V. (North-Holland), 1990.

[38] W. S. Song and B. R. Musicus, "Fault-tolerant architecture for a parallel digital signal processing machine," in *Proceedings – 1987 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, pp. 385–390, IEEE, October 1987.

[39] W. S. Song, *A Fault-Tolerant Multiprocessor Architecture for Digital Signal Processing Applications*. PhD thesis, M.I.T., January 1989.

[40] B. R. Musicus and W. S. Song, "Fault-tolerant digital signal processing via generalized likelihood ratio tests." To appear in *IEEE Transactions on Signal Processing*.

[41] Y.-H. Choi and M. Malek, “Fault-tolerant FFT processor,” *IEEE Transactions on Computers*, vol. 37, pp. 617–621, May 1988.

[42] J.-Y. Jou and J. A. Abraham, “Fault-tolerant FFT networks,” *IEEE Transactions on Computers*, vol. 37, pp. 548–561, May 1988.

[43] G. R. Redinbo, “Data protection in convolution computations,” in *Proceedings – International Conference on Acoustics, Speech, and Signal Processing*, pp. 1095–1098, IEEE, May 1989.

[44] G. R. Redinbo, “System level reliability in convolution computations,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, pp. 1241–1252, August 1989.

[45] I. N. Herstein, *Topics in Algebra.* New York, NY: John Wiley and Sons, 1975.

[46] J. K. Wolf, “Redundancy, the discrete Fourier transform, and impulse noise cancellation,” *IEEE Transactions on Communications*, vol. COM-31, pp. 458–461, March 1983.

[47] W. W. Peterson, “On checking an adder,” *IBM Journal*, pp. 166–168, April 1958.

[48] G. Strang, *Linear Algebra and its Applications.* Orlando, FL: Academic Press, Inc., 1980.

[49] D. Chin, J. Passe, F. Bernard, H. Taylor, and S. Knight, “The Princeton engine: A real-time video system simulator,” *IEEE Transactions on Consumer Electronics*, vol. 34, pp. 285–297, May 1988.

[50] J. H. McClellan and C. M. Rader, *Number Theory in Digital Signal Processing.* Englewood- Cliffs, N.J.: Prentice-Hall, 1979.

[51] R. E. Blahut, *Fast Algorithms for Digital Signal Processing.* Reading MA.: Addison-Wesley Publishing Company, 1985.

[52] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor, *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing.* New York, N.Y.: IEEE Press, 1986.

[53] C. S. Burrus, *DFT/FFT and Convolution Algorithms.* New York, N.Y.: Wiley Interscience, 1985.

[54] A. V. Oppenheim and C. J. Weinstein, “Effects of finite register length in digital filtering and the fast Fourier transform,” *Proceedings of the IEEE*, vol. 8, pp. 957–976, August 1972.

[55] R. L. Rivest, L. Adleman, and M. L. Dertouzos, *Foundations of Secure Computation*, ch. On Data Banks and Privacy Homomorphisms, pp. 169–180. New York, N.Y.: Academic Press, 1978.