

A Fault-Tolerant Multiprocessor Architecture For Digital Signal Processing Applications

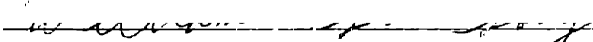
by
William S. Song

B.S. Massachusetts Institute of Technology (1982)
M.S. Massachusetts Institute of Technology (1984)

Submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirement
for the degree of
Doctor of Philosophy
at the
Massachusetts Institute of Technology
January 1989

©William S. Song 1988

The author hereby grants M.I.T. permission to reproduce and to
distribute copies of this thesis document in whole or in part.

Signature of Author 
Department of Electrical Engineering and Computer Science
January 26, 1989

Certified by _____
Bruce R. Musicus
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Students

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY
MAY 10 1989
ARCHIVES
LIBRARIES

Report Documentation Page			Form Approved OMB No. 0704-0188		
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE JAN 1989		2. REPORT TYPE		3. DATES COVERED 00-01-1989 to 00-01-1989	
4. TITLE AND SUBTITLE A Fault-Tolerant Multiprocessor Architecture for Digital Signal Processing Applications				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 77 Massachusetts Avenue, Cambridge, MA, 02139				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES 143	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

A Fault-Tolerant Multiprocessor Architecture For Digital Signal Processing Applications

Abstract

Proposed is a fault-tolerant multiprocessor architecture which needs much less redundant hardware than Modular Redundancy architectures. The architecture uses weighted checksum techniques and is suited for linear Digital Signal Processing applications in which multiple copies of the identical processor are used to meet the throughput requirement. Single fault detection/correction and multiple detection/correction techniques are discussed. Also proposed are statistical fault detection/correction algorithms for systems containing numerical roundoff or truncation noise such as fixed point or floating point systems. Presented are the simulations of these algorithms as well as the simulations of numerical noise distributions in real fixed point system applications. Our choice of weights reduces the dynamic range requirement of the checksum processors and minimizes the masking of small faults by the numerical noise. Efficient fault detection/correction algorithms for the exact arithmetic systems are presented, including one for residue arithmetics systems. Practical architectures for implementing the single fault detection/correction algorithm are also presented. These architectures are designed to mask any single component failure in the system.

Acknowledgements

I would like to express my thanks to:

Prof. Bruce R. Musicus for his excellent advice, creative ideas, and exuberant enthusiasm.

Paul R. Blasche for his guidance and his advice on the applications aspects of this work.

This research has been partly funded by US Air Force Office of Scientific Research (Contract AFOSR-86-0164) and by Draper Laboratories, Cambridge, Massachusetts.

Contents

1	Introduction	9
2	Fault-tolerant DSP Multiprocessor Systems	13
2.1	The Checksum Architecture	13
2.1.1	Single Fault Detection	13
2.1.2	Single Fault Correction	15
2.2	The Weighted Checksum Architecture	17
2.2.1	The Use of Weighted Checksums	17
2.2.2	Single Fault Correction	22
2.2.3	Block Matrix Notation	23
2.2.4	Multiple Fault Detection/Correction	25
2.2.5	Choice of Weight Vectors for Single Fault Correction	26
2.2.6	Reliability	29
3	Numerical Noise	31
3.1	The Effects of Numerical Noise	31
3.2	Single Fault Detection	32
3.2.1	The Threshold Method	32
3.2.2	Effects of Incorrect Fault Diagnosis	33
3.2.3	Dynamic Range vs. Numerical Noise	34
3.3	Single Fault Correction	35
3.3.1	The Projection Method	35
3.3.2	Effects of Incorrect Fault Diagnosis	39
3.3.3	Dynamic Range vs. Numerical Noise	42
3.4	Simple Case	43
3.4.1	Simple Case of Projection Method	43
3.4.2	Computational Overhead Example	46

3.5	Other Methods Single Fault Correction Methods	48
3.5.1	Using Range Test on The Syndromes	48
3.5.2	Error Coding Method	50
3.5.3	Angular Method	50
4	Generalized Likelihood Ratio Test	54
4.1	Single Fault Correction	54
4.2	Reducing The Numerical Noise	57
5	Fixed Point System Simulations	61
5.1	Numerical Noise	61
5.2	Single Fault Correction System Simulation	63
5.2.1	Simulated System	64
5.2.2	The False Alarm Rate	66
5.2.3	Dependence on q	70
5.2.4	Dependence on Changing False Alarm Rate	73
5.3	Probability Distribution of the Numerical Noise	74
5.4	Numerical Noise Histograms of Real Applications	78
6	Multiple Fault Correction Systems with Numerical Noise	83
6.1	Projection Method	83
6.1.1	K_m Fault Correction	83
6.1.2	White Noise Case	84
6.1.3	Reliability	86
6.1.4	Weight Vectors	87
6.1.5	Practicality	89
6.2	Generalized Likelihood Ratio Test	90
7	The Exact Arithmetic Systems	92
7.1	The Integer Arithmetic Systems	92
7.1.1	Single Fault Correction	92
7.1.2	Multiple Fault Correction	92
7.1.3	Modulo Arithmetic in Checksum Processors	95
7.2	Residue Arithmetic Systems	96
7.2.1	Residue Number System Multiprocessors	96
7.2.2	Modulo Arithmetic Processor Systems	98

8	Practical Architectures	106
8.1	Single Bus Architecture	107
8.2	Unidirectional Data Flow Architecture	111
8.3	Variations	112
9	Conclusion	116
A	Proof of Section 2.2.4	120
B	Derivation of GLRT (Proof of Section 4.1)	122
C	Derivation of Alternate GLRT (Proof of Section 4.2)	126
D	Mean and Variance of Likelihoods	132
E	Mean and Variance of Processor Output Estimates	137
F	Proof of $WW^T = I$ For Complete Sets of Weight Vectors	140

List of Figures

1.1	Triple Modular Redundancy	10
1.2	Double Modular Redundancy	11
2.1	Information Spreading for Fault Tolerance	14
2.2	Single Fault Detection Checksum Architecture	16
2.3	Single Fault Correction Checksum Architecture	18
2.4	Weighted Checksum Architecture	19
2.5	Possible Single Fault Syndrome Values	24
3.1	Projection Method Decision Regions, $N = 2, C = 2$	40
3.2	Overhead Ratio for Complex FFT ($N=10, C=3$)	48
3.3	Range Test Method Decision Regions, $N = 2, C = 2$	51
3.4	Angular Method Decision Regions, $N = 2, C = 2$	53
5.1	Error Distribution for One Roundoff	74
5.2	Noise Distribution of M Roundoff Operations	77
5.3	Noise Histogram of FIR Single Fault Detection System	80
5.4	Noise Histogram of 64 Point FFT Single Fault Detection System	81
5.5	Noise Histogram of 1000 Point FFT Single Fault Detection System	82
8.1	Single Bus Architecture	108
8.2	Single Bus Architecture Timing Diagram	109
8.3	Bus Guardian Unit	109
8.4	Unidirectional Data Flow Architecture	113
8.5	Unidirectional Data Flow Architecture Timing Diagram	114

List of Tables

2.1	The Maximum N for the Weights Used	28
3.1	Number of Multiplies and Adds	47
5.1	Simulation Histogram of Single Fault Correction ($N=10, C=3$)	68
5.2	Angle Between Neighboring Weight Vectors	69
5.3	Simulation with Varying q	72
5.4	Simulation with Varying $p(L'_k > 0 H_0)$	73
7.1	Weight Vectors and Syndromes for Modulo 5 System ($N = 4,$ $C = 2$)	101
7.2	Weight Vectors for Double Fault Correction ($C = 4$)	104
7.3	Weight Vectors for Double Fault Correction ($C = 5$)	105

Chapter 1

Introduction

In conventional fault-tolerant applications, multiple copies of the processor are used to mask one or more faults. This technique is called the Modular Redundancy Technique [Johnson 84, Nelson 82, Losq 76, de Sousa 78, Wensley 78]. One of the most popular is Triple Modular Redundancy [von Neumann 56, Johnson 84, Siewiorek 82], that von Neumann proposed during 1950's in which three identical processors and a majority voter are used to mask a single fault as shown in figure 1.1. The major advantage of Triple Modular Redundancy is that it is simple to implement and that it can be used for arbitrary applications. The major disadvantage of the Triple Modular Redundancy Technique is that it requires much excess hardware. Two-thirds of the hardware is being used for fault-tolerance purposes.

Double Modular Redundancy shown in figure 1.2 is another alternative, in which two copies of the processor are used to detect a single fault. In this case, only half of the hardware is being used for fault-tolerance purposes. However, once the fault is detected, the system is left with the non-trivial time-consuming task of figuring out which processor is at fault. One way to check which processor is faulty is to interrupt the processors and run self-diagnostic programs. Although a "permanent" hardware fault can be discovered this way, a "transient" fault would not be discovered with this method. One possible method to discover the transient fault is simply to do the computation over again. If the fault were transient, the results should agree a second time. In order to do this, the processors have to keep record of "check points" where the internal states of the processors are stored away at regular intervals so that when a fault is discovered, the processor can be

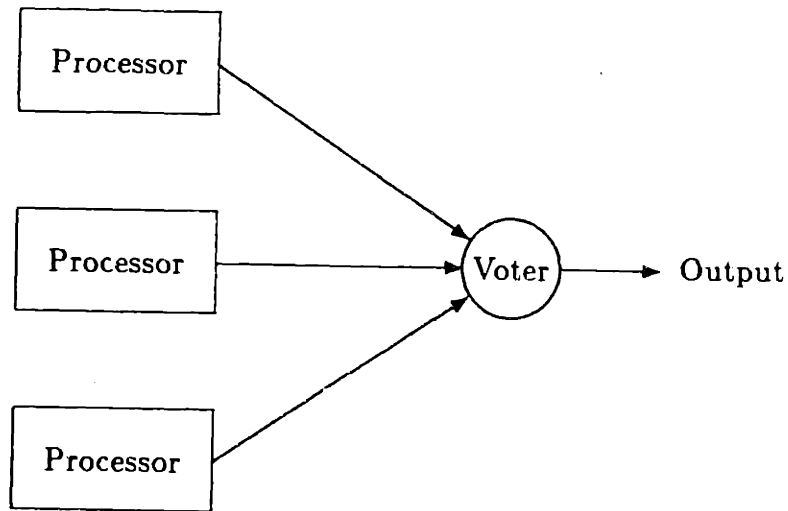


Figure 1.1: Triple Modular Redundancy

restarted from the last check point.

There are a number of other alternatives which attempt to increase the hardware utilization to above 50 %. One example is the self-testing system [Johnson 84]. The idea is that the system occasionally suspends the current job, stores the internal state, and runs the diagnostic test programs which check for the hardware faults. Another example is the Roving Emulator system [Breuer 83]. In this system, only one part of the system is tested for the fault at one time. The fault checker checks a hardware module by emulating its function, using the same inputs and internal states, and comparing the output. If no fault is detected for a while, the fault checker proceeds to check another hardware module. Although these systems may utilize greater than 50% of hardware, they do not detect or mask all the faults, and it may take a while to detect a failure.

The systems that achieve high reliability with little redundant hardware are data transmission systems using error coding techniques. The idea behind this system is to encode the data with a reliable encoder, transmit the encoded data using a slightly larger bandwidth than the minimum bandwidth required by the non-encoded data, and then decode the data at the receiving end with a reliable decoder. Even if some data were destroyed

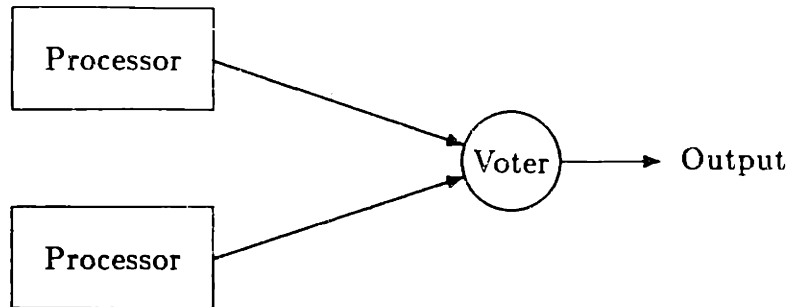


Figure 1.2: Double Modular Redundancy

in the transmission process, the decoder can reconstruct the data as long as not too much of the data were destroyed. How much extra transmission bandwidth is required depends on the noise model of the transmission channel and the reliability requirement of the application. Error coding techniques have also been used in memory systems. Although the error coded systems achieve high reliability with low hardware overhead, they can only be used when the output is identical to the input.

However, if we restrict the class of applications, it is possible to apply techniques similar to the error coding technique to other systems in order to achieve high reliability with little hardware overhead. Good examples of reliable systems with little hardware overhead have been studied by Huang and Abraham [Huang 82, Huang 84] and Jou [Jou 84, Jou 86]. They have achieved single fault correction for matrix operations in linear or mesh-connected processor arrays using the weighted checksum approach. The input matrix is encoded using the weighted checksum technique, processed in a processor array containing slightly more processors than the non-fault-tolerant case, and the output matrix is decoded also using the weighted checksum technique. However, their architecture is limited to doing the matrix operations in array processors, and their fault-tolerance algorithm only covers the processor calculation error. It is assumed that there is no fault in the datapath and that the faulty processor is still capable of passing the incoming data onto the next processor without introducing an error. Their choice of weight also requires the dynamic range of the "extra" processors to be much greater than the original processors, and the

numerical noise from “extra” processors heavily masks the errors from other processors in non-exact arithmetic systems (fixed or floating point systems). They also do not have effective fault detection/correction algorithms for the systems with numerical noise.

Proposed in this thesis is a more general fault-tolerant multiprocessor architecture with low hardware overhead that can be used for any linear signal processing purpose. It also uses weighted checksum techniques and its datapath is designed to tolerate any single point failure. Our choice of weights reduces the dynamic ranges of the the “extra” checksum processors, and improves the fault detection/correction capability in the presence of numerical noise. We also introduce effective fault detection/correction algorithms for the systems with numerical noise as well as for the exact arithmetic systems including the residue arithmetic systems.

Our work is presented as follows. In chapter 2, we shall present the basic idea behind the fault-tolerant multiprocessor architecture for DSP. In chapter 3, the fault detection/correction algorithms are presented for the systems containing numerical roundoff or truncation noises. In chapter 4, we shall present the generalized likelihood ratio test for detection/correction of the faults. In chapter 5, the simulations of these fault detection/correction algorithms and the simulation of the numerical noise distribution in various systems are presented. In chapter 6, we shall discuss the multiple fault detection/correction algorithms in the presence of the numerical noise. In chapter 7, we shall present efficient algorithms of fault detection/correction for exact arithmetic systems including the residue number arithmetic systems. In chapter 8, we shall discuss the practical architectures suitable for implementing the fault-tolerant architecture. Chapter 9 is the conclusion.

Chapter 2

Fault-tolerant DSP Multiprocessor Systems

2.1 The Checksum Architecture

2.1.1 Single Fault Detection

Digital Signal Processing systems often have to perform linear processing tasks on massive amounts of incoming data at a very rapid rate, often in real time. Radar and sonar systems are typical of such applications. Massive computational requirements often lead to highly parallel multiprocessor architectures [Faithi 83]. In such multiprocessor environments, it is not unusual to have multiple processors doing the identical linear processing task. Typical of linear processing tasks are filters and transforms.

Here is an example of a simple multiprocessor DSP architecture, which uses a number of identical linear processors. Suppose the speed of a single processor is N times slower than what is required by the application. One can use N processors in a rotating basis to meet the throughput requirement. The first segment of the input data goes to the first processor, the second segment of the input data goes to the second processor, and so on. By the time the N^{th} processor has received its input, the first processor has outputted its results and is ready to receive the first segment of the next batch of data.

In a multiprocessor architecture with N processors doing identical linear tasks on different set of input data, it is possible to use a technique simi-

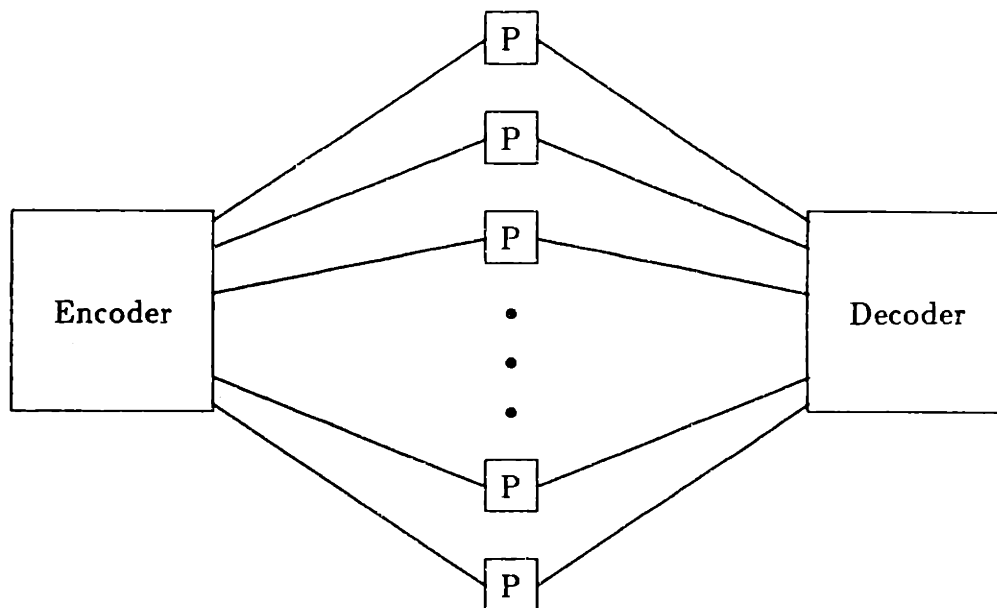


Figure 2.1: Information Spreading for Fault Tolerance

lar to error coding technique [Kohavi 78, Siewiorek 82] in order to achieve fault detection/correction. In data transmission systems, the data to be transmitted is often error coded to spread the information over a wider bandwidth. It is then sent over the transmission channel which requires a slightly larger bandwidth than the minimum, and is decoded on the receiving end. The reliability of the system can be greatly enhanced with relatively little extra transmission bandwidth, provided that encoders and decoders are fault-free. Similar things can be done in the multiprocessor environment. We would like to encode the input data and “spread” the information over a wider database. We would then want to process these data with slightly more processors than the minimum required, and decode the outputs as shown in figure 2.1. The encoding and decoding would be such that the desired number of processor faults can be detected or masked, provided that the encoder and the decoder are fault free.

The single fault detection version of such a system is shown in figure 2.2. It consists of N linear processors processing different sets of input

data. Let $\underline{x}_k$ be the input data segment and $\underline{y}_k$ be the output data segment of the processor k for the current batch of data. There is an extra $(N + 1)^{th}$ processor in the system which is responsible for fault detection. It is an identical processor doing the identical linear processing as all other processors, and its input is the sum of all other inputs.

$$\underline{x}_{N+1} = \sum_{k=1}^N \underline{x}_k \quad (2.1)$$

We shall call the extra $(N + 1)^{th}$ processor the checksum processor and all other processors the data processors. In absence of a fault, the output of the checksum processors should be equal to the output checksum, which is the sum of the outputs of the data processor. That means the syndrome $\underline{s}_{N+1}$ of the $(N + 1)^{th}$ checksum processor defined as

$$\underline{s}_{N+1} = \underline{y}_{N+1} - \sum_{k=1}^N \underline{y}_k \quad (2.2)$$

should be equal to zero when there is no fault. If there is a fault in the system, it would not be equal to zero. Therefore, one can achieve single fault detection with a single checksum processor.

2.1.2 Single Fault Correction

The checksum processor is very much like a "parity" processor. In the error coding technique with a parity bit, the parity bit is formed by modulo-2 addition of all the data bits or their complements. In the checksum processor case, the input to the checksum processor is formed by adding inputs of the data processors.

Treating the checksum processor like a parity processor, single error correcting codes such as Hamming Code can be directly applied to the multiprocessor system. In error correcting codes, there are a number of parity bits, each the modulo-2 sum of the different set of data bits. If there is no error, all the parity bits match correctly. If one of the bits is faulty, the faulty bit can be located by looking at which parity bits mismatch. Figure 2.3 shows how the Hamming Code can be used for single fault correction in the multiprocessor architecture to protect four data processors using three checksum processors. The processors 1 through 4 are data processors and

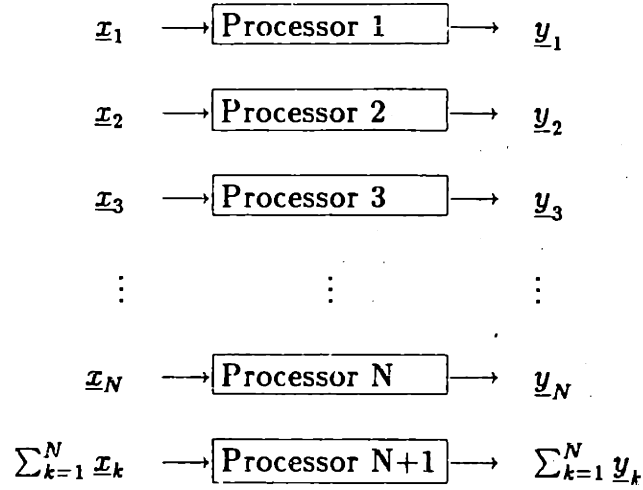


Figure 2.2: Single Fault Detection Checksum Architecture

the processors 5 through 7 are the checksum processors. On the right of the processors are their addresses which are used for fault location. Notice that the processor address is different from the processor number. The lines on the left of the processors represent which data processor inputs are used to form the inputs to each checksum processor. That means

$$\begin{aligned}
\underline{x}_5 &= \underline{x}_1 + \underline{x}_2 + \underline{x}_4 \\
\underline{x}_6 &= \underline{x}_1 + \underline{x}_3 + \underline{x}_4 \\
\underline{x}_7 &= \underline{x}_2 + \underline{x}_3 + \underline{x}_4
\end{aligned} \tag{2.3}$$

and

$$\begin{aligned}
\underline{s}_5 &= \underline{y}_5 - \underline{y}_1 - \underline{y}_2 - \underline{y}_4 \\
\underline{s}_6 &= \underline{y}_6 - \underline{y}_1 - \underline{y}_3 - \underline{y}_4 \\
\underline{s}_7 &= \underline{y}_7 - \underline{y}_2 - \underline{y}_3 - \underline{y}_4
\end{aligned} \tag{2.4}$$

Let us define the checksum match variable b_m to be equal to

$$b_m = \begin{cases} 0 & \text{if } \underline{s}_{N+1} = \underline{0} \\ 1 & \text{if } \underline{s}_{N+1} \neq \underline{0} \end{cases} \quad (2.5)$$

The faulty processor location is done from the codeword ($b_7 b_6 b_5$). When all the processors are working correctly, the codeword is (0 0 0). If one of the processors is faulty, the binary value codeword is equal to the address of the faulty processor.

If the faulty processor is a data processor, its correct output can be calculated using the syndrome of one of the checksum processors that are doing the checking on the faulty processor. If the k^{th} data processor is faulty, and if it is being checked by the m^{th} checksum processor, the correct output $\underline{y}_k$ is equal to

$$\underline{\bar{y}}_k = \underline{y}_k + \underline{s}_m \quad (2.6)$$

Note that when a checksum processor is detected as being at fault, it is usually not necessary to correct it.

Using this scheme, the number of the checksum processors C and the number of the data processors N required to achieve single fault correction are related by

$$N + C \leq 2^C - 1 \quad (2.7)$$

This formula comes from the fact that there are 2^C possible values for the code word ($b_{N+C} b_{N+C-1} \dots b_{N+1}$), which has to be able to represent $N + C$ possible single processor failure modes and one no-fault mode.

Unfortunately, the error coding technique cannot be used for the multiple fault detection/correction cases. This is because the error coding techniques are based on modulo-2 arithmetic.

2.2 The Weighted Checksum Architecture

2.2.1 The Use of Weighted Checksums

The checksum technique of the previous section is a special case of the weighted checksum technique. The weighted checksum technique requires

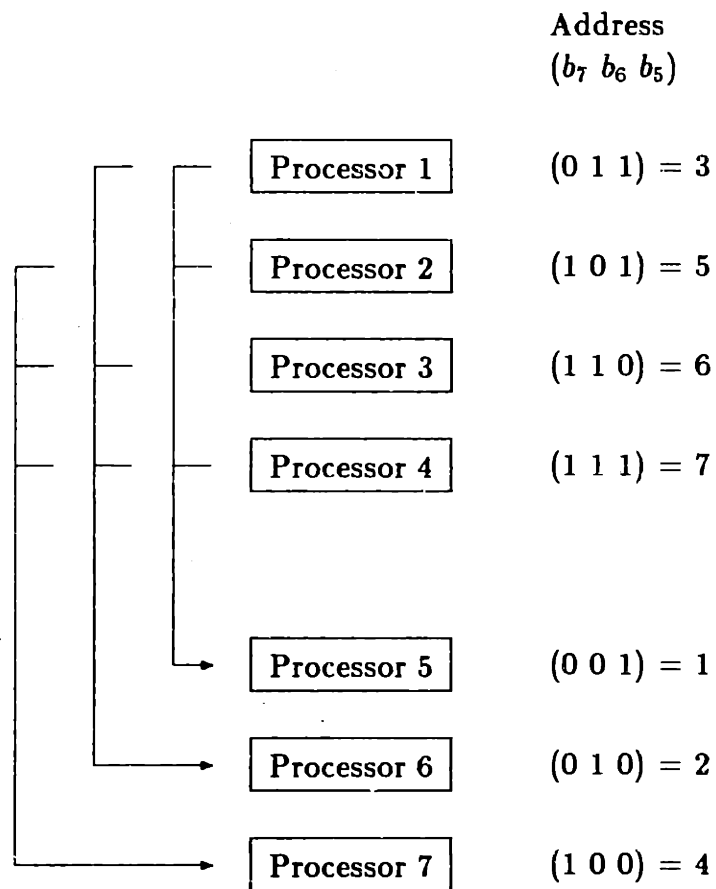


Figure 2.3: Single Fault Correction Checksum Architecture

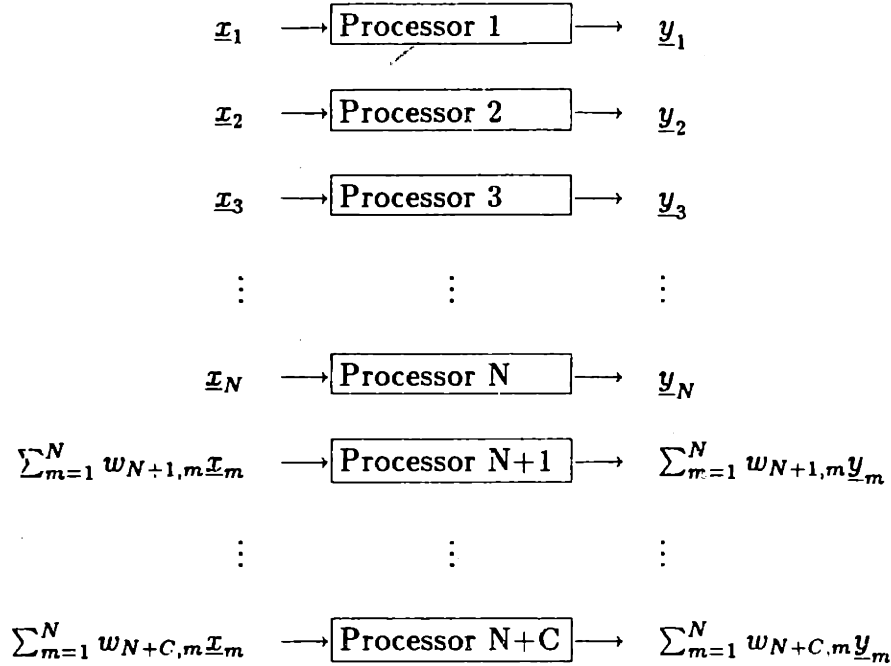


Figure 2.4: Weighted Checksum Architecture

fewer checksum processors than the simple checksum technique and can handle multiple fault detection/correction cases. In this case, the inputs to the checksum processors are formed by taking linear combinations of the data processor inputs. Suppose there are C checksum processors identical to the data processors as shown in figure 2.4. Then the inputs to those checksum processors are

$$\underline{x}_k = \sum_{m=1}^N w_{k,m} \underline{x}_m \quad \text{for } k = N+1, \dots, N+C \quad (2.8)$$

where $w_{k,m}$ are the scalar weights.

Let $\mathbf{F}$ be the linear function that the processors perform.

$$\underline{y}_k = \mathbf{F} \underline{x}_k \quad \text{for } k = 1, \dots, N \quad (2.9)$$

If we assume that input $\underline{x}_k$ has p data points and the output $\underline{y}_k$ has q data points, then $\mathbf{F}$ is a q by p matrix. The output of the checksum processors can now be written as

$$\underline{y}_k = \mathbf{F}\underline{x}_k = \sum_{m=1}^N w_{k,m} \mathbf{F}\underline{x}_m \quad \text{for } k = N+1, \dots, N+C \quad (2.10)$$

The syndrome $\underline{s}_k$ is defined as the difference between the checksum processor output and the corresponding output checksum.

$$\underline{s}_k = \underline{y}_k - \sum_{m=1}^N w_{k,m} \underline{y}_m \quad \text{for } k = N+1, \dots, N+C \quad (2.11)$$

If there is no fault in the system, the all $\underline{s}_k$ should be equal to zero. When there are faults in the system, let us assume that the fault in processor k introduces error ϵ_k on to the output $\underline{y}_k$.

$$\underline{y}_k = \mathbf{F}\underline{x}_k + \epsilon_k \quad \text{for } k = 1, \dots, N+C \quad (2.12)$$

The syndromes are now equal to

$$\underline{s}_k = \epsilon_k - \sum_{m=1}^N w_{k,m} \epsilon_m \quad \text{for } k = N+1, \dots, N+C \quad (2.13)$$

We can simplify this equation by defining the weights for the checksum processors to be

$$w_{N+i, N+j} = \begin{cases} -1 & \text{for } i = j = 1, \dots, C \\ 0 & \text{for } i \neq j \end{cases} \quad (2.14)$$

Now, the syndrome can be written as

$$\underline{s}_k = - \sum_{m=1}^{N+C} w_{k,m} \epsilon_m \quad \text{for } k = N+1, \dots, N+C \quad (2.15)$$

We can expand our fault detection/correction algorithms to include the faults in the input checksum calculations and the syndrome calculations, provided that each checksum processor has a separate hardware module for computing its input checksum and syndrome. The fault in the input checksum calculation or the syndrome calculation can be treated as if the fault were in the corresponding checksum processor. The fault occurring while calculating the input checksum causes the input checksum to deviate from the correct value by the error δ_k .

$$\underline{x}_k = \sum_{m=1}^N w_{k,m} \underline{x}_m + \underline{\delta}_k \quad \text{for } k = N+1, \dots, N+C \quad (2.16)$$

The fault occurring while calculating the syndrome introduces error $\underline{\lambda}_k$ on to the syndrome.

$$\underline{s}_k = \underline{y}_k - \sum_{m=1}^N w_{k,m} \underline{y}_m + \underline{\lambda}_k \quad \text{for } k = N+1, \dots, N+C \quad (2.17)$$

The effect of all three types of faults on the syndrome is

$$\underline{s}_k = \underline{\epsilon}_k + \mathbf{F} \underline{\delta}_k + \underline{\lambda}_k - \sum_{m=1}^n w_{k,m} \underline{\epsilon}_m \quad \text{for } k = N+1, \dots, N+C \quad (2.18)$$

Notice that the results of the input checksum error $\mathbf{F} \underline{\delta}_k$ and the output checksum error $\underline{\lambda}_k$ are indistinguishable from the checksum processor error $\underline{\epsilon}_k$ as far as their effects on the syndromes are concerned. They only affect the k^{th} syndrome, while the data processor error affects all the syndromes with non-zero weights. Therefore, we shall treat the input checksum or the syndrome calculation error as part of the checksum processor error. From now on, when we refer to checksum processor error, it shall automatically include the input checksum and the syndrome calculation error. Let us define the composite error $\underline{\phi}_k$ as

$$\underline{\phi}_k = \begin{cases} \underline{\epsilon}_k & \text{for } k = 1, \dots, N \\ \underline{\epsilon}_k + \mathbf{F} \underline{\delta}_k + \underline{\lambda}_k & \text{for } k = N+1, \dots, N+C \end{cases} \quad (2.19)$$

The syndrome $\underline{s}_k$ and the composite error $\underline{\phi}_k$ are related by

$$\underline{s}_k = - \sum_{m=1}^{N+C} w_{k,m} \underline{\phi}_m \quad \text{for } k = N+1, \dots, N+C \quad (2.20)$$

2.2.2 Single Fault Correction

Let us examine how a single fault can be detected and corrected using weighted checksums. When all the processors are working correctly, all the syndromes should be equal to zero. When only the processor k is faulty with the error $\underline{\phi}_k$, the syndromes would be

$$\begin{aligned} \underline{s}_{N+1} &= w_{N+1,k} \underline{\phi}_k \\ \underline{s}_{N+2} &= w_{N+2,k} \underline{\phi}_k \\ &\vdots \\ \underline{s}_{N+C} &= w_{N+C,k} \underline{\phi}_k \end{aligned} \quad (2.21)$$

Let us define the weight vector $\underline{w}_k$ to be

$$\underline{w}_k^T = (w_{N+1,k}, w_{N+2,k}, \dots, w_{N+C,k}) \quad (2.22)$$

With given $\underline{s}_k$'s, one can distinguish processor k_1 failure from processor k_2 failure if $\underline{w}_1$ is linearly independent from $\underline{w}_2$. That means the $\underline{w}_k$'s have to be linearly independent from each other in order to have single fault location. In order for the $\underline{w}_k$'s to be linearly independent, one needs at least two syndromes (i.e. two checksum processors).

Let us consider a very simple example case with two data processors and two checksum processors (i.e. $N=2$ and $C=2$). The weight vectors used are $\underline{w}_1^T = (1, 1)$, $\underline{w}_2^T = (1, -1)$. In figure 2.5, all the possible values for the syndromes are drawn in the syndrome plane. When the processor k is faulty, the syndrome would fall on the line along the vector $\underline{w}_k$ with its distance from the origin proportional to $\underline{\phi}_k$.

Once the faulty processor has been located, its correct output can be calculated from its faulty output and any one of the syndromes with non-zero weight. If the k^{th} data processor is faulty, its correct output $\underline{y}_k$ is equal to

$$\underline{\bar{y}}_k = \underline{y}_k + \frac{s_m}{w_{m,k}} \quad (2.23)$$

Note that when a checksum processor is detected as being faulty, it is not necessary to correct its output.

A special case of the single fault correction is when there is one data processor and two checksum processors ($N = 1, C = 2$) with the data processor weight vector $\underline{w}_1^T = (1, 1)$. This means that all three processors have the same input which is equivalent to the Triple Modular Redundancy technique.

2.2.3 Block Matrix Notation

The equations for the syndrome values in presence of a fault can be written in a convenient block vector ¹ notation. At this point, we shall define the syndrome block vector $\underline{s}$ and the composite error block vector $\underline{\phi}$ to be

$$\underline{s} = \begin{pmatrix} \underline{s}_{N+1} \\ \vdots \\ \underline{s}_{N+C} \end{pmatrix} \quad \underline{\phi} = \begin{pmatrix} \underline{\phi}_1 \\ \vdots \\ \underline{\phi}_{N+C} \end{pmatrix} \quad (2.24)$$

We also define the block weighting matrix $\mathbf{W}$,

$$\mathbf{W} = \begin{bmatrix} w_{N+1,1}\mathbf{I} & w_{N+1,2}\mathbf{I} & \dots & w_{N+1,N}\mathbf{I} & -\mathbf{I} & 0 & \dots & 0 \\ w_{N+2,1}\mathbf{I} & w_{N+2,2}\mathbf{I} & \dots & w_{N+2,N}\mathbf{I} & 0 & -\mathbf{I} & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ w_{N+C,1}\mathbf{I} & w_{N+C,2} & \dots & w_{N+C,N}\mathbf{I} & 0 & \dots & 0 & -\mathbf{I} \end{bmatrix} \quad (2.25)$$

where $\mathbf{I}$ is a q by q identity matrix. With this block matrix notation, the syndromes can simply be written as

$$\underline{s} = -\mathbf{W}\underline{\phi} \quad (2.26)$$

Let us also define $\mathbf{W}_k$ to be the k^{th} block column of $\mathbf{W}$. Then in case of the k^{th} processor failure, the syndrome will be:

¹Block vectors (matrices) are ordinary vectors (matrices) whose elements are themselves vectors (matrices).

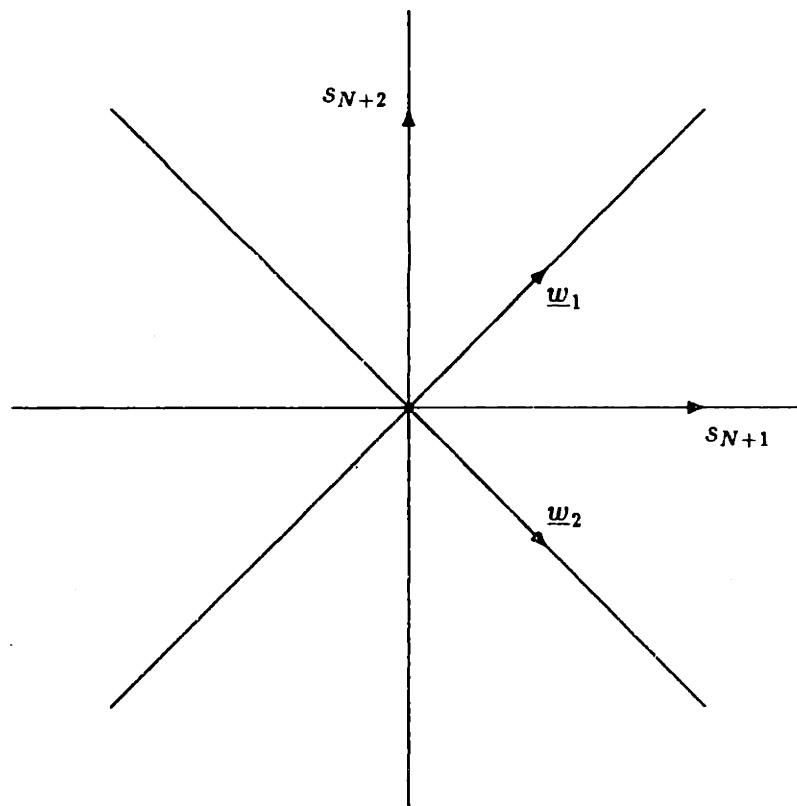


Figure 2.5: Possible Single Fault Syndrome Values

$$\underline{s} = -\mathbf{W}_k \underline{\phi}_k \quad (2.27)$$

2.2.4 Multiple Fault Detection/Correction

Let us examine what happens when there are two faulty processors in the single fault correction system with two checksum processors. The value of the syndrome $\underline{s}$ would be the linear combination of the two weight vectors corresponding to the faulty processors. Since all the weight vectors are linearly independent, the syndrome cannot be at the origin of the syndrome space. Thus, the presence of up to two faults can be recognized by the syndrome not being at the origin. However, one cannot distinguish which processors have failed, since the non-zero syndrome can be explained by the linear combination of any two processor failures. Also, one cannot always tell whether one processor has failed or two, since the linear combination of two processor failures can lie on another processor weight vector line. When there are three faulty processors, it is possible that the three processor errors would be multiplied by weights and added to form all-zero syndromes. Therefore, one cannot reliably detect three failures with two checksum processors.

In general, to be able to detect L_m failures, we have to make sure that every possible set of L_m weight vectors has to be linearly independent from each other, so that any combination of up to L_m failures would produce the non-zero syndromes. This requires the weight vectors to be at least L_m dimensional vectors which means that one needs at least L_m checksum processors. In order to be able to detect and correct up to K_m failures, every possible set of $2K_m$ weight vectors has to be linearly independent from each other. This is because the syndrome hyperplane defined by all possible linear combinations of one set of K_m weight vectors should not intersect with another hyperplane defined by all possible combinations of another set of K_m weight vectors, except at the origin. One needs at least a $2K_m$ dimensional syndrome space to be able to achieve this, and it requires at least $2K_m$ checksum processors. If one is interested in detecting up to $K_m + L_m$ failures and correcting failures if K_m or fewer failures have occurred, one needs a minimum of $2K_m + L_m$ checksum processors as proven in appendix A.

$$C \geq 2K_m + L_m \quad (2.28)$$

2.2.5 Choice of Weight Vectors for Single Fault Correction

The weight vectors for single fault correction should be chosen such that they minimize the computational effort involved in calculating the input checksums and the syndromes, as well as minimizing the number of checksum processors needed for protecting the given number of data processors. Jou and Abraham [Jou 84] used $w_{k,m} = 2^{(k-N-1)(m-1)}$ as weights for the data processors. These weights have the advantage that the multiply by a power of 2 can be done in simple bit-shifts. However, the dynamic range of the checksum processor registers have to be much greater than the data processors in order to be able to accommodate $w_{N+C,N} = 2^{(C-1)(N-1)}$. There are also disadvantages in having the weights varying largely in size. For example, in the presence of numerical computation noise, the numerical noise generated from the processors with large weights would mask the output signal of the processor with small weights in the syndrome calculations, as we shall see in later sections.

One choice of the weights that simplifies the checksum computation is to use only 0 and 1 as weights. This eliminates the multiplies in the checksum computations and is equivalent to using simple checksums instead of weighted checksums. It is also equivalent to using the conventional error coding techniques. Using these weights, there can be up to $N+C = (2^C - 1)$ weight vectors for a given number of checksum processors C . For example, we can have four data processors with three checksum processors ($C = 3$, $N = 4$). The $\mathbf{W}$ matrix in this case with $q = 1$ ($q = 1$ so that we do not have to write in the block matrix form with the identity matrix $\mathbf{I}$) would be

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 0 & 1 & -1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & -1 \end{bmatrix} \quad (2.29)$$

The $N + C = 7$ weight vectors in this case are seen as the block columns of the weight matrix $\mathbf{W}$. Recall that the case with one data processor

with two checksum processors ($N = 1, C = 2$) is equivalent to the Triple Modular Redundancy. The weight matrix in that case is

$$\mathbf{W} = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \quad (2.30)$$

If only 0, 1, and -1 are allowed to be weights, we still do not need multiplications in the checksumming process, but there can be more data processors for a given number of checksum processors than when only 0 and 1 were used as weights. There can be up to $N + C = (3^C - 1)/2$ weight vectors for a given C . For example, we can have ten data processors with three checksum processors to achieve single fault correction ($N = 10, C = 3$). The $\mathbf{W}$ matrix in such a case with $q = 1$ would be:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & 0 & 0 \\ 1 & -1 & 1 & 1 & 0 & 0 & 1 & 1 & -1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 0 & 0 & -1 \end{bmatrix} \quad (2.31)$$

Another choice for weights for single fault correction is to use small integers. This makes the multiplications in the checksum computations relatively easy and does not increase the dynamic range of the checksum processors nearly as much as Huang's choices for weights. A good way to get the single fault correction weight vector set is to start with all possible combinations of C weights each between $-M$ and $+L$ and to eliminate the zero vector and any vector which is linearly dependent on another vector. If weights between -2 and +2 were used, they would not require more than a shift operation for weight multiplication, and we can have $N + C = (5^C - 3^C)/2$ different weight vectors for given C . For example, with two checksum processors, we can have up to six data processors ($N = 6, C = 2$), with the following weight matrix:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 & 2 & 2 & -1 & 0 \\ 1 & -1 & 2 & -2 & 1 & -1 & 0 & -1 \end{bmatrix} \quad (2.32)$$

Table 2.1 shows the relationship between N and C depending on the range of weights used. Notice that one can protect a large number of data processors using relatively few checksum processors and using only very small integers as weights.

Weights Used	0, 1	0, ± 1	0, $\pm 1, \pm 2$
Max N for $C = 2$	1	2	6
Max N for $C = 3$	4	10	54
Max N for $C = 4$	11	36	268
Max N for $C = 5$	25	116	1436

Table 2.1: The Maximum N for the Weights Used

We define a “complete” set of weight vectors as the one that meets the following condition: if $(\alpha_1 \alpha_2 \dots \alpha_C)^T$ is a weight vector, then every $(\pm \alpha_1 \pm \alpha_2 \dots \pm \alpha_C)^T$ must also be a weight vector, or else its negative must be a weight vector. The complete set of weight vectors is created from the symmetric range of integers from $-L$ to $+L$. The weight matrix $\mathbf{W}$ for the complete sets of weight vectors has the property that $\mathbf{W}\mathbf{W}^T$ is a scaled identity matrix ($\mathbf{W}\mathbf{W}^T = \sigma_W^2 \mathbf{I}$ where σ_W^2 is a scalar) as proven in appendix F. This property becomes useful in later sections.

If complex arithmetic is used in the processors, we can have the weights of form $n + jm$ where n and m are small integers. Because the weights have twice the dimensionality of the non-complex case, we can have more data processors with the same number of checksum processors than in the non-complex case. For example, if we use $-1, 0$, and $+1$ for n and m , we can have $N + C = (9^C - 5^C)/4$ weight vectors possible. With two checksum processors, we can protect up to twelve data processors ($N = 12, C = 2$). This is many more than in the non-complex case with $-1, 0$, and $+1$ as weights, in which only two data processors can be protected with two checksum processors. The weight matrix in this case is:

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1+j & 1+j & 1+j & 1+j & -1 & 0 \\ 1 & j & -1 & -j & 1+j & -1+j & -1-j & 1-j & 1 & j & -1 & -j & 0 & - \end{bmatrix} \quad (2.33)$$

If 3 checksum processors were used, we can protect up to 148 data processors ($N = 148, C = 3$). The “complete” set of complex weight vectors of this form has the property that $\mathbf{W}\mathbf{W}^H = \sigma_W^2 \mathbf{I}$ where σ_W^2 is a scalar and $\mathbf{W}^H$ is the Hermitians (complex conjugate transpose) of $\mathbf{W}$.

2.2.6 Reliability

A system failure in the single fault detection or correction system occurs when there are two or more processor failures at the same time. Let us assume for convenience that only the processors can fail and that all other parts of the system are fault-free. Let P_f be the single processor failure rate. If we assume that P_f is much less than one ($P_f \ll 1$) and that the processor failures occur independently from each other, the system failure rate would be approximately equal to

$$\text{Prob}(> 1 \text{ failure}) \approx \frac{(N + C)(N + C - 1)}{2} P_f^2 \quad (2.34)$$

This is an approximation since the checksum processor failure rate would be slightly higher than P_f , since it includes the failure rate of the input checksum and the syndrome calculation. Let us compare this failure rate to the failure rate of the Triple Modular Redundancy (TMR) system in which each of the N processors are triplicated for single fault correction. The system failure in this case occurs when two or more processors fail at the same time in one or more of the N triplicated processor groups.

$$\text{Prob}(> 1 \text{ failure in any triple}) \approx 3NP_f^2 \quad (2.35)$$

The system failure rate in our architecture is approximately $(N + 2C)/6$ times higher than the TMR system failure rate, but the number of processors used ($N + C$) is significantly less than $3N$ in the TMR system.

The actual system failure rate of our system would be higher, since we have accounted only for the processor failure rate. We also have to account for the failure rate of any other parts of the system, such as the control, busses, the decoder which is responsible for locating faulty processor from the syndrome and correcting its output, and so forth. The actual system failure rate P_{sys} can be written as

$$P_{sys} \approx \frac{(N + C)(N + C - 1)}{2} P_f^2 + \sum_m P_m \quad (2.36)$$

where P_m is the failure rate of the m^{th} module of the system (not including the processors and the input checksum and the syndrome calculators). Notice the first term in the equation has the form P_f^2 , whereas the rest of

the terms have the form P_m . The square term is introduced by the single fault detection or correction of the processors. If any of the P_m were much greater than P_f^2 , then the system failure rate would be dominated by that term, and much of the effort that went into making the system fault-tolerant would be wasted. Therefore, any hardware module whose failure rate is significantly greater than P_f^2 should be protected by Triple Modular Redundancy or by other methods. If triple modular redundancy were used in some of the modules, the system failure rate would be equal to

$$P_{sys} \approx \frac{(N + C)(N + C - 1)}{2} P_f^2 + \sum_m P_m + \sum_l 3P_l^2 \quad (2.37)$$

where P_m is equal to failure rate of the non-triplicated hardware modules, and P_l is equal to the failure rate of the triplicated hardware module.

Chapter 3

Numerical Noise

3.1 The Effects of Numerical Noise

If only integer arithmetic were used in the system, all the calculations would be exact including the syndrome calculation. This makes it relatively trivial to detect and/or correct a single fault from the syndromes. In single fault detection, the syndrome would be exactly zero when there is no fault in the system, and non-zero when there is a fault. In locating the faulty processor, the non-zero syndrome would lie exactly on the corresponding weight vector line in case of a single fault.

However, if fixed point or floating point arithmetic were used, there would be roundoff or truncation noise included in the processor outputs as well as in the input checksums and the syndromes. This numerical noise causes the syndrome to deviate slightly from zero even when there is no fault in the system. In cases when there is a faulty processor in the system, the syndrome no longer lies exactly on the corresponding weight vector line, but may deviate from it slightly. We need to develop reasonable methods to sort out the small numerical noise in the syndromes from the effects of the faulty processor, and be able to make reasonable decisions as to which processor has failed.

A reasonable thing to do is to model the numerical noise as a statistical process, and attempt to make the system fault diagnosis based on the statistical noise model. However, if the fault diagnosis is based on a statistical noise model, any method that we devise for the fault diagnosis cannot be

correct all the time. There are many design criteria for the fault diagnosis method. Obviously, we would like the diagnosis to be as accurate as possible, as often as possible. More importantly, we want to design the system fault diagnosis method so that a false diagnosis would not have any detrimental effects on the system application.

3.2 Single Fault Detection

3.2.1 The Threshold Method

For the single fault detection system with one checksum processor, we have developed the following threshold method for detecting a fault. The idea behind this detection method is that the small non-zero syndrome is likely caused by numerical noise and the large non-zero syndrome is likely caused by a fault within the system. Let us assume that the expected mean of the numerical noise in the syndrome is equal to zero. If the numerical noise has a non-zero expected mean (such as one caused by some truncation methods), the mean value can be subtracted from the syndrome to yield the zero mean syndrome.

Let us start by examining the case when the fault detection is carried out for each point of the syndrome (i.e. as if $q = 1$). When the syndrome magnitude exceeds a certain preset threshold value γ , the system would be diagnosed as containing a fault.

$$\text{If } |s| > \gamma \text{ Then A Fault Detected} \quad (3.1)$$

The threshold value γ can be set in number of ways. For example, one can set it to be the maximum possible noise magnitude. This would prevent mistakenly declaring a failure when the non-zero syndrome is actually caused by computational noise. However, the system would not be able to detect small processor errors with such a large threshold. Furthermore, it is highly unlikely that the numerical noise magnitude of the syndrome magnitude would be even close to the maximum noise magnitude. A more reasonable threshold can be calculated by examining the probability distribution of the syndrome when no hardware fault has occurred (this distribution might be derived by modeling the numerical noise, or by running experiments). If the probability distribution of the fault is also known, it is

possible to set the threshold so that the threshold test gives the most likely explanation for the observed syndrome. However, the failure mechanisms are difficult to model accurately. One reasonable criterion for setting the threshold is to fix the false alarm rate at a desired value. The false alarm rate is equal to the probability that the syndrome magnitude exceeds γ under the no-fault condition, and is thus the integral of the probability distribution of the noisy syndrome in the range $|s| > \gamma$.

For the case when $q > 1$, the threshold testing can be done on the energy in the syndrome, which is equal to the sum of the squares of the syndrome data points.

$$\text{If } \underline{s}_{N+1}^T \underline{s}_{N+1} > \gamma \text{ Then A Fault Detected} \quad (3.2)$$

Again, the threshold value needed for the desired false alarm rate can be calculated from the probability distribution of syndrome energy $\underline{s}_{N+1}^T \underline{s}_{N+1}$ under the no-fault condition.

The output batch size q plays an important role in determining whether the system is better at detecting a small transient fault or a small permanent fault. The transient fault usually affects few data points within the batch and the permanent fault usually affects many data points within the batch. The mean of the syndrome energy $\underline{s}_{N+1}^T \underline{s}_{N+1}$ under the no-fault condition grows as $O(q)$, where as the standard deviation grows as $O(\sqrt{q})$. Therefore, with larger q , the system is more likely to detect a small permanent failure affecting many of the output data points. However, it is less likely to detect a transient failure affecting only few of the output data points, since even a relatively large transient failure may not increase the average syndrome energy enough to set off the detection. Of course, one has the option of testing the syndrome in both ways, using a large q and a small q at the expense of more computation.

3.2.2 Effects of Incorrect Fault Diagnosis

There are two possible false fault diagnoses in the single fault detection system. One is the false alarm case in which a large numerical noise sets off the fault detection mechanism, and the other is when a fault is very small and escapes detection. There is a tradeoff between these two kinds of fault misdiagnosis. If the threshold value is large, there is a low probability of

a false alarm cause by the numerical noise, but there is a high probability that a small fault may escape detection. If the threshold value is small, the false alarm probability is high, but there is less chance of a small fault escaping detection.

How the false alarm affects the system performance depends much on how the system handles an occurrence of a fault. If it is desirable to avoid the false alarm, the threshold value should be set sufficiently high.

In the case when a fault is too small to be detected reliably, the net effect of the fault is the slight increase in the noise level of the faulty processor output. An example of such a small fault is when the least significant bit of the processor output is stuck. Such a small fault can easily escape threshold detection, and would appear to the system as a slight increase in the noise level of the corresponding processor output. If the small undetected fault were in the checksum processor, it would not have any effect on the system application.

3.2.3 Dynamic Range vs. Numerical Noise

The dynamic range of the checksum processor is also an important consideration in designing a single fault detection system in the presence of numerical noise. One must prevent the overflow in the checksum processors and in the input checksum and the syndrome calculations. There is also a tradeoff between the number of bits used in the checksum processor registers and the fault detection capability. The exact nature of the tradeoff depends heavily on whether the fixed point or floating point algorithm is used and on the computational algorithms employed.

Since the checksum processor input is the weighted sum of N data processor inputs, the dynamic range of the k^{th} checksum processor should be ω_k times larger than the data processors in order to prevent overflow, where ω_k is defined as

$$\omega_k = \sum_{m=1}^N |w_{k,m}| \quad \text{for } k = N + 1, \dots, N + C \quad (3.3)$$

When floating point arithmetic is used, the dynamic range is not generally a big problem. However, when fixed point arithmetic is used, the

checksum processors should have $\log_2 \omega_k$ more bits in its registers than the data processors (more bits should also be used in the input checksum and syndrome calculations). If using more bits in the checksum processor registers is not desired because of the added complexity, the weights should be chosen so that $\omega_k \leq 1$ in order to prevent the overflow. For example, in the single fault detection system with one checksum processor, the weights $w_{N+1,k}$ can all be equal to $1/N$, except for $w_{N+1,N+1}$ which is equal to -1 . However, with these weights, the syndrome $\underline{s}_{N+1} = -\sum_{m=1}^{N+1} w_{N+1,m} \underline{\phi}_m$ is heavily dominated by the checksum processor error $\underline{\phi}_{N+1}$. The numerical noise from the checksum processor would be weighted $O(N)$ times higher than the noises from the data processors in the syndrome. This means that the noise from the checksum processor will mask the small faults from the data processors in the threshold test. The data processor fault size has to be significant before it is detected. One can eliminate this problem by making all the weights comparable in magnitude (for example, $w_{N+1,k} = 1$ for $k = 1, \dots, N$ and $w_{N+1,N+1} = -1$), but this requires adding $O(\log N)$ bits in the checksum processor registers.

3.3 Single Fault Correction

3.3.1 The Projection Method

In the single fault detection system with one checksum processor, we have used the threshold method for the fault diagnosis. This method assumes that small syndrome energy is likely caused by the numerical error and large syndrome energy is likely caused by a hardware fault. For the single fault correction system, we shall use a similar test for fault detection and location. An obvious method would be to do the threshold testing on each of the processors. That means doing the threshold test on the syndrome energy in each weight vector direction in the syndrome space. Such threshold tests involve projecting the syndrome $\underline{s}$ on to each weight vector and doing the threshold test on the energy of the projection. If the syndrome noises are white, we can use Euclidian projections, but if the syndrome noises are correlated, we should compensate for covariance of the syndromes when finding the "projection" along the weight vector.

The numerical noise variance in the syndrome space can be calculated

as follows. We assume that the numerical noises have zero mean and certain expected variances. When there is no fault in the system, the input checksum error $\underline{\delta}_k$, the processor noise $\underline{\epsilon}_k$, and the syndrome computation noise $\underline{\lambda}_k$ are no longer equal to zero and are modeled as zero-mean random variables. We shall define Δ_k , Γ_k , and Λ_k to be the variances of the $\underline{\delta}_k$, $\underline{\epsilon}_k$, and $\underline{\lambda}_k$. Let us define Φ_k as the variance of the composite error $\underline{\phi}_k$.

$$\Phi_k = \begin{cases} \Gamma_k & \text{for } k = 1, \dots, N \\ \Gamma_k + \mathbf{F}\Delta_k\mathbf{F}^T + \Lambda_k & \text{for } k = N + 1, \dots, N + C \end{cases} \quad (3.4)$$

Let us define Φ to be the $(N + C)q$ by $(N + C)q$ block diagonal matrix with the q by q diagonal blocks Φ_k .

$$\Phi = \begin{bmatrix} \Phi_1 & 0 & \cdots & 0 \\ 0 & \Phi_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & \Phi_{N+C} \end{bmatrix} \quad (3.5)$$

Then the variance $\mathbf{V}$ of the syndrome $\underline{s}$ is equal to

$$\mathbf{V} = \mathbf{W}\Phi\mathbf{W}^T = \sum_{m=1}^{N+C} \mathbf{W}_m\Phi_m\mathbf{W}_m^T \quad (3.6)$$

where $\mathbf{V}$ is a Cq by Cq block matrix.

Now, we are ready calculate the projection of the syndrome onto each weight vector. Let us define a $\mathbf{V}^{-1}$ norm as

$$\|\underline{v}\|_{\mathbf{V}^{-1}}^2 = \underline{v}^T \mathbf{V}^{-1} \underline{v} \quad (3.7)$$

Let us also define $\hat{\underline{\phi}}_k$ as the processor error estimate found by projecting the syndrome $\underline{s}$ onto the k^{th} weight vector with respect to the $\mathbf{V}^{-1}$ norm

$$\hat{\underline{\phi}}_k \leftarrow \min_{\underline{\phi}_k} \|\underline{s} + \mathbf{W}_k \bar{\underline{\phi}}_k\|_{\mathbf{V}^{-1}}^2 \quad (3.8)$$

where $\bar{\underline{\phi}}_k$ is the possible processor error. We use the $\mathbf{V}^{-1}$ norm to compensate for any correlations in the syndrome outputs. Solving for the minimum, we have

$$\hat{\phi}_k = - \left[\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k \right]^{-1} \mathbf{W}_k^T \mathbf{V}^{-1} \underline{s} \quad \text{for } k = 1, \dots, N + C \quad (3.9)$$

The $\hat{\phi}_k$ can be thought of as the k^{th} processor error that comes closest, in a least squares sense, to “explaining” the syndrome $\underline{s}$.

Let H_0 represent the no-fault hypothesis and H_k represent the hypothesis that processor k has failed. The most likely failure hypothesis $\hat{H}_k$ can be determined by computing the following.

$$\begin{aligned} L_0 &= \|\underline{s}\|_{\mathbf{V}^{-1}}^2 \\ L_k &= \min_{\phi_k} \|\underline{s} + \mathbf{W}_k \bar{\phi}_k\|_{\mathbf{V}^{-1}}^2 \quad \text{for } k = 1, \dots, N + C \end{aligned} \quad (3.10)$$

The L_k is equal to the best guess of the computation noise under hypothesis H_k . The L_k 's are called log likelihoods for reasons that will become clear when we later discuss the generalized likelihood ratio test which gives an identical result as the projection method. The most likely failure hypothesis $\hat{H}_k$ can be computed by

$$\hat{H}_k \leftarrow \min_k (L_k + \gamma_k) \quad (3.11)$$

where γ_k act as scalar thresholds. These thresholds are used to compensate for the different failure rate of the processors (the checksum processor failure rate would be higher than the data processor failure rate, since it includes the failure in input checksum and syndrome calculations). The L_k can be rewritten as

$$\begin{aligned} L_k &= \min_{\phi_k} \|\underline{s} + \mathbf{W}_k \bar{\phi}_k\|_{\mathbf{V}^{-1}}^2 \\ &= \left\| \left(\mathbf{I} - \mathbf{W}_k \left[\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k \right]^{-1} \mathbf{W}_k^T \mathbf{V}^{-1} \right) \underline{s} \right\|_{\mathbf{V}^{-1}}^2 \end{aligned} \quad (3.12)$$

$$= \|\underline{s}\|_{\mathbf{V}^{-1}}^2 - \|\mathbf{W}_k \bar{\phi}_k\|_{\mathbf{V}^{-1}}^2 \quad (3.13)$$

Using this, the equations for the log likelihoods can be simplified by defining the relative log likelihood L'_k to be

$$L'_k = L_0 - L_k + \gamma'_k \quad (3.14)$$

where $\gamma'_k = \gamma_0 - \gamma_k$. If L'_k is the largest, then this indicates that failure hypothesis H_k is most likely. The L'_k can be rewritten as

$$L'_0 = 0$$

$$L'_k = \|\mathbf{W}_k \bar{\phi}_k\|_{\mathbf{V}^{-1}}^2 + \gamma'_k \text{ for } k = 1, \dots, N + C \quad (3.15)$$

Notice that this is equivalent to doing the energy threshold test on $\mathbf{W}_k \hat{\phi}_k$, the projection of the syndrome onto the weight vector.

Once the fault has been determined to be in the k^{th} data processor, its correct output $\hat{\underline{y}}_k$ can be calculated by subtracting the projection $\hat{\phi}_k$ from faulty processor's erroneous output.

$$\hat{\underline{y}}_k = \underline{y}_k - \hat{\phi}_k \quad (3.16)$$

The numerical noise level in the corrected output would be significantly larger than other processors. This is because $\hat{\phi}_k$ contains not only the fault, but also the weighted sum of the numerical noise from all other processors. Therefore, $\hat{\underline{y}}_k$ is equal to the weighted sum of the numerical noise from all other processors.

The projection method for single fault location is equivalent to dividing the Cq dimensional syndrome space into $(N + C + 1)$ decision regions. Each decision region belongs to a failure hypothesis. If we let H_0 represent the no fault hypothesis and H_k represent the hypothesis that processor k has failed, the decision region of hypothesis H_0 would be around the origin, and the decision region of the hypothesis H_k would be around the k^{th} weight vector. For example, when two checksum processors are used to check two data processor with weight matrix

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & -1 \end{bmatrix} \quad (3.17)$$

the decision regions belonging to each failure hypothesis when $q = 1$ are plotted in the syndrome space in figure 3.1.

Setting the thresholds for single fault correction is very similar to the single fault detection case. For example, if we know the probability distribution of the normalized projection energy $\|\mathbf{W}_k \hat{\phi}_k\|_{\mathbf{V}^{-1}}^2$ under H_0 , then we can set the probability that L'_k exceeds zero under H_0 by adjusting γ'_k . The actual false alarm rate for the processor k is lower than that, since one has to account for the cases when the numerical noise causes more than one L'_k to be greater than zero. This happens because weight vectors are not orthogonal to each other.

As was the case in the single fault detection threshold method, the system's ability to detect a small transient fault or a small permanent fault depends very much on the output batch size q . The mean value of the normalized projection energy $\|\mathbf{W}_k \hat{\phi}_k\|_{\mathbf{V}^{-1}}^2$ grows as $O(q)$ where as its standard deviation grows as $O(\sqrt{q})$. Therefore, with a larger q the system is more likely to detect a small permanent failure affecting many of the output data points, but it is less likely to detect a transient failure affecting few of the output data points.

3.3.2 Effects of Incorrect Fault Diagnosis

The probability of making the correct fault diagnoses can be calculated by evaluating the following integral

$$\text{Prob}(\text{Guess } H_m \mid H_k \text{ is correct}) = \int_{H_m} p(\underline{s} \mid H_k, \bar{\phi}_k) d\underline{s} \quad (3.18)$$

which evaluates the probability that the syndrome under H_k would fall under the decision region H_m . This is a very difficult integral to evaluate.

In a single fault correction system with numerical noise, there are three types of incorrect fault diagnosis that can be made. The first is the false alarm case in which the large numerical noise in the system makes one or more L'_k exceed zero. The second type of false diagnosis is a small fault that is not detected by the system. The third type of false diagnosis happens when a small fault has occurred in one processor, but the numerical noise has pushed the syndrome closer to another weight vector, causing that processor to be diagnosed as faulty.

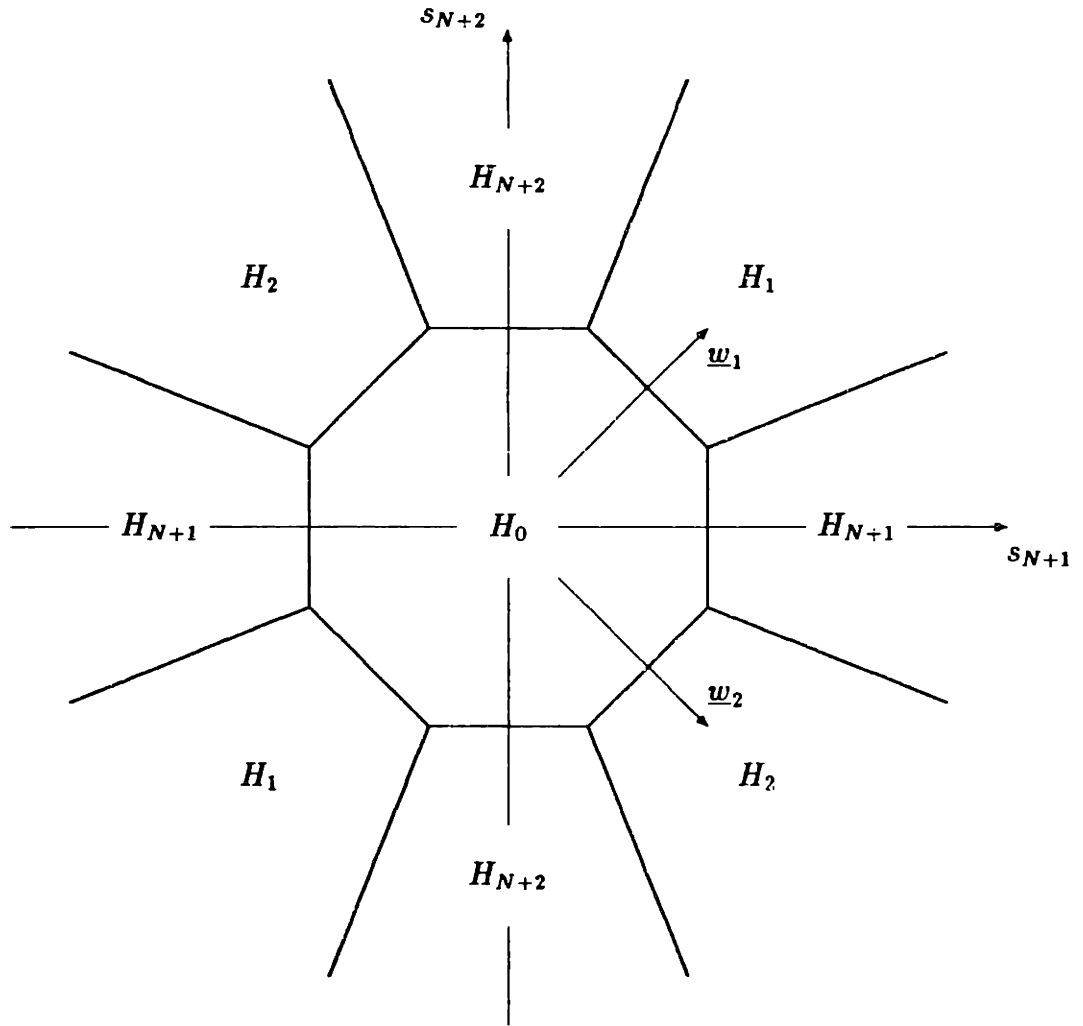


Figure 3.1: Projection Method Decision Regions, $N = 2, C = 2$

In the case of a false alarm, one of the processors would be incorrectly diagnosed to be faulty. If it happens to be a data processor, its output is corrected by subtracting $\hat{\phi}$. However, $\hat{\phi}$ under H_0 is equal to the weighted sum of the numerical noise from all the processors. Therefore, the corrected processor output now has its own numerical noise subtracted out, but contains the weighted sum of the numerical noise from all other processors. Furthermore, the numerical noise level in the system is much higher than the norm, since the false alarm is caused by a large numerical noise. Therefore, the numerical noise level in the corrected output is much higher than the noise level in the normal processor output.

In the case when the fault is small and does not get detected by the system, the effect on the system is equivalent to a slight increase in the noise level in the faulty processor. However, if the processor error energy level is too small to be detected, it should not affect the system performance greatly.

In the case when one processor has a small fault, but the numerical noise pushes the syndrome closer to another weight vector, the effect is similar to increasing the noise level in both processors. The faulty processor output would have a higher noise level than normal, since it contains a fault. The output of the processor that is falsely diagnosed to be faulty would be needlessly corrected. Therefore, the corrected output of that processor would have an increased noise level, similar to the false alarm case. Again, this type of false diagnosis is most likely only when the fault is small, thus the system performance should not be severely degraded. Notice this increase in noise level only applies to data processors. If a checksum processor is falsely diagnosed to be faulty, its output is not corrected. If the small non-detected fault were in a checksum processor, it does not affect the data processor outputs.

As one can see, the system is likely to make a false diagnosis only when the fault is relatively small, and the net effect of the false diagnosis is the increased noise level in some of the data processor outputs. One can also reduce the chances of misdiagnosis by choosing the weight vectors so that the angles between the weight vectors are as large as possible (i.e. spread them as far apart from each other as possible). When the angles between neighboring weight vectors are large, the decision regions associated with the weight vectors become "fatter," and there is less chance that the syndrome

of a small fault would be pushed closer to another weight vector line by the numerical noise. The larger angles between neighboring weight vectors also mean that they are more "orthogonal" to each other. Therefore, there is less cross-covariance between the syndromes and less chance of confusing a failure in one processor with a failure in another processor. The lower cross-covariance also lowers the false alarm rate since the numerical noise from other processors would not be weighted as heavily in the threshold tests. With a lower false alarm rate and a lower misdiagnosis rate, one can also afford to reduce the thresholds and increase the detection sensitivity of the small faults.

It is possible to make some tradeoffs between the chance of the occurrence of the different types of misdiagnosis by adjusting the γ'_k , thereby adjusting the size of the decision regions. For example, increasing the magnitude of one γ'_k decreases the size of that decision region and increases the sizes of all the neighboring decision regions. This decreases the probability of a false alarm in that processor but increases the probability of a false alarm in the processors of the neighboring decision regions. It also increases the size of the H_0 region, thereby increasing the chance that the small fault in that processor escapes detection. If we increase the magnitudes of all the γ'_k 's by the same amount, the decision regions would still have the same shape, but the size of H_0 would expand, causing less false alarms, but increasing the chance that a small fault may escape detection.

In deriving the projection method for the single fault correction system, we have assumed that there is no numerical noise involved in calculating L'_k 's. However, in real systems, there will most likely be additional computation noise in the calculation of the L'_k 's. This will increase the probability of fault misdiagnosis by the system, and also increase the numerical noise in the corrected processor output.

3.3.3 Dynamic Range vs. Numerical Noise

The dynamic range of the checksum processor is an important consideration in the single fault correction system. The dynamic range of the checksum processor k should be ω_k times larger than the data processors, where ω_k is defined in equation 3.3. Again, when floating point arithmetic is used, the dynamic range is not generally a big problem. However, when fixed point arithmetic is used, the k^{th} checksum processors should have $\log_2 \omega_k$ more

bits than the data processors. Otherwise, the weights should be chosen so that $\omega_k \leq 1$. Following is an example of a single fault correction weight matrix which meets such a condition.

$$\mathbf{W} = \begin{bmatrix} \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & 0 & 0 & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & -1 & 0 & 0 \\ \frac{1}{8} & -\frac{1}{8} & 0 & 0 & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & \frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & 0 & -1 & 0 \\ 0 & 0 & \frac{1}{8} & -\frac{1}{8} & \frac{1}{8} & -\frac{1}{8} & \frac{1}{8} & -\frac{1}{8} & \frac{1}{8} & -\frac{1}{8} & 0 & 0 & -1 \end{bmatrix} \quad (3.19)$$

However, using weights on the order of $1/N$ effectively weighs the numerical noise from the checksum processor approximately N times higher than the noise from the data processors. This means that the noise energy from the checksum processors will heavily mask the noise energy from the data processors in the threshold test, and thus the system would be less sensitive in detecting small failures in the data processors.

3.4 Simple Case

3.4.1 Simple Case of Projection Method

The computation of the projection method becomes especially simple when the variance matrix $\mathbf{V}$ is a scaled identity matrix. There are two sets of assumptions under which $\mathbf{V}$ is white.

The first set of assumptions is that there is no numerical error in input checksum calculations or in syndrome calculations, and that the weight matrix is of a specific form.

$$\begin{cases} \Gamma_k &= \sigma_\Gamma^2 \mathbf{I} \\ \Delta_k &= \underline{0} \\ \Lambda_k &= \underline{0} \\ \mathbf{W}\mathbf{W}^T &= \sigma_W^2 \mathbf{I} \end{cases} \quad (3.20)$$

In this case, $\mathbf{V}$ is equal to

$$\mathbf{V} = \sigma_V^2 \mathbf{I} \quad \text{where} \quad \sigma_V^2 = \sigma_W^2 \sigma_\Gamma^2 \quad (3.21)$$

For example, when fixed point arithmetic is used with low integer weights, the input checksumming noise and the syndrome calculation noise would be equal to zero and this condition can be satisfied.

The second set of assumptions is that all the computational noises are white, $\mathbf{F}$ is an orthogonal transform (an example of an orthogonal transform is a Fourier Transform), and $\mathbf{W}$ is of a specific form.

$$\begin{cases} \Gamma_k &= \sigma_\Gamma^2 \mathbf{I} \\ \Delta_k &= \sigma_\Delta^2 \mathbf{I} \\ \Lambda_k &= \sigma_\Lambda^2 \mathbf{I} \\ \mathbf{W}\mathbf{W}^T &= \sigma_W^2 \mathbf{I} \\ \mathbf{F}\mathbf{F}^T &= \sigma_F^2 \mathbf{I} \end{cases} \quad (3.22)$$

In this case, $\mathbf{V}$ is equal to

$$\mathbf{V} = \sigma_V^2 \mathbf{I} \quad \text{where} \quad \sigma_V^2 = \sigma_W^2 \sigma_\Gamma^2 + \sigma_F^2 \sigma_\Delta^2 + \sigma_\Lambda^2 \quad (3.23)$$

If one of the above two conditions are satisfied, $\hat{\phi}_k$ becomes

$$\hat{\phi}_k = -\frac{1}{r_{kk}} \sum_{m=N+1}^{N+C} w_{m,k} s_m \quad (3.24)$$

where r_{km} is equal to

$$r_{km} = \sum_{l=N+1}^{N+C} w_{l,k} w_{l,m} \quad (3.25)$$

The r_{km} can be thought of as the cross-correlation between the k^{th} and m^{th} weight vectors. When $k = m$, r_{km} can be thought of as the weight vector magnitude squared. The relative likelihood L'_k can be then written as

$$L'_k = \frac{1}{r_{kk} \sigma_V^2} \sum_{l=N+1}^{N+C} \sum_{m=N+1}^{N+C} w_{l,k} w_{m,k} s_l^T s_m + \gamma'_k \quad (3.26)$$

One needs approximately $(N+C)C(C+1)q/2$ multiply/adds if all the likelihoods are computed straightforwardly with this formula. The number of multiply/adds can be minimized if the L'_k 's are computed in the following way. Define $\rho_{l,m}$ as

$$\rho_{l,m} = \underline{s}_l^T \underline{s}_m \quad \text{for} \quad \begin{cases} l = N+1, \dots, N+C \\ m = N+1, \dots, N+C \end{cases} \quad (3.27)$$

Then L'_k for $k = 1, \dots, N$ can be calculated as

$$L'_k = \frac{1}{\tau_{kk}\sigma_V^2} (w_{N+1,k} \cdots w_{N+C,k}) \begin{bmatrix} \rho_{N+1,N+1} & \cdots & \rho_{N+1,N+C} \\ \rho_{N+2,N+1} & \cdots & \rho_{N+2,N+C} \\ \vdots & \vdots & \vdots \\ \rho_{N+C,N+1} & \cdots & \rho_{N+C,N+C} \end{bmatrix} \begin{pmatrix} w_{N+1,k} \\ w_{N+2,k} \\ \vdots \\ w_{N+C,k} \end{pmatrix} + \gamma'_k \quad (3.28)$$

and L'_k for $k = N+1, \dots, N+C$ can be calculated as

$$L'_k = \frac{1}{\tau_{kk}} \rho_{k,k} + \gamma'_k \quad (3.29)$$

One needs approximately $C(C+1)q/2$ multiply/adds to compute the $\rho_{l,m}$, and about $N(C+1)C/2$ multiply/adds to compute L'_k 's. This is substantially less than the direct calculation method. If one of the data processors is faulty, the correct value $\hat{\underline{y}}_k$ is calculated as

$$\hat{\underline{y}}_k = \underline{y}_k + \frac{1}{\tau_{kk}} \sum_{m=N+1}^{N+C} w_{m,k} \underline{s}_m \quad (3.30)$$

One needs approximately Cq multiply/adds to correct the faulty processor output.

Therefore, in order to detect and correct the fault in the simple white noise case, it takes approximately $(C+1)C(q+N)/2 + Cq$ multiply/adds. It also takes approximately CNp multiply/adds to compute the input checksums and CNq multiply/adds to compute the syndromes. Let us assume that the hardware module that detects and corrects the fault from the syndromes is triplicated, so that its failure rate does not dominate the system failure rate. In that case, the total number of multiply/adds needed for the single fault correction is equal to

$$CT + CN(p+q) + 3[(C+1)C(q+N)/2 + Cq] \quad (3.31)$$

where T is equal to the number of multiply/adds needed to compute F . The first term is the computation associated with the checksum processor computation, the second term reflects the input checksum and syndrome computations, and the third term is the work required by the triplicated fault detection/correction algorithm. We can define the computation overhead ratio R_c to be the ratio between the computations needed for fault tolerance overhead and the computations needed for the data processors. Therefore, we divide the overhead computation by NT to get R_c .

$$R_c = \frac{C}{N} + \frac{C(p+q)}{T} + 3 \left[\frac{(C+1)C(q+N)/2}{NT} + \frac{Cq}{NT} \right] \quad (3.32)$$

In general, R_c can be made lower by decreasing the number of checksum processors C and increasing the number of data processors N . The R_c is also lower when T/q is higher. When the computational effort involved in the computation of each of the output data points is high, the fault detection/correction effort becomes relatively smaller.

3.4.2 Computational Overhead Example

We can further reduce the computation by choosing weights to be small integers. When the small integers are used as weights, the number of multiplies needed is drastically reduced in input checksum and the syndrome computations as well as in the fault detection and correction procedure. Multiplication by weights 0, -1, +1, -2, +2, -1/2, +1/2, etc. is very simple, requiring much less computation than a full multiply. Let us consider the system with weight matrix in equation 2.31 with ten data processors and three checksum processors ($N = 10$, $C = 3$). Since only -1, 0, and +1 were used as the weights, there would be no multiplication involved with the input checksum and the syndrome calculations. These weight vectors are spread relatively far apart from each other minimizing the chances for incorrect fault diagnosis. Furthermore, the weight matrix satisfies $\mathbf{W}^T \mathbf{W} = 9\mathbf{I}$, one of the conditions required to achieve uncorrelated syndromes, and a simple relative likelihood test. However, this system does use one more checksum processor than the minimum number of the checksum processors required for single fault correction.

<u>Operation</u>	<u>Multiplies</u>	<u>Additions</u>
Input Checksums	0	$21q$
Syndromes	0	$24q$
Syndrome Correlations ρ_{1m}	$6q$	$6(q-1)$
Relative Likelihoods $\sigma_V^2 L'_k$	4	35
Choose largest likelihood	0	12
Correction of Failed Processor	q	$3q$

Table 3.1: Number of Multiplies and Adds

If we assume that all the numerical noises are white and $\mathbf{V} = \sigma_F^2 \mathbf{I}$, the number of adds and multiplies needed for input checksum and syndrome calculation and for the fault detection/correction algorithm are listed in table 3.1. We have not counted multiplication by $1/2$ or $-1/2$ (there are only 6 such operations). We have also used a recursive summation algorithm to compute L'_k , which cuts down on the number of additions. If we assume that the computation involved in the fault detection/correction algorithm is triplicated, the computational overhead ratio R_c for the multiplies becomes

$$R_c = 0.3 + \frac{3(7q + 4)}{10T} \quad (3.33)$$

and for the additions becomes

$$R_c = 0.3 + \frac{45q + 3(9q + 41)}{10T} \quad (3.34)$$

The factor 3 is there because the fault detection/correction computation must be triplicated.

Let us pick a complex FFT as the task $\mathbf{F}$. The FFT is a very realistic example of the task that the high performance DSP systems has to perform in real time. If we assume that all the numerical noises are white, then $\mathbf{V}$ is diagonal since the FFT is an orthogonal transform ($\mathbf{F}^T \mathbf{F} = \mathbf{I}$), and since $\mathbf{W}\mathbf{W}^T = \sigma_W^2 \mathbf{I}$. The complex FFT of length $q/2$ consists of $p = q$ input and output real data points. Computing $\mathbf{F}$ requires $2q \log_2(q/2)$ real multiplies and $3q \log_2(q) - q$ real additions. Therefore, the computational overhead ratio for the multiplies is equal to

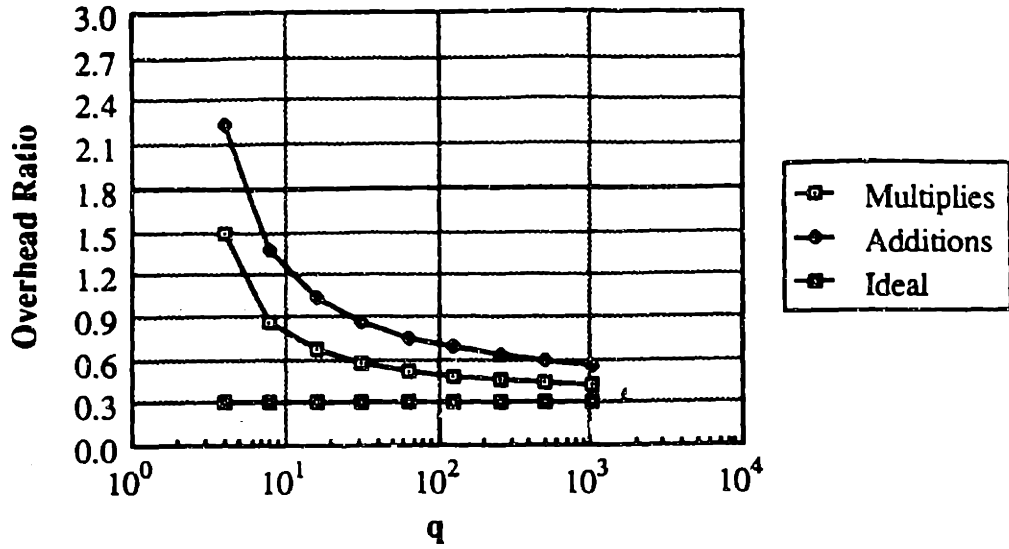


Figure 3.2: Overhead Ratio for Complex FFT (N=10, C=3)

$$R_c = 0.3 + \frac{21q + 12}{20q \log_2(q/2)} \quad (3.35)$$

and for the additions

$$R_c = 0.3 + \frac{72q + 123}{30q \log_2 q - 10q} \quad (3.36)$$

These overhead ratio curves are plotted in figure 3.2 with the "Ideal" overhead ratio accounting only for the checksum processor computations.

3.5 Other Methods Single Fault Correction Methods

3.5.1 Using Range Test on The Syndromes

There is another easy method for single fault correction when only -1, 0, and +1 are used as weights. It involves performing a range test on the individual syndromes, component by component. Fault diagnosis is

carried out component by component as if $q = 1$. Each component of each syndrome is classified into one of three ranges of values, and the code digit b_m is assigned to each syndrome depending on its range.

$$b_m = \begin{cases} +1 & \text{If } s_m < -\gamma \\ 0 & \text{If } -\gamma < s_m < +\gamma \\ -1 & \text{If } +\gamma < s_m \end{cases} \quad (3.37)$$

The γ is a scalar threshold. If there is no fault, all the b_m would be zero. If there is a fault, the faulty processor is the one with the weight vector $\pm(b_{N+1}, b_{N+2}, \dots, b_{N+C})$. Figure 3.3 shows the decision regions used when there are two data processors with two checksum processors with the weight vector matrix

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & -1 \end{bmatrix} \quad (3.38)$$

This range test method involves $2Cq$ comparison tests for each batch of data processing. The major advantage of this method is that it is very simple to implement and requires little computation with no multiplies.

The probability that a syndrome magnitude exceeds the threshold γ can be determined by integrating the probability distribution of the syndrome over the range $s_k < -\gamma$ and $s_k > \gamma$. Therefore, the false alarm rate can be set by adjusting γ as well. Notice that if the γ is not set large enough, it is possible to misdiagnose a checksum processor fault even if the fault is very large. This is because the checksum processor decision regions width does not increase with the fault size. In figure 3.3, one can see that the decision regions for the failure hypothesis H_{N+1} and H_{N+2} have constant widths. On the other hand, the projection method decision regions are all pie-shaped (getting wider as the fault size increases) as shown in figure 3.1, so that the probability of fault misdiagnosis decreases as the fault size increases. Consider the case when the checksum processor $N+1$ has a large fault, but the numerical error pushes the syndrome point from p_1 to p_2 as illustrated in figure 3.3. This causes the system to diagnose the processor 1 as being faulty. If s_{N+1} were used to correct the fault ($\hat{y}_1 = y_1 + s_{N+1}/w_{N+1,1}$), the corrected output $\hat{y}_1$ would be very wrong.

One way to get around this problem is to correct the data processor fault using the smallest syndrome with non-zero weight.

$$\hat{y}_k = y_k + \frac{\min(s_m)}{w_{m,k}} \quad \text{for } w_{m,k} \neq 0 \quad (3.39)$$

Another way to get around the problem is to increase the threshold γ , and widen the checksum processor decision regions $H_{N+1}, \dots, H_{N+C}$. Wider decision regions would lower the probability of misdiagnosis for the checksum processor fault. However, the larger γ would make it more difficult to detect a small processor fault. This could be a problem since detecting a small fault is difficult in the first place, due to the fact that the fault detection is done on a point by point basis (as if $q = 1$).

3.5.2 Error Coding Method

If only the 0 and 1 were used as weights with the range method discussed above, the method is equivalent to using error coding technique for single fault correction. In this case, it is possible to use $q > 1$. The syndrome energies are classified into two categories and the code digit b_m is assigned to each syndrome depending on the range.

$$b_m = \begin{cases} 1 & \text{If } \underline{s}_m^T \underline{s}_m > \gamma \\ 0 & \text{If } \underline{s}_m^T \underline{s}_m \leq \gamma \end{cases} \quad (3.40)$$

If there is no fault, all the b_m would be zero. If there is a fault, the faulty processor is the one with the weight vector $(b_{N+1}, b_{N+2}, \dots, b_{N+C})$. The major disadvantage of this method is that it requires more checksum processors than other methods. Another problem is that once again the decision regions for the checksum processors are narrow even when the fault is large. Therefore, one has to either avoid using the maximum syndrome for fault correction or increase the threshold γ and reduce the probability of misdiagnosis for the checksum processor fault as was the case in the previous range test method.

3.5.3 Angular Method

Another simple method for single fault correction is to use the slopes between the syndromes for the fault location. This method also requires that

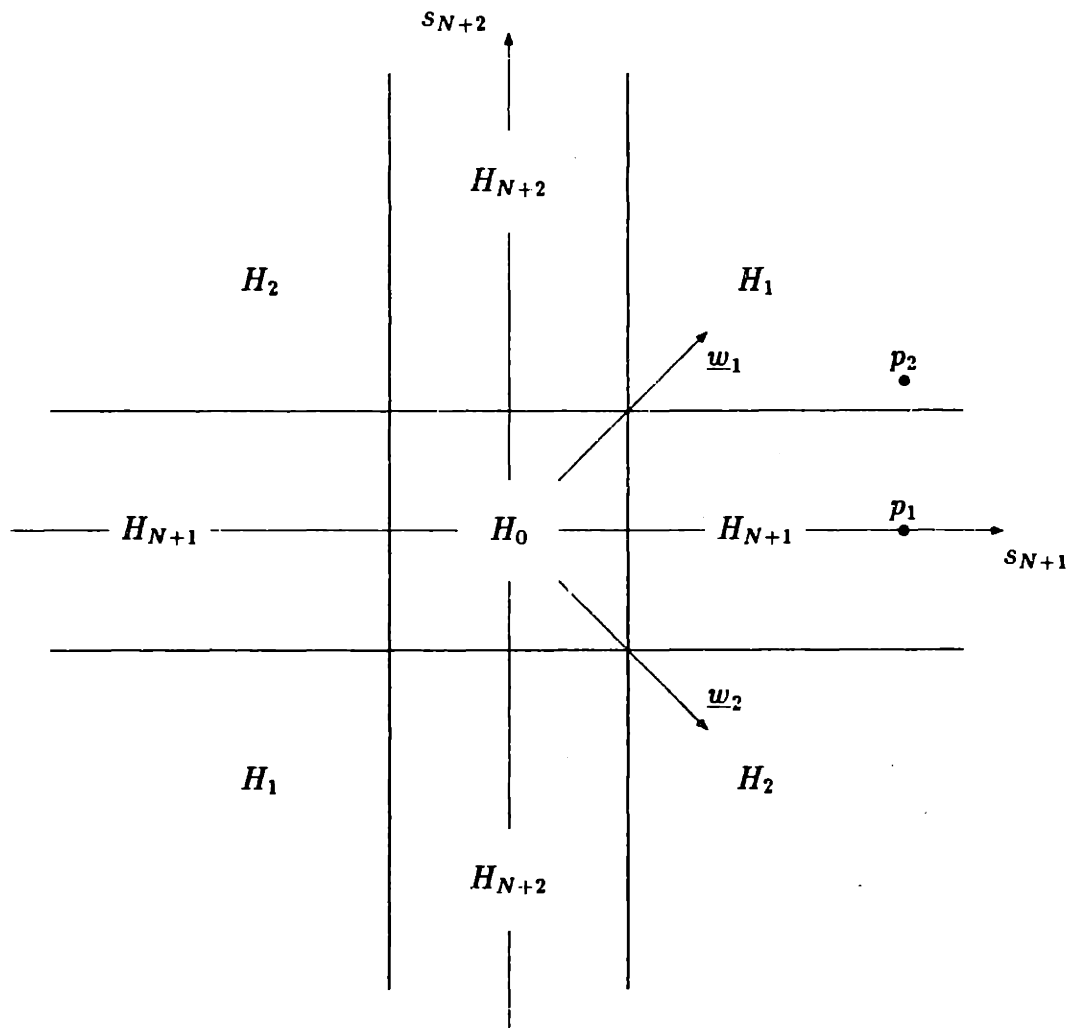


Figure 3.3: Range Test Method Decision Regions, $N = 2, C = 2$

the fault diagnosis is done point by point (i.e. as if $q = 1$). When there are only two checksum processors, the evenly spaced $(N + 2)$ weight vectors are chosen as

$$\begin{aligned}\underline{w}_{N+1,m} &= \cos\left(\frac{m\pi}{N+2}\right) \\ \underline{w}_{N+2,m} &= \sin\left(\frac{m\pi}{N+2}\right)\end{aligned}\tag{3.41}$$

where N is an even number. The weight vectors with $m = 0$ and $m = (N + 2)/2$ are the checksum processor weight vectors. The fault is detected if the energy of the syndromes $\sum_{m=N+1}^{N+2} s_m^2$ exceeds the threshold. Once a fault is detected, the faulty processor number m_f is equal to

$$m_f = \text{int}\left(\frac{N+2}{\pi} \arctan\left(\frac{s_{N+2}}{s_{N+1}}\right)\right)\tag{3.42}$$

where the function $\text{int}()$ is the roundoff function to the nearest integer. The decision regions of this method are drawn in figure 3.4. In actual implementation, one may want to do the range test on the slope s_{N+2}/s_{N+1} instead of computing explicit $\arctan$. This would require $\log_2(N + C)$ threshold tests on the slope. Since computing the slope requires a division which is computationally costly, the slope threshold test may be carried out by multiplying s_{N+1} by the test slope and comparing with s_{N+2} . In the systems with more than 2 checksum processors, the angles between $(C - 1)$ syndrome pairs have to be calculated.

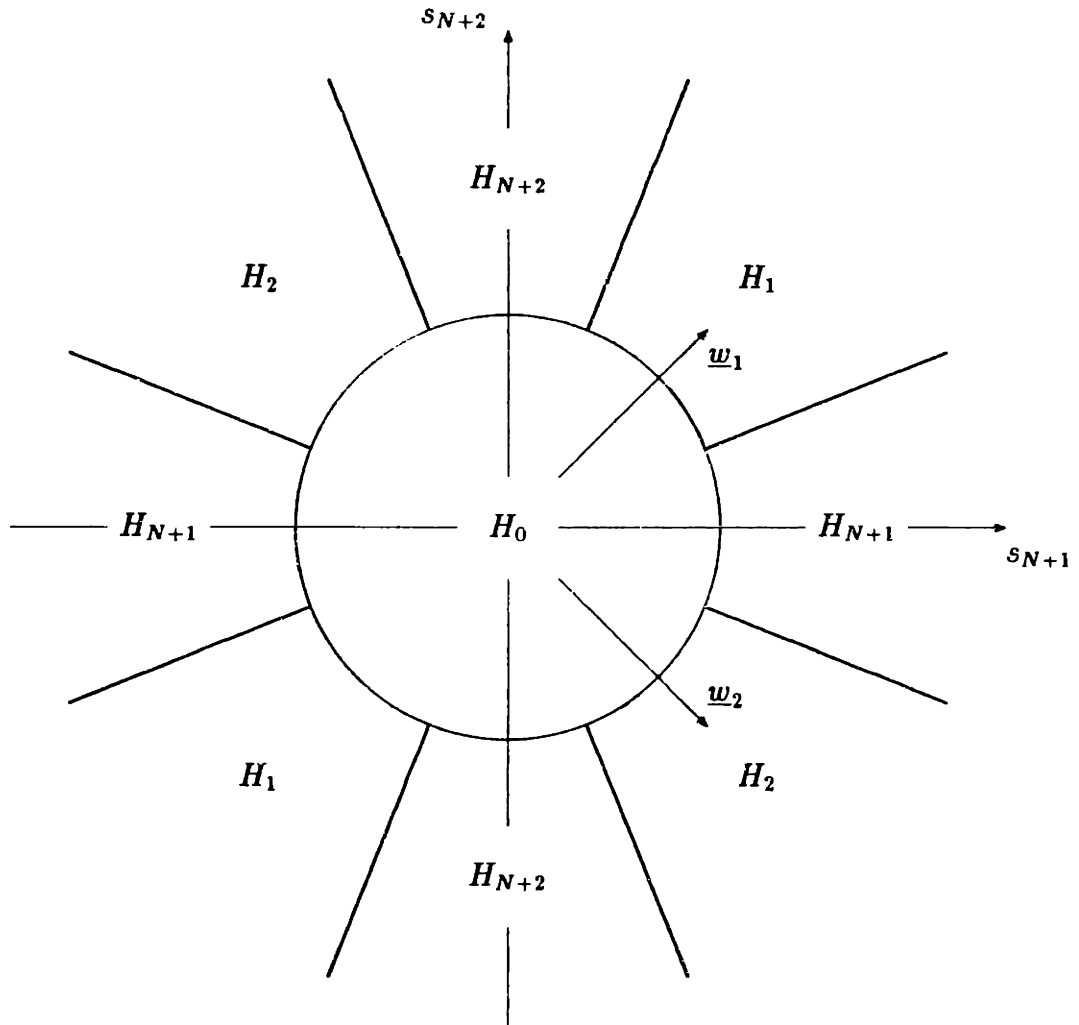


Figure 3.4: Angular Method Decision Regions, $N = 2$, $C = 2$

Chapter 4

Generalized Likelihood Ratio Test

4.1 Single Fault Correction

Musicus and Song [Musicus 88, Song 87] have proven that when the numerical noises are of Gaussian type, the projection method we have used for single fault correction is equivalent to a generalized likelihood ratio test which gives a “optimum” or “near optimum” performance. The detailed proof in their paper is given in appendix B, and the following is the summary of their derivation.

Assuming that processor failures occur independently from each other, with P_k equal to the probability of failure of processor k , then the probability of failure hypothesis H_k is equal to

$$\begin{aligned} p(H_0) &= \prod_{m=1}^{N+C} (1 - P_m) \\ p(H_k|\phi_k) &= P_k \prod_{\substack{m=1 \\ m \neq k}}^{N+C} (1 - P_m) \end{aligned} \quad (4.1)$$

Note that the probabilities of the hypothesis that are considered do not add exactly to 1 ($\sum_{k=0}^{N+C} < 1$). This is because only the no-fault and single

fault cases are considered. The assumption here is that $P_k \ll 1$ and that the probability of multiple faults occurring is negligible compared to the probability of a single fault occurring. The P_k of the checksum processors include the probability of failure of the input checksum calculation and the syndrome calculation.

Under H_0 , the syndromes $\underline{s}$ are modeled as zero-mean Gaussian random variables with variance $\mathbf{V}$

$$p(\underline{s}|H_0) = N(\underline{0}, \mathbf{V}) \quad (4.2)$$

where

$$\mathbf{V} = \mathbf{W}\Phi\mathbf{W}^t = \sum_{m=1}^{N+C} \mathbf{W}_m \Phi_m \mathbf{W}_m^T \quad (4.3)$$

as defined before.

Under presence of a processor fault, the model is modified as follows. Under H_k , ϕ_k is modeled as a Gaussian random variable with an unknown mean $\bar{\phi}_k$ and known covariance $\bar{\Phi}_k$. The $\bar{\phi}_k$ is to be estimated from the syndrome. This estimation is necessary because we do not have the probability distribution of the fault available. With this model for ϕ_k under H_k , the probability distribution of the syndrome $\underline{s}$ under H_k is

$$\begin{aligned} p(\underline{s} | H_k, \bar{\phi}_k) &= N \left(-\mathbf{W}_k \bar{\phi}_k, \sum_{\substack{m=1 \\ m \neq k}}^{N+C} \mathbf{W}_m \Phi_m \mathbf{W}_m^T + \mathbf{W}_k \bar{\Phi}_k \mathbf{W}_k^T \right) \\ &= N \left(-\mathbf{W}_k \bar{\phi}_k, \mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T \right) \end{aligned} \quad (4.4)$$

where $\Delta \Phi_k = \Phi_k - \bar{\Phi}_k$ is the difference between the k^{th} processor's working covariance and the failure covariance.

A Generalized Likelihood Ratio Test (GLRT) is then used to find the most likely failure hypothesis which would explain the values of the syndrome $\underline{s}$. Let L_k be the log likelihood of $\underline{s}$ and hypothesis H_k .

$$L_0 = \log p(\underline{s}, H_0)$$

$$L_k = \max_{\bar{\phi}_k} \log p(\underline{s}, H_k | \bar{\phi}_k) \text{ for } k = 1, \dots, N + C \quad (4.5)$$

Notice that the L_k for $k = 1, \dots, N + C$ must be maximized over the failure size $\bar{\phi}_k$. This is because the actual processor error is not known and must be estimated from the syndrome values. Let $\hat{\bar{\phi}}_k$ be the value that maximizes the k^{th} likelihood. The $\hat{\bar{\phi}}_k$ can also be thought of as the most likely value of the processor error that would have caused the syndrome values to be as they are, if the processor k were faulty.

The $\hat{\bar{\phi}}_k$ can be found by using Bayes' Rule, substituting Gaussian densities into the likelihood formulas, and then solving for the maximum L_k with respect to $\bar{\phi}_k$. The resulting $\hat{\bar{\phi}}_k$ is exactly the same as the projection of the syndrome $\underline{s}$ onto the k^{th} weight vector direction with respect to the $\mathbf{V}^{-1}$ norm as we found in the projection method.

$$\hat{\bar{\phi}}_k \leftarrow \min_{\bar{\phi}_k} \|\underline{s} + \mathbf{W}_k \bar{\phi}_k\|_{\mathbf{V}^{-1}}^2 \quad (4.6)$$

or

$$\hat{\bar{\phi}}_k = -[\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k]^{-1} \mathbf{W}_k^T \mathbf{V}^{-1} \underline{s} \text{ for } k = 1, \dots, N + C \quad (4.7)$$

Substituting this into L_k ,

$$\begin{aligned} L_0 &= -\frac{1}{2} \|\underline{s}\|_{\mathbf{V}^{-1}}^2 + \gamma_0 \\ L_k &= -\frac{1}{2} \|\underline{s} + \mathbf{W}_k \hat{\bar{\phi}}_k\|_{\mathbf{V}^{-1}}^2 + \gamma_k \text{ for } k = 1, \dots, N + C \end{aligned} \quad (4.8)$$

where the γ_k constants do not depend on $\underline{s}$. If we define relative likelihood L'_k (relative to H_0) as

$$L'_k = 2(L_k - L_0) \quad (4.9)$$

then we have

$$L'_0 = 0$$

$$L'_k = \|\mathbf{W}_k \hat{\Phi}_k\|_{\mathbf{V}^{-1}}^2 + \gamma'_k \text{ for } k = 1, \dots, N + C \quad (4.10)$$

where

$$\gamma'_k = -\log |\mathbf{I} - \mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k \Delta \Phi_k| + 2 \log \left(\frac{P_k}{1 - P_k} \right) \quad (4.11)$$

These formulas are exactly what we derived in the projection method section except for the fact that the likelihood ratio test arrives at certain fixed constants for the thresholds.

Although the generalized likelihood ratio test is designed to give the most likely failure hypothesis for the syndromes with these constants, it is not clear that these constants are appropriate. The reason is that the processor failure model was not known in advance and thus the failure size had to be jointly estimated along with the hypothesis from the syndromes. This is more of an ad hoc likelihood ratio test in which some the probabilities are known in advance, but the parameters have to be guessed. Therefore, although the form of the likelihood equations is desirable, their constants may not be the most useful. The fact that the GLRT does not give the exact constants is not very relevant in application, since we would like to exploit the selection of constants to achieve the desired tradeoff between various probabilities of misdiagnosis.

4.2 Reducing The Numerical Noise

So far, we have used syndromes exclusively for fault detection/correction purposes. However, in some cases, it is possible to also use the syndromes for reducing the numerical noise in the outputs of the data processors. The derivation of the method involves a modified generalized likelihood ratio test derived by Musicus and Song [Musicus 88, Song 87]. The detailed proof in their paper is given in appendix C, and the following is the summary of their derivation. In the modified generalized likelihood ratio test, the outputs of the data processors as well as the syndromes are used in order

not only to detect and correct the faulty processor, but also to estimate the correct output of the working data processors and thus possibly reduce the noise level in the outputs.

Let us define $\underline{\bar{y}}_k$ as what the k^{th} processor output should be.

$$\underline{\bar{y}}_k = \mathbf{F} \underline{x}_k \quad (4.12)$$

Let us also define $\underline{y}$ as the block vector of the actual processor outputs and $\underline{\bar{y}}$ as the block vector of what the correct output should be. These Nq length block vectors are

$$\underline{y} = \begin{pmatrix} \underline{y}_1 \\ \vdots \\ \underline{y}_N \end{pmatrix}, \quad \underline{\bar{y}} = \begin{pmatrix} \underline{\bar{y}}_1 \\ \vdots \\ \underline{\bar{y}}_N \end{pmatrix} \quad (4.13)$$

The log likelihood L_k in the modified generalized likelihood ratio test are given as

$$L_0 = \max_{\underline{\bar{y}}} \log p(\underline{y}, \underline{s}, H_0 | \underline{\bar{y}}) \quad (4.14)$$

$$L_k = \max_{\underline{\bar{y}}} \max_{\underline{\bar{\phi}}_k} \log p(\underline{y}, \underline{s}, H_k | \underline{\bar{y}}, \underline{\bar{\phi}}_k) \text{ for } k = 1, \dots, N + C \quad (4.15)$$

The L_k 's are maximized over the correct output $\underline{\bar{y}}$ and the processor error $\underline{\bar{\phi}}$. Let $\hat{\underline{\bar{y}}}$ and $\hat{\underline{\bar{\phi}}}_k$ be the values of $\underline{\bar{y}}$ and $\underline{\bar{\phi}}_k$ that maximize L_k . The $\hat{\underline{\bar{y}}}$ and $\hat{\underline{\bar{\phi}}}_k$ can also be thought of as the most likely values of the correct outputs and the processor error that would cause the observed syndromes, the processor outputs, and the failure H_k . The $\hat{\underline{\bar{y}}}$ and $\hat{\underline{\bar{\phi}}}_k$ are again calculated by using Bayes' rule, substituting Gaussian densities, and solving for the maximum. The most likely $\hat{\underline{\bar{\phi}}}_k$ is given by

$$\hat{\underline{\bar{\phi}}}_k \leftarrow \min_{\underline{\bar{\phi}}_k} \|\underline{s} + \mathbf{W}_k \underline{\bar{\phi}}_k\|_{\mathbf{V}^{-1}}^2 \quad (4.16)$$

or

$$\hat{\underline{\bar{\phi}}}_k = -[\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k]^{-1} \mathbf{W}_k^T \mathbf{V}^{-1} \underline{s} \text{ for } k = 1, \dots, N + C \quad (4.17)$$

exactly as in the previous likelihood ratio test. Under the hypothesis H_0 , $\hat{\underline{y}}$ is equal to

$$\hat{\underline{y}}_m = \underline{y}_m + \Phi_m \mathbf{W}_m^T \mathbf{V}^{-1} \underline{s} \quad \text{for } m = 1, \dots, N \quad (4.18)$$

and under the hypothesis H_k for $k = 1, \dots, N + C$, $\hat{\underline{y}}$ is equal to

$$\hat{\underline{y}}_m = \begin{cases} \underline{y}_m + \Phi_m \mathbf{W}_m^T \mathbf{V}^{-1} (\underline{s} + \mathbf{W}_k \hat{\underline{\phi}}_k) & \text{for } m = 1, \dots, N, m \neq k \\ \underline{y}_k - \hat{\underline{\phi}}_k & \text{for } m = k, (1 \leq m \leq N) \end{cases} \quad (4.19)$$

The estimate $\hat{\underline{y}}_k$ of the non-faulty processor is the sum of the processor output $\underline{y}_k$ and the "adjustment term" calculated from the syndromes for reducing the numerical noise.

Substituting these values into L_k gives exactly the same result as the former likelihood calculation but with different constants γ_k . Converting L_k into relative likelihood L'_k , the constants γ'_k are now equal to

$$\gamma'_k = -\log |\bar{\Phi}_k \Phi_k^{-1}| + 2 \log \left(\frac{P_k}{1 - P_k} \right) \quad (4.20)$$

Therefore, this modified likelihood ratio test works exactly like the former likelihood ratio test and projection method but with different threshold constants. However, it also enables us to estimate the working processors' correct output values, as well as the failed processor's correct output values.

Appendix E shows that the numerical noise in $\hat{\underline{y}}_k$ has zero mean. Intuitively, the variance of $\hat{\underline{y}}_k$ should be reduced by a factor of $O(C/N + C)$, provided that there is no numerical noise involved in the noise reduction process.

Although the fact that the correct processor output can be estimated is theoretically interesting, it may not be very useful in many practical cases. First, $O(C/(N + C))$ reduction is not that significant, since we usually want to keep C small compared to N . Such small improvement in noise level may not justify the increase in the computational costs, especially if the computation has to be done in the Triple Modular Redundancy form. Moreover, additional numerical error incurred during the noise reduction computation may reduce the benefits even further. In cases when the magnitude of some of the weights are significantly larger than some others (as

was the case when the checksum processor dynamic range was same as the data processors), the noise from some processors would be weighted very heavily, and attempting to reduce the numerical noise using the syndromes may actually increase the noise level in the outputs of the processors with the small weights.

Chapter 5

Fixed Point System Simulations

5.1 Numerical Noise

In deriving the single fault correction methods in the presence of numerical noise, the projection method and the generalized likelihood ratio test yielded the same result. The k^{th} relative log likelihood in these methods consists of the normalized energy in the projection of the syndrome $\underline{s}$ onto the k^{th} weight vector with respect to the variance V , plus the threshold constant γ'_k . Although the likelihood ratio test theory suggests a specific threshold γ'_k for each relative log likelihood, these suggested values are not necessarily appropriate for all applications. The major reason is that the probability distribution of the fault was not known for the likelihood ratio test. We had to make an ad hoc modification to the test so that the fault had a Gaussian distribution with a mean which had to be estimated from the syndromes. Therefore, although the form of the log likelihoods are intuitively reasonable, the specific threshold constants may not be so accurate.

One also wants to be able to change the values of the thresholds to make tradeoffs between the probabilities of different fault misdiagnosis. For example, one would want to make a tradeoff between the probability of fault detection and the probability of false alarm. This can be achieved through appropriate changes in γ'_k 's.

Another reason why the thresholds derived in the generalized likelihood ratio test may not be appropriate is that the derivation assumes the numerical noise to be Gaussian. Numerical noises are often non-Gaussian, and the probability distribution of the numerical noises usually depends on the processing tasks. Although we expect the numerical noise distribution from the processors to look similar to a Gaussian distribution in most cases, some processing tasks may generate numerical noise whose probability distribution deviates significantly from Gaussian. Even for the cases in which the noise distribution is close to Gaussian, the probability distribution often differs substantially from Gaussian in the tails. For example, the actual noise probability distribution is zero in the range beyond the maximum possible numerical noise value. These deviations in the tail ends can affect the probability of fault detection and the probability of false alarm significantly. Because the thresholds are set to discriminate between high noise and failure, the shape of the noise distribution in the tails is crucial to setting an appropriate threshold for balancing the various probabilities of misdiagnosis. These probability distribution deviations can, however, be compensated for by appropriate choices of the threshold γ'_k .

In this section we shall use simulations to study the workings of the projection fault detection/correction method and the adjustments one needs to make to the γ'_k in real fixed point systems. First, we shall simulate the projection method in an example fixed point single fault correction system assuming all the numerical noises are Gaussian. The variables that were difficult to predict accurately, such as the probability of false alarm and the numerical noise in the corrected processor output were observed. We also examine how the fault detection/correction algorithm is dependent on various parameters such as the batch size, the processor reliabilities, and the noise level.

Second, we model fixed point roundoff as a statistical process and simulate the numerical noise probability distribution in the syndromes. We shall show how the tail ends of such a probability distribution deviate from Gaussian and show how γ'_k can be adjusted to compensate for these deviations.

Third, we shall examine the probability distribution of the numerical noises in the syndromes of real systems by simulating a fixed point single fault detection system. The purpose of this simulation is to see whether the roundoff operations are indeed independent statistical processes as was as-

sumed in the previous simulation. The fixed point Finite Impulse Response Filter and Fast Fourier Transform were the computational tasks simulated. We shall examine how the numerical noise probability distribution deviates from theory in real systems, and how one can compensate by adjusting γ'_k . Even though our study is limited to fixed point systems, similar studies can be done on floating point systems as well.

5.2 Single Fault Correction System Simulation

In this section, we have simulated a fixed point single fault correction system using the projection method in order to observe the variables that are difficult to predict analytically such as the false alarm rate. In the first simulation, we simulate a fixed point single fault correction system for a large number of batches in order to collect an accurate histogram. As predicted, the actual false alarm rate is less than $p(L'_k > 0 | H_0)$, the probability that L'_k exceeds zero, and the processor false alarm rate depends on the size and the shape of the decision region. The numerical noise filtering algorithm in section 2.2 was also applied in this simulation and found that the noise reduction rate is very close to the predicted value.

In the second simulation, we simulated the same system using different batch sizes q . We found a slight increase in the false alarm rate with increasing q , which was not predicted analytically. However, this dependence on q was found to be weak. The numerical noise in the corrected output in the case of a false alarm decreases with increasing q , as predicted analytically.

In the third simulation, we simulated the same system using different $p(L'_k > 0 | H_0)$ (i.e. different γ'_k). We found that the processor false alarm rate is closer to $p(L'_k > 0 | H_0)$ for smaller values of $p(L'_k > 0 | H_0)$. However, again this dependence was weak. The numerical noise in the corrected output in the case of a false alarm increases with decreasing $p(L'_k > 0 | H_0)$, as predicted analytically.

Therefore, we conclude that the false alarm rate $p(\text{Choose } H_k | H_0)$ is less than $p(L'_k > 0 | H_0)$ and the difference is dependent on q and the value of $p(L'_k > 0 | H_0)$. However, these dependencies are weak and the false alarm

rate $p(\text{Choose } H_k | H_0)$ and $p(L'_k > 0 | H_0)$ are within an order of magnitude. Therefore, it is good practice to use $p(L'_k > 0 | H_0)$ as the desired false alarm rate in designing systems.

5.2.1 Simulated System

The system simulated is a fixed point single fault correction system, based on a weighted checksum and the projection method. It consists of 10 data processors and 3 checksum processors ($N = 10$, $C = 3$) with the weighting matrix given in equation 2.31. All the numerical noises are assumed to be white Gaussian random variables. We shall first derive the appropriate values for γ'_k and then simulate the projection fault detection/correction method to observe the variables that are difficult to predict accurately, such as the probability of false alarm and the numerical noise in the corrected processor output. We also examine how the fault detection/correction algorithm is dependent on various parameters such as the batch size, the processor reliabilities, and the noise level.

Since all the weights are -1, 0, and +1, there is no numerical noise involved in the input checksum calculation or in the syndrome calculation. This means that

$$\begin{aligned}\Gamma_k &= \sigma_\Gamma^2 \mathbf{I} \\ \Delta_k &= \mathbf{0} \\ \Lambda_k &= \mathbf{0}\end{aligned}\tag{5.1}$$

Since $\mathbf{W}\mathbf{W}^T = \sigma_W^2$, according to section 3.4.1, the variance of the syndrome is equal to

$$\mathbf{V} = \sigma_V^2 \mathbf{I} \quad \text{where} \quad \sigma_V^2 = \sigma_W^2 \sigma_\Gamma^2\tag{5.2}$$

Now the relative log likelihood can be rewritten as

$$L'_k = \frac{1}{r_{kk}\sigma_V^2} \sum_{l=N+1}^{N+C} \sum_{m=N+1}^{N+C} w_{l,k} w_{m,k} \underline{s}_l^T \underline{s}_m + \gamma'_k\tag{5.3}$$

This equation can also be written as

$$L'_k = \frac{1}{\sigma_W^2 \sigma_V^2} \sum_{j=1}^q \left| \sum_{m=N+1}^{N+C} w_{m,k} s_{m,j} \right|^2 + \gamma'_k \quad (5.4)$$

The expected value of the relative log likelihood under hypothesis H_0 is equal to

$$\begin{aligned} E(L'_k) &= \frac{1}{\sigma_W^2 \sigma_V^2} q \sigma_W^2 \sigma_V^2 + \gamma'_k \\ &= q + \gamma'_k \end{aligned} \quad (5.5)$$

which is consistent with the result in appendix D. The variance under H_0 can be calculated as

$$\begin{aligned} \text{Var}(L'_k) &= E(L_k'^2) - (E(L'_k))^2 \\ &= \left(\frac{1}{\sigma_W^2 \sigma_V^2} \right)^2 2q (E(|\sum_{m=N+1}^{N+C} w_{m,k} s_{m,j}|^2))^2 \\ &= 2q \end{aligned} \quad (5.6)$$

The first term of equation 5.4 (all except the term γ'_k) is equivalent to the sum of the squares of the Gaussian variables $\sum_{m=N+1}^{N+C} w_{m,k} s_{m,j}$ normalized by its variance. It is thus equivalent to the sum of the squares of q unit variance Gaussian variables. The probability distribution function for the sum of the squares of q unit variance Gaussian variables is the chi-square probability function with q degrees of freedom.

The chi-square probability function is defined as follows. Suppose $X_1, X_2, \dots, X_q$ are independent Gaussian random variables with zero mean and unit variance. Then $X^2 = \sum_{i=1}^q X_i^2$ is said to follow the chi-square distribution and the probability that X^2 exceeds χ^2 is given by

$$p(X^2 > \chi^2 | q) = [2^{q/2} \Gamma(\frac{q}{2})]^{-1} \int_{\chi^2}^{\infty} (t)^{q/2-1} e^{-t/2} dt \quad (5.7)$$

The expected value of X^2 is equal to q . With large q , chi-square distribution can be approximated by Gaussian distribution of mean q and variance $2q$.

Since the probability distribution of L'_k is equal to a chi-square distribution, we can choose γ'_k in order to set $p(L'_k > 0 | H_0)$, the probability that L'_k exceeds zero. The γ'_k in this case is equal to the negative of the value χ^2 needed to achieve the probability $p(X^2 > \chi^2 | q) = p(L'_k > 0 | H_0)$.

$$\gamma'_k = -\chi^2 \quad \text{for} \quad p(X^2 > \chi^2 | q) = p(L'_k > 0 | H_0) \quad (5.8)$$

Values for γ'_k can be calculated from chi-square distribution function tables.

5.2.2 The False Alarm Rate

Although we can calculate γ'_k and set $p(L'_k > 0 | H_0)$, the probability that the log likelihood exceeds zero under the no-fault condition, we cannot easily calculate $p(\text{Choose } H_k | H_0)$, the actual probability that a false alarm will occur in that processor. This is because the weight vectors are not orthogonal, the likelihoods are correlated with each other, and the large numerical noise can cause more than one relative log likelihood to exceed zero. The probability of false alarm can be calculated by evaluating

$$p(\text{false alarm}) = \sum_{m=1}^{N+C} \int_{H_m} p(\underline{s} | H_0) d\underline{s} \quad (5.9)$$

which expresses the probability that the syndrome under H_0 will fall in other decision regions. However, calculating this integral is non-trivial, and we have instead determined the probability of false alarm by simulation.

The false alarm rate is an important design parameter, since a false alarm can cause a data processor output to be needlessly corrected, and the numerical noise in the corrected output is most likely much larger than the processor numerical noise. There is also a tradeoff between the false alarm rate and the size of the fault that can escape detection.

In these simulations, only the no-fault cases (H_0 cases) are simulated because we do not have a reliable model for processor failures. If a fault model were available, we could have simulated the probability of fault misdiagnosis. However, the probability of fault misdiagnosis is most likely much lower than the false alarm rate for the following reason. Only a small fault is likely to escape detection or to be misdiagnosed as some other processor's fault. Since the probability of a fault occurring is very low and the probability of a fault being small is most likely very low, the probability of a small fault occurring is very low indeed. Therefore, the probability of fault misdiagnosis is most likely much lower than the false alarm rate, and thus is not a very meaningful design parameter. Besides, the effect of the

small fault in either misdiagnosis case is equivalent to an increase in the noise level in some of the processors, and is usually not fatal to the system application.

Instead of simulating a particular processing task, we assumed that all inputs are equal to zero ($\underline{x}_k = \underline{0}$). Thus, all the processors generate signals $\underline{y}_k = \underline{0}$ with added white Gaussian noise with variance σ_1^2 as outputs. Each syndrome is equal to a weighted sum of the processor noises and is zero mean because of the no-fault condition. The white Gaussian noise was generated using pseudo random numbers. From the syndromes, the relative log likelihoods L'_k 's were calculated. The largest relative likelihood was then found in order to classify the fault. If all were negative, then we concluded that there is no fault.

In these experiments, the thresholds γ'_k are set such that the probabilities $p(L'_k > 0 | H_0)$ are fairly high (around the 0.01 to 0.1 range rather than the more realistic 10^{-30} to 10^{-20} range) so that we can get meaningful histograms without doing the simulation for an enormous number of repetitions. All $p(L'_k > 0 | H_0)$ were set to an identical value in all processors. Therefore, all γ'_k were set to an identical value as well.

Table 5.1 shows the simulation result histogram for 500,000 trials with $\sigma_1 = 10$ lsb., $q = 10$ and the $p(L'_k > 0 | H_0) = 0.1$. As one can see, $\#(L'_k > 0 | H_0)$, the number of times that each relative likelihood exceeds zero, is very close to the predicted value 50,000. This also corresponds to the fact that $\langle L'_k \rangle$, the average value of relative likelihood, is very close to -5.987, as predicted by equation 5.5. However, $\#(\text{Choose } H_k | H_0)$, the number of false alarms on processor k , is around 17,305 for $k = 1, \dots, 6$, around 16,334 for $k = 6, \dots, N$, and around 20,339 for $k = N + 1, \dots, N + C$.

The reason is that the weight vectors have different sets of neighboring weight vectors and differently shaped decision regions. By neighbors we mean abutting decision regions. (The fault location algorithm is most likely to confuse between the faults in neighboring decision regions.) When the angles between neighboring weight vectors are small, the decision region is "skinny", and has a smaller probability of correctly diagnosing the failure and a smaller likelihood of picking this vector when H_0 is true. Therefore, one would expect that the closer the neighboring weight vectors are, the greater the difference between the actual false alarm rate $p(\text{Choose } H_k | H_0)$ and the probability $p(L'_k > 0 | H_0)$. Table 5.2 has the angles between the neighboring weight vectors. The $\theta_{k,m}$, the angle between weight vectors k

k	$\#(\text{Choose } H_k \mid H_0)$	$\#(L'_k > 0 \mid H_0)$	$\langle L'_k \rangle$
0	269,820	0	0.000000e+00
1	17,049	49,933	-5.974040e+00
2	17,379	50,172	-5.982454e+00
3	17,467	50,513	-5.976576e+00
4	17,444	50,595	-5.966286e+00
5	17,202	50,158	-5.984101e+00
6	17,286	50,366	-5.977914e+00
7	16,383	50,002	-5.979583e+00
8	16,327	50,438	-5.968598e+00
9	16,267	50,218	-5.978332e+00
10	16,360	50,345	-5.981068e+00
11	20,163	50,181	-5.987824e+00
12	20,348	50,285	-5.968671e+00
13	20,505	50,426	-5.974191e+00

Table 5.1: Simulation Histogram of Single Fault Correction (N=10, C=3)

and m , is defined as

$$\theta_{k,m} = \cos^{-1} \left(\frac{\underline{w}_k^T \underline{w}_m}{\|\underline{w}_k\| \|\underline{w}_m\|} \right) \quad (5.10)$$

Weight vectors $k = 1, \dots, 6$ each have two weights of ± 1 and one zero and have two neighboring vectors of 35.3 degrees and two neighboring vectors at 45 degrees. The weight vectors for $k = 6, \dots, N$ each have three ± 1 coefficients with three neighboring vectors of 35.3 degrees. The weight vectors of $k = N + 1, \dots, N + C$ (the checksum processors) each have only one -1 coefficient and two zero coefficients and have four nearest neighbors at 45 degrees. Thus if all threshold constants γ'_k are equal, then decision regions $H_{N+1}, \dots, H_{N+C}$ are the “fattest” and the most likely to be confused with H_0 ; regions $H_1, \dots, H_6$ are next in size and have medium false alarm rates, while regions $H_7, \dots, H_N$ are the skinniest and have the least false alarm rate. This is despite the fact that $\#(L'_k > 0 \mid H_0)$ is the same for all processors.

	1	2	3	4	5	6	7	8	9	10	11	12	13
1							35.3	35.3			45	45	
2									35.3	35.3	45	45	
3							35.3			35.3		45	45
4								35.3	35.3			45	45
5							35.3		35.3		45		45
6								35.3		35.3	45		45
7	35.3		35.3		35.3								
8	35.3			35.3		35.3							
9		35.3		35.3	35.3								
10		35.3	35.3			35.3							
11	45	45			45	45							
12	45	45	45	45									
13			45	45	45	45							

Table 5.2: Angle Between Neighboring Weight Vectors

While we were doing the simulation for the false alarm rate, we also attempted to use the syndromes to filter some of the data processors' numerical noises. We have assumed the no-fault condition H_0 for all cases, and calculated the data processor's correct output $\hat{\underline{y}}$ as

$$\hat{\underline{y}}_k = \underline{y}_k + \frac{\sigma_r^2}{\sigma_v^2} \sum_{m=N+1}^{N+C} w_{m,k} \underline{s}_m \quad \text{for } k = 1, \dots, N \quad (5.11)$$

The observed ratio between the corrected output noise variance $\text{Var}(\hat{\underline{y}}_k - \bar{\underline{y}}_k)$, where $\underline{y}_k = \underline{0}$, and the original noise variance $\text{Var}(\underline{y}_k)$ is equal to

$$\frac{\text{Var}(\hat{\underline{y}}_k - \bar{\underline{y}}_k)}{\text{Var}(\underline{y}_k)} = 0.73416 \quad (5.12)$$

This is very close to the predicted value $N/(N + C) \approx 10/13 = 0.76923$. The noise reduction rate with this method is marginal and may not justify the extra computation (adding just one bit to processor registers would reduce the noise variance by a factor of 0.25).

5.2.3 Dependence on q

This simulation examines the dependency of the projection method on the output batch size q . Table 5.3 shows a set of simulations with five different values of q . For each q the histogram was compiled with 10,000 simulations. The thresholds γ'_k were set so that the probability that each relative likelihood exceeds zero were all set to $p(L'_k > 0 | H_0) = 0.1$. The processor noise standard deviation was set to $\sigma_T = 10 \text{ lsb}$. Note that γ'_k does not scale linearly with q . The Gaussian approximation of chi-square distribution indicates that for large q , the mean of L'_k approaches $q + \gamma'_k$ and the standard deviation approaches $\sqrt{2q}$. Therefore, the γ'_k scales approximately as $-q - \tau\sqrt{q}$, where τ is fixed by the desired $p(L'_k > 0 | H_0)$.

One can see that $\langle \#(L'_k > 0 | H_0) \rangle$, the average number of times that each relative likelihood is greater than zero, is again close to the predicted value 1,000 for all k . However, $\langle \#(\text{Choose } H_k | H_0) \rangle$, the average number of false alarms on processor k , increases slightly as q increases for all k . This dependency on q was not predicted by our likelihood ratio test. Although we do not have an exact explanation as to why this happens, it is most likely because the shape of the probability distribution of the relative likelihoods changes with increasing q , dropping the tail of the distribution curve until it looks Gaussian. The false alarm is caused by events in the tail of the likelihood distribution curve. However, using different threshold values for different q results in a chi-square distribution whose tail shape changes with q . The integral of $p(\underline{s} | H_0)$ over decision regions H_k will be a complicated function of q . The increasing false alarm rate with increasing q also suggests that the likelihoods get more uncorrelated with increasing q .

However, this dependency on q is weak in nature. Since the probability of processor failure is estimated at best in orders of magnitude, this weak dependency on q does not greatly affect the application of our projection method.

Another variable dependent on q is σ_{crr}^2 , the noise variance of the corrected output of the processor which has been misdiagnosed to be faulty. In case there is a false alarm due to the excessive numerical noise, the output of the processor which has been diagnosed to be faulty is corrected. The σ_{crr}^2 is defined as

$$\sigma_{crr}^2 = \text{Var}(\hat{\underline{y}}_k - \bar{\underline{y}}_k) \quad (5.13)$$

where

$$\hat{\underline{y}}_k = \underline{y}_k - \hat{\underline{\phi}}_k \quad (5.14)$$

and $\underline{y}_k$ is the correct value of $\underline{y}_k$ without any numerical noise. In our simulation, since the correct value of all the processors is set to zero, the variance of the corrected output which has been misdiagnosed to be faulty is equal to $\text{Var}(\hat{\underline{y}}_k)$.

When there is a false alarm, $\hat{\underline{\phi}}_k$ is equal to the numerical noises from all the processors projected onto the weight vector k .

$$\hat{\underline{\phi}}_k = \sum_{m=1}^{N+C} \frac{\underline{w}_k^T \underline{w}_m}{\|\underline{w}_k\| \|\underline{w}_m\|} \underline{\phi}_m \quad (5.15)$$

The output of the processor with the false alarm is corrected by subtracting $\hat{\underline{\phi}}_k$. Therefore, the corrected output $\hat{\underline{y}}_k$ has its own processor noise subtracted out, but now contains the projection of the numerical noises from all other processors with non-orthogonal weight vectors.

$$\hat{\underline{\phi}}_k = - \sum_{\substack{m=1 \\ m \neq k}}^{N+C} \frac{\underline{w}_k^T \underline{w}_m}{\|\underline{w}_k\| \|\underline{w}_m\|} \underline{\phi}_m \quad (5.16)$$

Therefore, if we assume that $\underline{\phi}_m$ is the white noise with variance σ_f^2 , the $\sigma_{err}^2/\sigma_f^2$ in this case is equal to

$$\frac{\sigma_{err}^2}{\sigma_f^2} = \sum_{\substack{m=1 \\ m \neq k}}^{N+C} \left| \frac{\underline{w}_k^T \underline{w}_m}{\|\underline{w}_k\| \|\underline{w}_m\|} \right|^2 = 3.33 \quad \text{for } k = 1, 2, \dots, N \quad (5.17)$$

However, in case of a false alarm, the processor noise variance would most likely be higher than σ_f^2 . This is because a higher than normal noise level is needed to cause a false alarm. Therefore, the numerical noise level in the corrected output would also be higher than the value computed above.

Let us examine what happens to the noise level in the corrected output as q increases. Using the Gaussian approximation, as q increases, the mean value of relative log likelihood increases as q , and the standard deviation

q	1	3	10	30	100
$\langle \#(L'_k > 0 \mid H_0, k \leq 6) \rangle$	1,029	1,010	997	1,005	1,007
$\langle \#(L'_k > 0 \mid H_0, 6 < k \leq N) \rangle$	989	1,020	1007	996	1,025
$\langle \#(L'_k > 0 \mid H_0, k > N) \rangle$	999	996	1,003	1,018	999
$\langle \#(\text{Choose } H_k \mid H_0, k \leq 6) \rangle$	312	334	337	363	372
$\langle \#(\text{Choose } H_k \mid H_0, 6 < k \leq N) \rangle$	268	300	327	324	357
$\langle \#(\text{Choose } H_k \mid H_0, k > N) \rangle$	365	385	403	420	419
$\#(\text{Choose } H_0 \mid H_0)$	5,960	5,641	5,465	5,264	5,088
$\sigma_{err}^2 / \sigma_\Gamma^2$	11.5	7.53	5.22	4.21	3.57

Table 5.3: Simulation with Varying q

increases as $\sqrt{2q}$. Therefore, the normalized threshold $|\gamma'_k|/q$ decreases approximately as $O(1 + \sqrt{2\tau}/\sqrt{q})$. Such a decrease in normalized threshold increases the detection sensitivity to smaller permanent faults. This also causes the average system noise level needed to set off a false alarm to decrease with increasing q . The decreasing syndrome noise level with increasing q in the false alarm case is reflected on σ_{err}^2 as shown in table 5.3. Note that this σ_{err}^2 is for false alarm cases only. In the event of a real fault, the noise level in the system will most likely be at an average level. Therefore, the numerical noise in the corrected output would be significantly less, around $\sigma_{err}^2 / \sigma_\Gamma^2 \approx 3.33$.

Also note that the probability of a fault occurring increases approximately linearly with the processing time spent on the batch. Therefore, for any given calculation, $p(L'_k > 0 \mid H_0)$ should be increased approximately linearly with q for doing the same task (see equation 4.11). This results in a more rapidly decreasing σ_{err}^2 with increasing q until $\sigma_{err}^2 / \sigma_\Gamma^2 \approx 3.33$. The batch size q used in the fault detection/correction does not necessarily have to be equal to the computation batch size. The fault detection/correction batch size q would most likely be determined by the tradeoff between the fault detection sensitivity of a small transient fault and the fault detection sensitivity of a small permanent fault.

$p(L'_k > 0 H_0)$	0.25	0.1	0.05	0.025	0.01
$\#(L'_k > 0 H_0, \text{theory})$	2,500	1,000	500	250	100
$\langle \#(L'_k > 0 H_0, k \leq 6) \rangle$	2,478	997	492	249	101
$\langle \#(L'_k > 0 H_0, 6 < k \leq N) \rangle$	2,480	1,007	512	249	102
$\langle \#(L'_k > 0 H_0, k > N) \rangle$	2,484	1,003	500	248	99
$\langle \#(\text{Choose } H_k H_0, k \leq 6) \rangle$	564	337	212	128	59
$\langle \#(\text{Choose } H_k H_0, 6 < k \leq N) \rangle$	540	327	202	115	55
$\langle \#(\text{Choose } H_k H_0, k > N) \rangle$	680	403	250	142	65
$\#(\text{Choose } H_0 H_0)$	2,420	5,465	7,170	8,350	9,231
$\sigma_{err}^2 / \sigma_r^2$	4.70	5.22	5.61	6.06	6.61

Table 5.4: Simulation with Varying $p(L'_k > 0 | H_0)$

5.2.4 Dependence on Changing False Alarm Rate

This simulation examines the system behavior with changing processor reliability. Table 5.4 shows the set of simulations with five different values of $p(L'_k > 0 | H_0)$. For each value, the simulation histogram was collected from 10,000 simulation trials. The processor noise standard deviation was again set to $\sigma_r = 10 \text{ lsb}$. The $\langle \#(L'_k > 0 | H_0) \rangle$, the average number of times that each relative likelihood exceeds zero, is again close to the theoretically predicted value 1000 for all k . However, $\langle \#(\text{Choose } H_k | H_0) \rangle$, the average number of false alarms in processor k , is much less for all k , especially for high values of $p(L'_k > 0 | H_0)$. This is because the high values of $p(L'_k > 0 | H_0)$ are unrealistically high, with $\sum_{k=1}^{N+C} p(L'_k > 0 | H_0)$ exceeding 1. For more realistic values of $p(L'_k > 0 | H_0)$, the $\langle \#(\text{Choose } H_k | H_0) \rangle$ would still be less than $\langle \#(L'_k > 0 | H_0) \rangle$, but closer to it. However, it is difficult to simulate with realistic failure rates and thresholds since it requires too many trials. (The simulator itself will probably develop a fault before the result is complete.) This simulation shows that for realistic values of $p(L'_k > 0 | H_0)$, $\langle \#(\text{Choose } H_k | H_0) \rangle$ would be within the order of magnitude of $\langle \#(L'_k > 0 | H_0) \rangle$, which is all we need to know, since the processor failure rate itself can best be predicted only within an order of magnitude.

Notice that σ_{err}^2 , the noise variance of the corrected output of the processor in false alarm cases, increases with decreasing $p(L'_k > 0 | H_0)$. This

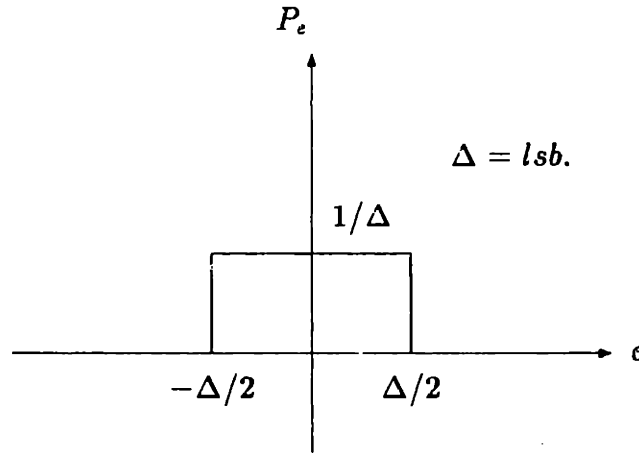


Figure 5.1: Error Distribution for One Roundoff

is because the magnitude of γ'_k increases, and the system noise level needed to set off a false alarm also increases.

5.3 Probability Distribution of the Numerical Noise

When the numerical noises are white Gaussian random variables, we can easily set the thresholds using a chi-square function. However, the numerical noise in real systems is not exactly white Gaussian. The object of this section is to model the fixed point roundoff operations as independent statistical processes and to find out how closely the probability distribution of the fixed point roundoff noise based on this model matches a Gaussian distribution. We also examine how one can compensate for the difference by adjusting γ'_k . Let us assume that each roundoff operation generates an independent roundoff error which has a uniform probabilistic distribution between $-1/2 \text{ } lsb.$ and $+1/2 \text{ } lsb.$, where $lsb.$ stands for the least significant bit. This distribution is shown in figure 5.1. In fixed point arithmetic, addition and subtraction are exact and only multiplication introduces roundoff noise.

The simple example chosen in this section deals with the probabilistic distribution of the numerical noise in the syndrome. Suppose that each processor uses m successive roundings. Suppose further that we use a single fault detection system with one checksum processor whose weights are all equal to 1. Then the total number of roundings, M , contributing quantization noise to the syndrome is equal to

$$M = (N + 1)m \quad (5.18)$$

Such a situation can be imagined when a fixed point FIR filter of length m is implemented in such a way that m input points are first multiplied with m coefficients, rounded, and then summed to produce the output. In real systems, this may not be a desirable way to perform FIR filtering. The preferred method would be not to perform rounding operation after each multiplication, but to sum the non-rounded products and to perform one rounding operation at the end. One would need an accumulator with twice the number of bits to implement the FIR filter this way, which may not always be possible. In this case m would be equal to 1.

Assuming that all rounding steps are independent, the probability distribution of the syndrome numerical noise is equal to the uniform probability distribution of a single rounding step convolved with itself M times. As M approaches infinity, the probability distribution of the numerical noise approaches a Gaussian distribution. However, when M is finite, the probability distribution of the numerical noise deviates from a Gaussian distribution especially at the tail ends of the distribution. For example, when $M = 2$, the noise distribution is triangular, peaking at the origin and becoming zero at $+1 \text{ } lsb.$ and at $-1 \text{ } lsb.$ The variance of the distribution is equal to $1/\sqrt{6} \text{ } lsb.$ Therefore, the tail ends of the distribution are equal to zero beyond about 2.5 times the standard deviation from the origin. This changes our fault detection strategy greatly. Since false alarm is caused by events in the tail, such a drastic change in the shape of the tail causes the threshold to be modified greatly. For example, any syndrome beyond the maximum noise range should be now considered a fault detection ($\gamma'_k = -1 \text{ } lsb.$). As M gets larger, the maximum noise range increases and the distribution curve looks more like Gaussian. However, the tails of the curve are still significantly below Gaussian and one must make adjustments in the γ'_k .

Since it is difficult to calculate the distribution curve for M larger than 2 in a closed form, we have used a numerical simulation in order to find out how closely the distribution curve matches the Gaussian curve for different values of M . The simulation of the convolution is carried out for values of M that are powers of 2. The initial uniform distribution ($M = 1$) between $-1/2 \text{ lsb.}$ and $+1/2 \text{ lsb.}$ is represented by 2048 data points. The simulation is carried out by first convolving the uniform distribution curve with itself. This gives the numerical noise probability distribution for $M = 2$. The result is then decimated by a factor of 2 in order to keep the number of data points from growing exponentially, and the result is convolved with itself again to produce the noise probability distribution for $M = 4$. This decimation and convolution process is repeated for the desired number of repetitions.

The results of this simulation are shown in figure 5.2 and are plotted against a Gaussian distribution curve. The abscissa of the curve is normalized by the standard deviation σ where the variance σ^2 is equal to

$$\sigma^2 = M/12 (\text{lsb}^2) \quad (5.19)$$

When M is low, the tail ends of the noise distribution are much less than Gaussian. As M increases, the noise distribution gets closer to the Gaussian distribution over wider and wider intervals around the origin. Normally, we set γ'_k so that $p(L'_k > 0 \mid H_0)$ has a fixed value. If the tails of the syndrome noise probability distribution are not Gaussian and fall off more rapidly, then we need to use smaller γ'_k than in the Gaussian case. Since the tails of the noise distribution are closer to Gaussian for larger M , the adjustment one has to make on γ'_k also decreases with M .

The exact value of γ'_k depends on the exact shape of the tail. For example, when $q = 1$, the probability that L'_k exceeds zero is equal to the integral of the syndrome noise distribution curve beyond $\pm\sqrt{-\gamma'_k}$. When $q > 1$, the exact value of γ'_k can be calculated from the probability distribution of the relative likelihoods.

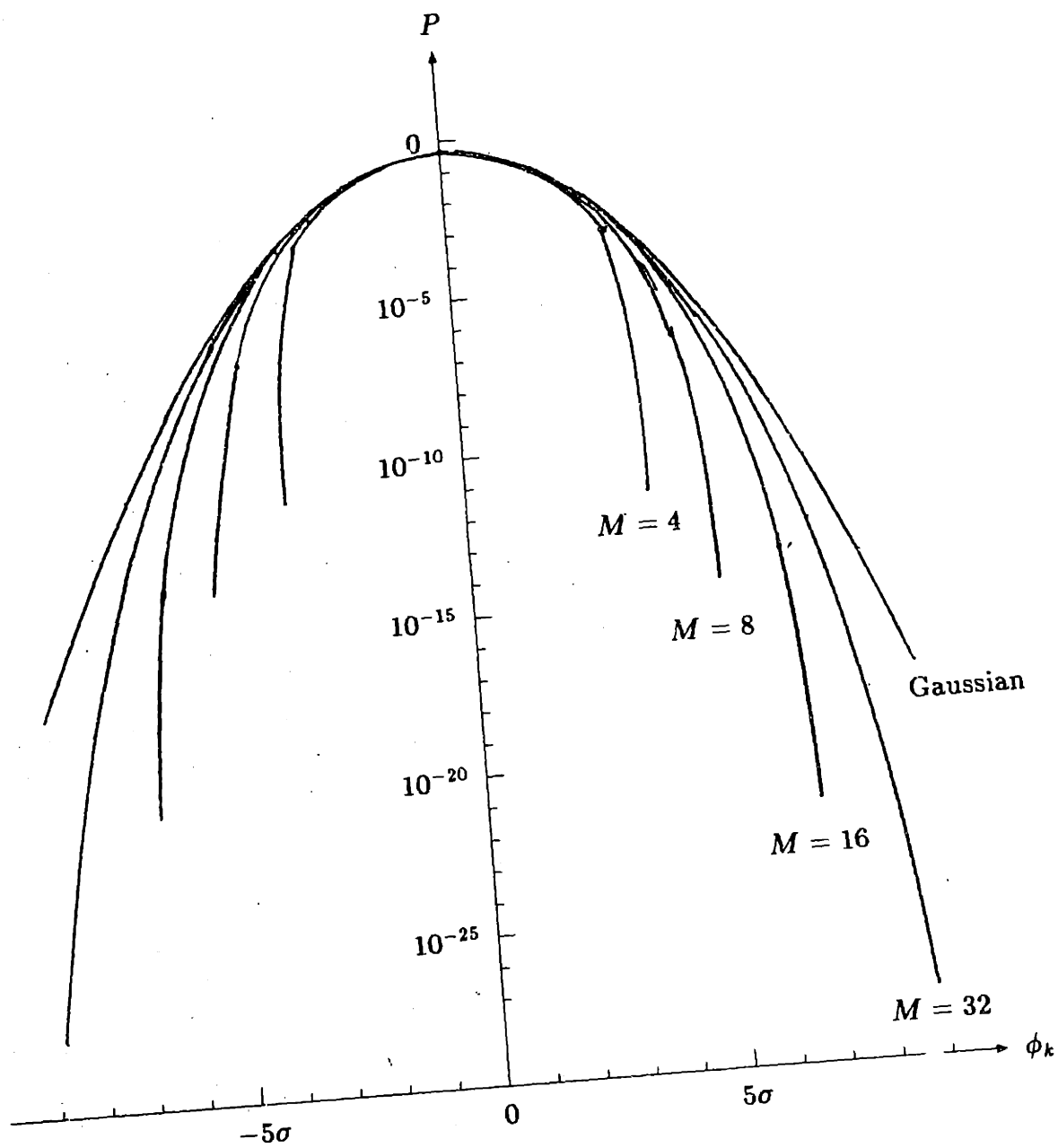


Figure 5.2: Noise Distribution of M Roundoff Operations

5.4 Numerical Noise Histograms of Real Applications

The previous section assumed that the fixed point multiprocessor system's syndrome noise is the result of independent rounding noises that are summed together. The purpose of this simulation is to see whether the fixed point roundoff operations are indeed independent statistical processes. We are interested in seeing whether the syndrome noise probability distributions of real systems match the distribution curve of the previous simulation. Specifically, we simulated a single fault detection system with ten data processors and one checksum processor ($N = 10$ and $C = 1$). All the data processor weights are equal to one ($w_{N+1,k} = 1$) and the checksum processor weight is equal to minus one ($w_{N+1,N+1} = -1$). The simulated tasks are a Finite Impulse Response Filter and a Fast Fourier Transform with random inputs as well as with sinusoidal inputs. If all the roundoff operations were random and non-correlated in nature, the numerical noise in the syndrome should have the probability distribution discussed in the previous section. It should be very close to Gaussian around the origin, and drop below Gaussian in the tail ends.

Figure 5.3 shows the syndrome histogram when all the processors are working correctly and when the processing task is a 10 point FIR filter. In each processor, the input is first multiplied by the coefficients, the results are rounded, and then summed to produce the output. There is one roundoff operation associated with each multiplication. That means the syndrome contains the numerical noise which is the sum of $M = (N + 1) \times q = 110$ roundoff operations. Filter coefficients are chosen randomly, and the results are calculated for both random input and sinusoidal input. The results match the Gaussian curve very closely. Although the previous simulation indicates that the histogram should be much less than Gaussian near the tail, it would take an impractically large number of repetitions of the simulation in order to be able to see the tapering off the tail ends from Gaussian distribution.

Figure 5.4 and figure 5.5 show the syndrome histograms for fixed point FFT operations in a single fault detection system with all processors working correctly. The algorithm chosen for the FFT is the straightforward radix two butterfly algorithm [Oppenheim 75]. There is no scaling by a

factor of two between the stages of the butterfly. (If there were scaling by two in each stage, the theoretical signal to noise ratio based on independent roundoff assumption would have been approximately $q/4$ times higher [Oppenheim 75].) There is a roundoff operation associated with each multiplication. Therefore, each complex multiplication introduces two roundoff operations into each of the real and the imaginary parts of the output. The real and imaginary parts of the syndrome noise are treated as separate numerical noises in the histogram. Both 64 point and 1024 point FFT's are simulated for random input and sinusoidal input.

These histograms differ from Gaussian distribution significantly, peaking around the origin and much below Gaussian near the tails. This result is consistent with Welch's work [Welch 69], which found that the numerical noise variance is much less than predicted by theory based on randomness of the roundoff operations. Therefore, the likelihood ratio constant γ'_k associated with the FFT processing systems would be significantly less than the γ'_k associated with the Gaussian noise case.

These simulations indicate that the numerical noise distribution profile depends heavily on the application and is not necessarily close to the Gaussian profile. Therefore, the γ'_k in the likelihood ratio test have to be adjusted for each application appropriately. The exact value of the γ'_k can be determined from the relative log likelihood probability distribution under the no-fault condition H_0 . The integral of the probability distribution over the range $L'_k > 0$ should be set to the desired false alarm rate. In a case such as the FFT case, the γ'_k would be significantly less than the Gaussian case, due to the much faster falling tails.

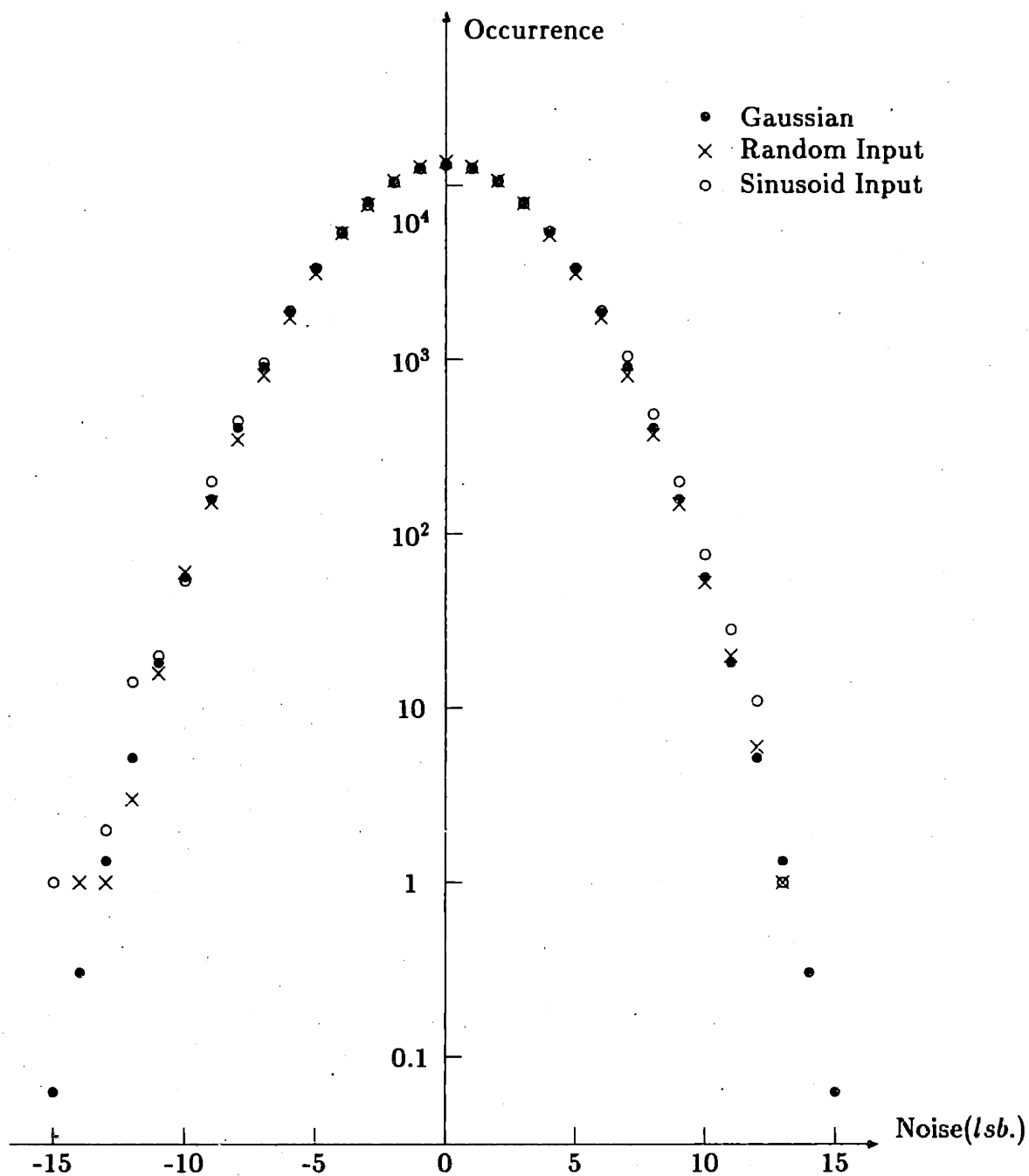


Figure 5.3: Noise Histogram of FIR Single Fault Detection System

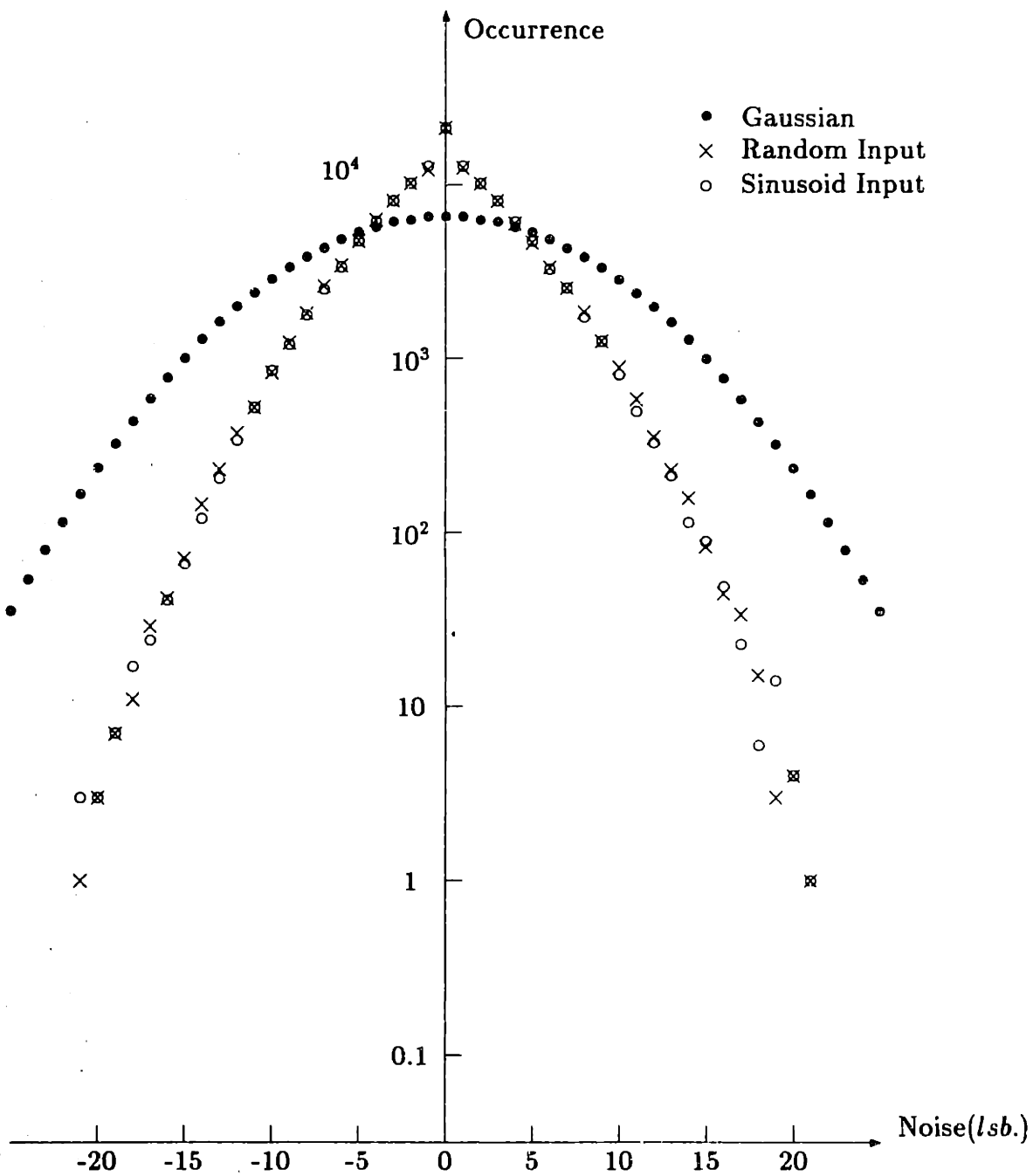


Figure 5.4: Noise Histogram of 64 Point FFT Single Fault Detection System

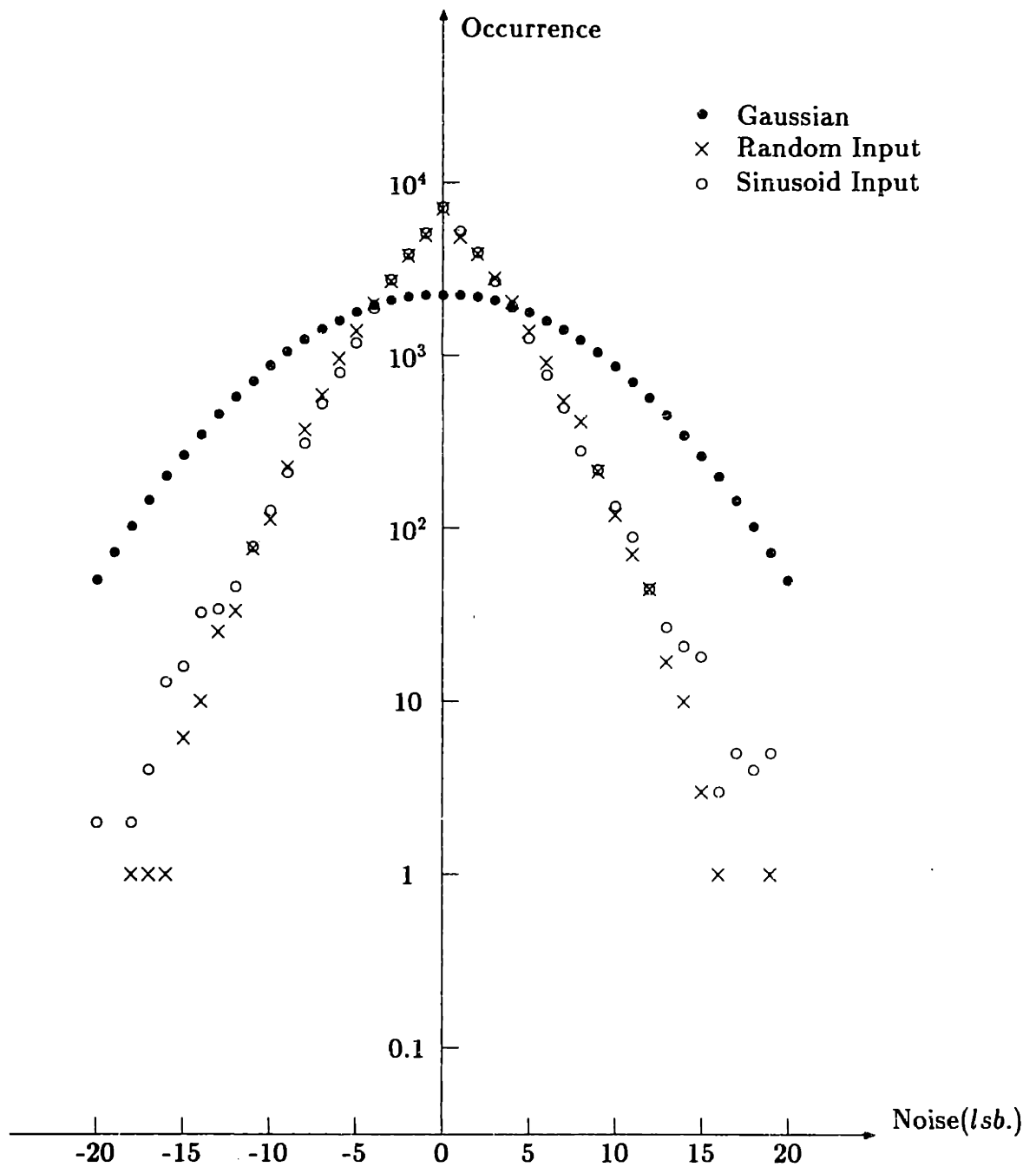


Figure 5.5: Noise Histogram of 1000 Point FFT Single Fault Detection System

Chapter 6

Multiple Fault Correction Systems with Numerical Noise

6.1 Projection Method

6.1.1 K_m Fault Correction

The projection method for K_m fault correction involves the projection of the syndrome $\underline{s}$ on to hyperplanes belonging to the different failure hypotheses. Let $H_{k_1 \dots k_K}$ represent the hypothesis that the processors $k_1, \dots, k_K$ have failed, where $0 < K \leq K_m$. There are $\sum_{k=1}^{K_m} (N+C)! / ((N+C-k)!k!)$ different failure hypotheses.

We define $\sum_{i=1}^K \mathbf{W}_{k_i} \hat{\underline{\phi}}_{k_i}$ as the projection of the syndrome $\underline{s}$ onto the hyperplane of the failure hypothesis $H_{k_1 \dots k_K}$ with respect to the $\mathbf{V}^{-1}$ norm. The estimate of the processor errors $\hat{\underline{\phi}}_{k_1}, \dots, \hat{\underline{\phi}}_{k_K}$ are found by

$$\hat{\underline{\phi}}_{k_1}, \dots, \hat{\underline{\phi}}_{k_K} \leftarrow \min_{\underline{\phi}_{k_1}, \dots, \underline{\phi}_{k_K}} \left\| \underline{s} + \sum_{i=1}^K \mathbf{W}_{k_i} \underline{\phi}_{k_i} \right\|_{\mathbf{V}^{-1}}^2 \quad (6.1)$$

where $\underline{\phi}_{k_1}, \dots, \underline{\phi}_{k_K}$ are possible processor error values. Solving for the minimum,

$$\begin{bmatrix} \hat{\phi}_{k_1} \\ \vdots \\ \hat{\phi}_{k_K} \end{bmatrix} = - \left[\begin{bmatrix} \mathbf{W}_{k_1}^T \\ \vdots \\ \mathbf{W}_{k_K}^T \end{bmatrix} \mathbf{V}^{-1} [\mathbf{W}_{k_1} \dots \mathbf{W}_{k_K}] \right]^{-1} \begin{bmatrix} \mathbf{W}_{k_1}^T \\ \vdots \\ \mathbf{W}_{k_K}^T \end{bmatrix} \mathbf{V}^{-1} \underline{s} \quad (6.2)$$

We can carry out the threshold test for the normalized projection energy in each failure hypothesis $k_1, \dots, k_K$ with $0 \leq K \leq K_m$ by calculating $L'_{k_1 \dots k_K}$ for each projection.

$$\begin{aligned} L'_{k_1 \dots k_K} &= \left\| \sum_{i=1}^K \mathbf{W}_{k_i} \hat{\phi}_{k_i} \right\|_{\mathbf{V}^{-1}}^2 + \gamma'_{k_1 \dots k_K} \\ &= -\underline{s}^T \mathbf{V}^{-1} [\mathbf{W}_{k_1} \dots \mathbf{W}_{k_K}] \begin{bmatrix} \hat{\phi}_{k_1} \\ \vdots \\ \hat{\phi}_{k_K} \end{bmatrix} + \gamma'_{k_1 \dots k_K} \end{aligned} \quad (6.3)$$

where $\gamma'_{k_1 \dots k_K}$ is a constant. If one of the $L'_{k_1 \dots k_K}$ exceeds zero, then the processors $k_1, \dots, k_K$ are chosen to be faulty. If more than one $L'_{k_1 \dots k_K}$ exceed zero, then the largest $L'_{k_1 \dots k_K}$ is chosen to be the failure hypothesis.

Once the processors $k_1, \dots, k_K$ are estimated as being faulty, the correct values of the processor outputs are calculated by

$$\hat{\underline{y}}_{k_i} = \underline{y}_{k_i} - \hat{\phi}_{k_i} \quad \text{for } i = 1, \dots, K \quad (6.4)$$

for the data processors.

6.1.2 White Noise Case

When the numerical noises are white ($\mathbf{V} = \sigma_V^2 \mathbf{I}$), the computational effort can be reduced by using the following calculation method.

$$\rho_{k,l} = \frac{1}{\sigma_V^2} \underline{s}_k^T \underline{s}_l \quad \text{for } \begin{cases} k = N+1, \dots, N+C \\ l = N+1, \dots, N+C \end{cases} \quad (6.5)$$

$$L'_{k_1 \dots k_K} = \text{tr} \left\{ G_{k_1 \dots k_K} \begin{bmatrix} \rho_{N+1, N+1} & \cdots & \rho_{N+1, N+C} \\ \vdots & \ddots & \vdots \\ \rho_{N+C, N+1} & \cdots & \rho_{N+C, N+C} \end{bmatrix} \right\} + \gamma'_{k_1 \dots k_K} \quad (6.6)$$

where $G_{k_1 \dots k_K}$ is a precomputed $C \times C$ matrix of constants defined as

$$G_{k_1 \dots k_K} = \begin{bmatrix} w_{N+1, k_1} & \cdots & w_{N+1, k_K} \\ \vdots & \ddots & \vdots \\ w_{N+C, k_1} & \cdots & w_{N+C, k_K} \end{bmatrix} \begin{bmatrix} r_{k_1 k_1} & \cdots & r_{k_1 k_K} \\ \vdots & \ddots & \vdots \\ r_{k_K k_1} & \cdots & r_{k_K k_K} \end{bmatrix}^{-1} \begin{bmatrix} w_{N+1, k_1} & \cdots & w_{N+C, k_1} \\ \vdots & \ddots & \vdots \\ w_{N+1, k_K} & \cdots & w_{N+C, k_K} \end{bmatrix} \quad (6.7)$$

where r_{kl} is the cross-correlation between the k^{th} and l^{th} weight vectors.

The estimation of the faulty processors' correct output also becomes simpler.

$$\hat{\underline{y}}_{k_i} = \underline{y}_{k_i} - \hat{\underline{\phi}}_{k_i} \quad \text{for } i = 1, \dots, K \quad (6.8)$$

where $\hat{\underline{\phi}}_{k_i}$ can be efficiently computed as

$$\hat{\underline{\phi}}_{k_i} = - \sum_{m=N+1}^{N+C} \tilde{w}_{m, k_i} \underline{s}_m \quad (6.9)$$

where the normalized weights $\tilde{w}_{m, k_i}$ can be precomputed as

$$\begin{bmatrix} \tilde{w}_{N+1, k_1} & \cdots & \tilde{w}_{N+C, k_1} \\ \vdots & \ddots & \vdots \\ \tilde{w}_{N+1, k_K} & \cdots & \tilde{w}_{N+C, k_K} \end{bmatrix} = \begin{bmatrix} r_{k_1 k_1} & \cdots & r_{k_1 k_K} \\ \vdots & \ddots & \vdots \\ r_{k_K k_1} & \cdots & r_{k_K k_K} \end{bmatrix}^{-1} \begin{bmatrix} w_{N+1, k_1} & \cdots & w_{N+C, k_1} \\ \vdots & \ddots & \vdots \\ w_{N+1, k_K} & \cdots & w_{N+C, k_K} \end{bmatrix} \quad (6.10)$$

If we assume that the $2K_m + 1$ modular redundancy technique is used for doing the fault detection/correction from the syndromes, then the computational overhead ratio R_c for the K_m correction is equal to

$$R_c = \frac{C}{N} + \frac{C(p+q)}{T} + (2K_m + 1) \left[\frac{\frac{C(C+1)q}{2}}{NT} + \frac{\frac{C(C+1)}{2} \sum_{k=1}^{K_m} \frac{(N+C)!}{(N+C-k)!k!}}{NT} + \frac{K_m C q}{NT} \right] \quad (6.11)$$

The first term represents the computation in the C checksum processors. The second term represents the input checksum and syndrome calculations. The third term is computation involved in the fault detection/correction algorithm replicated $2K_m + 1$ times. The first term inside the big bracket in the third term represents the computation of ρ_{kl} . The second term in the bracket represents the computation of the relative log likelihoods, and the third term represents the fault correction. As K_m increases, the number of log likelihoods increases as $\sum_{k=1}^{K_m} (N+C)!/((N+C-k)!k!)$ and the term associated with the computation of the log likelihoods grows as $O((2K_m + 1)(N+C)^{K_m-1})$. Therefore, this system may require more overhead computation than a $2K_m + 1$ modular redundancy system, even for a moderate K_m .

6.1.3 Reliability

For the system that can detect or correct up to K_m faults, a system failure occurs if more than K_m processors fail. If we assume that all the processors have the same failure rate P_f and the processor failures are independent of each other, then the system failure rate is equal to

$$\text{Prob}(> K_m \text{ failures}) \approx \frac{(N+C)!}{(N+C-K_m-1)!(K_m+1)!} P_f^{K_m+1} \quad (6.12)$$

In comparison, the system failure rate of a system in which there are $2K_m + 1$ copies of each data processor used in modular redundancy form is equal to

$$\text{Prob}(> K_m \text{ failures in any redundant group}) \approx N \frac{(2K_m+1)!}{K_m!(K_m+1)!} P_f^{K_m+1} \quad (6.13)$$

The failure rate of our system is approximately $\frac{K_m!(N+C)!}{N(2K_m+1)!(N+C-K_m-1)!}$ times higher than the $2K_m + 1$ modular redundancy system, but uses $N + C$ processors rather than $N(2K_m + 1)$ processors, where the minimum C is equal to $2K_m$.

The major difficulty in implementing a K_m fault detection or correction system is that other parts of the system, such as the data distribution network, fault detection/correction hardware, clocks, controls, and power sources, have to be as reliable as the protected processor group. If the failure rates of other parts are comparable to the single processor failure rate, they will have to be protected by a $2K_m + 1$ modular redundancy technique. Therefore, even though the number of the processors does not increase dramatically with K_m , the size of other hardware parts grow as $O(2K_m + 1)$.

The misdiagnosis probability in multiple fault correction is also much higher than in single fault correction because there are many more hypotheses compared to the number of the checksum processors. Therefore, the angles between the projections would be smaller than in the single fault correction case, and it is more likely that the numerical noise may push the syndrome closer to a neighboring hyperplane of another failure hypothesis.

6.1.4 Weight Vectors

It is difficult to find suitable weights for the multiple fault detection/correction systems. Jou and Abraham [Jou 86] used $w_{k,m} = 2^{(k-N-1)(m-1)}$ as weights for the data processors. They have also proven that with these weights, any combination of C weight vectors are linearly independent from each other. These weights also have the advantage that multiplication by a power of 2 can be done in simple bit-shifts. However, the dynamic ranges of the checksum processor registers have to be much greater than those of the data processors in order to be able to accommodate for $w_{N+C,N} = 2^{(C-1)(N-1)}$. The weights vary greatly in magnitude and the numerical noises from the processors with large weights will heavily mask the numerical noises from the processors with small weights in the syndromes. Therefore, the system would be less sensitive to detecting failures in the processors with small weights.

Using low integers as weights is one possible way to get around the dynamic range and the noise masking problems. We have found the following

example set of weight vectors for double fault correction with $C = 4$. These vectors are found by searching through all possible weight vectors with a given range of weights and picking the weight vectors in such a way that any set of $2K_m$ vectors are linearly independent. Using only 0 and ± 1 as weights, there can be only one data processor ($N = 1$) with the weight matrix

$$\mathbf{W} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & -1 \end{bmatrix} \quad (6.14)$$

This is equivalent to the 5-way modular redundancy technique. Using 0, ± 1 , and ± 2 as weights, we found four data processor weight vectors ($N = 4$) with the weight matrix

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 & -1 & 0 & 0 & 0 \\ 1 & -1 & 2 & -2 & 0 & -1 & 0 & 0 \\ 1 & 2 & -2 & -1 & 0 & 0 & -1 & 0 \\ 1 & -2 & -1 & 2 & 0 & 0 & 0 & -1 \end{bmatrix} \quad (6.15)$$

Using 0, ± 1 , ± 2 , and ± 3 as weights, we found six data processor weight vectors ($N = 6$) with the weight matrix

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 2 & -1 & 0 & 0 & 0 \\ 1 & -1 & 2 & -2 & -3 & 3 & 0 & -1 & 0 & 0 \\ 1 & 2 & -1 & -3 & -2 & -3 & 0 & 0 & -1 & 0 \\ 1 & -2 & -3 & -1 & 3 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \quad (6.16)$$

Using 0, ± 1 , ± 2 , ± 3 and ± 4 as weights, we found six data processor weight vectors ($N = 9$) with the weight matrix

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 3 & 4 & 4 & -1 & 0 & 0 & 0 \\ 1 & -1 & 2 & -2 & 3 & -3 & -2 & 3 & -3 & 0 & -1 & 0 & 0 \\ 1 & 2 & -1 & -3 & -4 & -2 & -4 & -3 & 1 & 0 & 0 & -1 & 0 \\ 1 & -2 & -3 & -1 & 2 & 3 & 1 & 1 & 2 & 0 & 0 & 0 & -1 \end{bmatrix} \quad (6.17)$$

The number of data processor weight vectors N was found to be dependent on the order the vectors were picked. This indicates the existence of many different sets of weight vectors with weights $0, \pm 1, \dots, \pm L$, with different numbers of weight vectors in different sets. Although using low integers as weights spreads the weight vectors relatively far apart from each other, the angles between the hyperplanes of different failure hypothesis H_k are not necessarily large. If the angles between the hyperplanes are small, the probability of misdiagnosis increases because the decision regions are narrow.

Another possible set of weights for K_m fault correction is to use the weights in the following form.

$$w_{k,m} = A^{(k-N-1)(m-1)} \quad \text{where } A \neq 1 \quad (6.18)$$

Jou and Abraham have proven that any set of C weight vectors are linearly independent when $A = 2$. However, their proof is valid for some other values of A as well. When the computation is real, any value other than 1 can be used for A . When the computation is complex, A can be of form

$$A = e^{j\omega\pi} \quad (6.19)$$

For example, using $A = e^{j\pi/((C-1)(N-1))}$ is a possibility.

6.1.5 Practicality

The multiple fault correction system may be impractical to implement because of the reasons we have discussed so far, and may not have much advantage over the $2K_m + 1$ modular redundancy system. It is difficult to find suitable weight vectors. The amount of computation needed grows as $O((2K_m + 1)(N + C)^{K_m-1})$, and the overhead computation can easily become more than in the $2K_m + 1$ modular redundancy method. Furthermore, it is difficult to make rest of the system hardware as reliable without using the $2K_m + 1$ modular redundancy method.

It is also difficult to develop good ad hoc methods to reduce the fault detection computation as we did in the single fault correction case. However, when the computation is exact, there are some good ad hoc methods, as we shall discuss later.

6.2 Generalized Likelihood Ratio Test

If we assume that the numerical noise is Gaussian, we can use the generalized likelihood ratio test that we have used for the single fault correction case for multiple fault correction [Musicus 88]. For the K_m fault correction case, the processor faults $\underline{\phi}_{k_1}, \dots, \underline{\phi}_{k_K}$ are again modeled as non-zero mean Gaussian random variables with unknown means $\bar{\phi}_{k_1}, \dots, \bar{\phi}_{k_K}$ and known covariances $\bar{\Phi}_{k_1}, \dots, \bar{\Phi}_{k_K}$. The $\underline{\phi}_k$ numerical noise from the fault-free processors are modeled as zero mean Gaussian random variables with known variance Φ_k . The log likelihood for the failure hypothesis $H_{k_1 \dots k_K}$ is defined as

$$L_{k_1 \dots k_K} = \max_{\bar{\phi}_{k_1}, \dots, \bar{\phi}_{k_K}} \log p(\underline{s}, H_{k_1 \dots k_K} | \bar{\phi}_{k_1}, \dots, \bar{\phi}_{k_K}) \quad (6.20)$$

Let $\hat{\bar{\phi}}_{k_1}, \dots, \hat{\bar{\phi}}_{k_K}$ be the values of $\bar{\phi}_{k_1}, \dots, \bar{\phi}_{k_K}$ that maximize the log likelihood. The $\hat{\bar{\phi}}_{k_1}, \dots, \hat{\bar{\phi}}_{k_K}$ can be thought of as the most likely failure sizes that would have caused the syndrome $\underline{s}$. Using Bayes's Rule and substituting Gaussian densities into the log likelihoods, and then solving for the maximum $L_{k_1 \dots k_K}$ respect to $\bar{\phi}_{k_1}, \dots, \bar{\phi}_{k_K}$, we can show that

$$\hat{\bar{\phi}}_{k_1}, \dots, \hat{\bar{\phi}}_{k_K} \leftarrow \min_{\bar{\phi}_{k_1}, \dots, \bar{\phi}_{k_K}} \|\underline{s} + \sum_{i=1}^K \mathbf{W}_{k_i} \bar{\phi}_{k_i}\|_{\mathbf{V}^{-1}}^2 \quad (6.21)$$

or

$$\begin{bmatrix} \hat{\bar{\phi}}_{k_1} \\ \vdots \\ \hat{\bar{\phi}}_{k_K} \end{bmatrix} = - \left[\begin{bmatrix} \mathbf{W}_{k_1}^T \\ \vdots \\ \mathbf{W}_{k_K}^T \end{bmatrix} \mathbf{V}^{-1} [\mathbf{W}_{k_1} \dots \mathbf{W}_{k_K}] \right]^{-1} \begin{bmatrix} \mathbf{W}_{k_1}^T \\ \vdots \\ \mathbf{W}_{k_K}^T \end{bmatrix} \mathbf{V}^{-1} \underline{s} \quad (6.22)$$

which is the same as the projection case. Substituting this into $L_{k_1 \dots k_K}$,

$$\begin{aligned} L_0 &= -\frac{1}{2} \|\underline{s}\|_{\mathbf{V}^{-1}}^2 + \gamma_0 \\ L_{k_1 \dots k_K} &= -\frac{1}{2} \|\underline{s} + \sum_{i=1}^K \mathbf{W}_{k_i} \hat{\bar{\phi}}_{k_i}\|_{\mathbf{V}^{-1}}^2 + \gamma_{k_1 \dots k_K} \end{aligned} \quad (6.23)$$

If we define the relative log likelihood $L'_{k_1 \dots k_K}$ as

$$L'_{k_1 \dots k_K} = 2(L_{k_1 \dots k_K} - L_0) \quad (6.24)$$

then

$$\begin{aligned} L'_{k_1 \dots k_K} &= \left\| \sum_{i=1}^K \mathbf{W}_{k_i} \hat{\phi}_{k_i} \right\|_{\mathbf{V}^{-1}}^2 + \gamma'_{k_1 \dots k_K} \\ &= -\underline{s}^T \mathbf{V}^{-1} [\mathbf{W}_{k_1} \dots \mathbf{W}_{k_K}] \begin{bmatrix} \hat{\phi}_{k_1} \\ \vdots \\ \hat{\phi}_{k_K} \end{bmatrix} + \gamma'_{k_1 \dots k_K} \end{aligned} \quad (6.25)$$

where $\gamma'_{k_1 \dots k_K}$ is a constant. This is exactly the same result as the projection method.

If the syndromes are also used for reducing the numerical noise as well as for fault detection/correction, then it can be shown that the estimates of the correct processor outputs are

$$\hat{\underline{y}}_m = \begin{cases} \underline{y}_m + \frac{\sigma_y^2}{\sigma_v^2} \left(\sum_{l=N+1}^{N+C} w_{l,m} s_l + \sum_{i=1}^K r_{m,k_i} \hat{\phi}_{k_i} \right) & \text{for } m \neq k_1, \dots, k_K \\ \underline{y}_m - \hat{\phi}_m & \text{for } m = k_1, \dots, k_K \end{cases} \quad (6.26)$$

Chapter 7

The Exact Arithmetic Systems

7.1 The Integer Arithmetic Systems

7.1.1 Single Fault Correction

When integer arithmetic with no rounding is used in fault tolerant multiprocessor architecture, no numerical noise is introduced and all the computations are exact. This makes single fault detection/correction procedure simple. If there is no fault, all the syndromes will be equal to zero ($\underline{s} = \underline{0}$). If there is a fault in processor K , the syndromes will be equal to $\underline{s}_m = -w_{m,k}\underline{\phi}_k$. A good fault detection/correction method is to do the fault location on a point by point basis as if $q = 1$ using the slope s_l/s_m between the syndromes. For example, in a single fault correction system with two checksum processors ($C = 2$), the syndrome slope s_{N+2}/s_{N+1} is equal to

$$\frac{s_{N+2}}{s_{N+1}} = \frac{w_{N+2,k}}{w_{N+1,k}} \quad (7.1)$$

If there are more than two syndromes, one has to compute the slopes between $C - 1$ pairs of the syndromes.

7.1.2 Multiple Fault Correction

Previously in section 2.1.2, we mentioned that a single fault detection/correction system using only 0 and 1 as weights is equivalent to using error coding techniques in the multiprocessor environment. The checksum processors in

this case look very much like “parity” processors. Using this parity processor technique, many existing single error correcting code techniques based on parity checks, such as Hamming Code, can be directly applied to the multiprocessor systems.

Multiple error correction codes, however, cannot be as readily applied to the multiprocessor system as can the single error correction codes. The reason is that the parity bit techniques rely on the fact that the parity bit is formed by the modulo 2 addition of the protected bits. That means that an odd number of faulty bits produces a parity mismatch and an even number of faulty bits produces a the parity match, which is not applicable if the addition is not of modulo 2.

The reason why it would be desirable to be able to apply the multiple error correcting code to the multiprocessor system is that in order to figure out which processors are faulty, the system has to figure out which multidimensional hyperplane of the different failure hypothesis the syndrome belongs. For the K_m fault correction system, there are $\sum_{k=1}^{K_m} (N + C)! / ((N + C - k)!k!)$ hyperplanes, each corresponding to one possible failure hypothesis. Therefore, figuring out which hyperplane the syndrome belongs to can take a lot of computation even for a moderate K_m .

One possible way to apply a multiple error correction code to multiprocessor architecture is to apply it at the bit levels. Consider the following multiple error correction code used by a data transmission system in which the bits $b_1, b_2, \dots, b_N$ are protected by the “parity” bits $b_{N+1}, b_{N+2}, \dots, b_{N+C}$. The parity bits $b_{N+1}, b_{N+2}, \dots, b_{N+C}$ are calculated at the transmission end to be

$$b_k = \left(\sum_{m=1}^N w_{k,m} b_m \right) \text{ mod } 2 \quad \text{for } k = N + 1, \dots, N + C \quad (7.2)$$

where the weights $w_{k,m}$ are 1's and 0's. At the receiving end, the fault location is done with the parity check syndrome s_k .

$$s_k = \left(b_k - \sum_{m=1}^N w_{k,m} b_m \right) \text{ mod } 2 \quad \text{for } k = N + 1, \dots, N + C \quad (7.3)$$

The faulty bit location can be done using the code word $(s_{N+1}, s_{N+2}, \dots, s_{N+C})$ with a lookup table.

We can apply this multiple error correcting code to the multiprocessor system at the bit levels. The input checksums in this case are formed using the same set of weights $w_{k,m}$.

$$x_k = \sum_{m=1}^N w_{k,m} x_m \quad \text{for } k = N+1, \dots, N+C \quad (7.4)$$

At the output, the decoding is done on a point by point basis (as if $q = 1$) at the bit levels. First, the least significant bits of the data processors are corrected. Let $b_{k,n}$ represent the n^{th} least significant bit in the processor output y_k . The least significant bits $b_{N,1}, b_{N-1,1}, \dots, b_{1,1}$ are corrected using the bits $b_{N+C,1}, b_{N+C-1,1}, \dots, b_{N+1,1}$, just like in the error correcting code case. In order to do this correction, C least significant bit syndromes are formed.

$$s_k = b_{k,1} - \sum_{m=1}^N w_{k,m} b_{m,1} \quad \text{for } k = N+1, \dots, N+C \quad (7.5)$$

Let $s_{k,n}$ represent the n^{th} least significant bit of s_k . The faulty bits are located by using the code word $(s_{N+C,1}, s_{N+C-1,1}, \dots, s_{N+1,1})$, just as in the error correcting code case. The correct least significant bit values $\hat{b}_{k,1}$ are then calculated.

After the least significant bits are corrected, the second least significant bits can be corrected. This time the syndromes are calculated from the second least significant bits $b_{k,2}$ and the correct least significant bits $\hat{b}_{k,1}$.

$$s_k = b_{k,2} - \sum_{m=1}^N w_{k,m} (2b_{m,2} + \hat{b}_{m,1}) \quad \text{for } k = N+1, \dots, N+C \quad (7.6)$$

Then the fault corrections for $y_{k,2}$ are done with the code word $(s_{N+C,2}, s_{N+C-1,2}, \dots, s_{N+1,2})$. Since the least significant bits $\hat{b}_{m,1}$ are correct, the odd number of faulty second least significant bits produces the parity mismatch ($s_{m,2} = 1$), and the even number of faulty least significant bits produces the parity match ($s_{m,2} = 0$), just like the error correcting code case.

Then the third least significant bits are corrected and so on. For n^{th} least significant bit correction s_k is equal to

$$s_k = b_{k,n} - \sum_{m=1}^N \left(w_{k,m} \sum_{l=1}^n 2^l \hat{b}_{m,l} \right) \quad \text{for } k = N+1, \dots, N+C \quad (7.7)$$

and the fault correction is done with the code word $(s_{N+C,n}, s_{N+C-1,n}, \dots, s_{N+1,n})$.

Although using only 1's and 0's as weights requires more than the minimum number of checksum processors, using the multiple error correcting codes in bit by bit fashion for integer processors may be much simpler than using weighted checksum, since figuring out which decision region the syndrome belongs to is most likely a computationally intensive task. This technique may be especially useful for bit-serial type machines.

7.1.3 Modulo Arithmetic in Checksum Processors

In integer arithmetic systems, the weights are integers as well. Assuming that there is no overflow or rounding in the processors, we have to use more bits in the checksum processors than in the data processors to prevent the overflow in the checksum processors. In a single fault detection/correction system, the checksum processor k should have $\log_2 \omega_k$ more bits than the data processors in order to prevent the overflow, where the ω_k is defined as

$$\omega_m = \sum_{k=1}^N |w_{m,k}| \quad \text{for } m = N+1, \dots, N+C \quad (7.8)$$

We can reduce the number of extra bits needed in the checksum processors by using modulo M arithmetic in the checksum processors. The modulo M should be carefully chosen so that using the modulo arithmetic in place of the integer arithmetic does not change the value of the syndrome $\underline{s}_m$. Suppose that the data processor outputs $\underline{y}_k$'s are always in the range $|\underline{y}_k| < R$. Then the m^{th} syndrome of the single fault detection/correction system would always be

$$|\underline{s}_m| < \max(|w_{m,k}|)R \quad (7.9)$$

Therefore, if the modulo M arithmetic is used in the checksum processor m where

$$M \geq \max_k (|w_{m,k}|) R \quad (7.10)$$

then the syndrome $\underline{s}_m$ would be identical to the syndrome of an ordinary integer arithmetic system. Note that modulo M arithmetic can be used in the m^{th} input checksum calculation and the m^{th} syndrome calculation as well without changing the value of the syndrome $\underline{s}_m$.

An easy form of modulo M arithmetic to implement in hardware is when M is a power of 2. Therefore, a good choice of M would be a power of two that satisfies $M \geq \max_k (|w_{m,k}|) R$. This modulo arithmetic is especially easy to implement if only the weights -1, 0, and +1 are used in a single fault detection/correction system. Suppose that $R = 2^B$ where B is the number of bits used in the data processor registers. In this case, one can use modulo 2^B arithmetic in the checksum processors, and all the processors would have the same number of bits in their registers, simplifying the hardware design.

For the K_m fault detection or correction systems, one has to use modulo M arithmetic in the m^{th} checksum processor and the corresponding input checksum and syndrome calculations, where M/R is equal to or greater than the sum of the K_m largest $|w_{m,k}|$'s. This choice of M again insures that the syndrome value is not changed. Using a power of 2 for M is convenient to implement in hardware.

7.2 Residue Arithmetic Systems

7.2.1 Residue Number System Multiprocessors

Our fault tolerance technique using weighted checksums can also be applied to multiprocessor systems using residue number system (RNS) processors. A RNS processor converts the input to L residues of different modulus $M_1, M_2, \dots, M_L$ and processes each residue independent of each other. All the residues are combined and converted back to integers at the output. The modulus $M_1, M_2, \dots, M_L$ have to be mutually prime. The RNS processor potentially can run much faster than the conventional integer processors because there is no carry chain delay involved between residues [Taylor 84].

A RNS processor can be viewed as an integer processor which performs modulo M arithmetic, where $M = \prod_{l=1}^L M_l$.

In a conventional RNS processor, fault tolerance is achieved by using extra residues [Etzel 80]. For example, single error detection is achieved by using one extra residue, which is of a modulo that is greater than the other modulos and is mutually prime. The fault detection is done by range detection. If there is a fault in one of the residues, the output y , which is converted from the $L + 1$ residues, is not within the acceptable range of $0 \leq y < M$, where $M = \prod_{l=1}^L M_l$.

Single fault correction in a conventional RNS processor is done with two extra residues, which are of modulos that are greater than the original L modulos. All $(L + 2)$ modulos have to be mutually prime. At the output, $L + 2$ outputs are formed, each from a different set of $(L + 1)$ modulos. When there is no fault, all these outputs are identical. When there is a fault, only one output would be within the acceptable range $0 \leq y < M$, and that is the correct output. It takes L_m extra residues for L_m fault detection, and $2K_m$ extra residues for K_m fault correction. Converting $L + 2$ sets of $L + 1$ modulos to integers and doing the range test is a computationally intensive task. Also, the $(L + 1)^{th}$ and $(L + 2)^{th}$ modulos that are used for fault tolerance have to be bigger than the original L modulos, increasing the size of the hardware involved with those modulos.

For a high throughput system which uses multiple RNS processors parallel, the weighted checksum architecture can be used for fault tolerance instead of using extra residues in each processor. Assuming that there is no overflow (overflow over M where $M = \prod_{l=1}^L M_l$) in the data processors, the weighted checksum technique can be used in the same way as in the ordinary integer processor systems. The only difference is that if the checksum processor dynamic range needs to be larger than the data processor dynamic range, then one has to use extra modulos in the checksum processor in order to increase its dynamic range. If only -1 , 0 , and $+1$ are used for single fault detection/correction, the checksum processors can be modulo M arithmetic processors (i.e. checksum processors are identical to the data processors) as discussed previously in the integer processor systems. If weights other than -1 , 0 , and $+1$ are used for single fault detection/correction, the checksum processor m should use extra residues (use L' residues where $L' > L$) so that $M' \geq \max(|w_{m,k}|)M$ where $M' = \prod_{l=1}^{L'} M_l$. If K_m fault detection/correction is desired, the M'/M should be equal to

or greater than the sum of the K_m largest $|w_{m,k}|$'s.

The major advantage of our fault tolerant architecture is that the fault location can be done with simple slope tests whereas the conventional fault tolerance method with extra residues requires $L + 2$ residue-to-integer conversions which are computationally costly.

7.2.2 Modulo Arithmetic Processor Systems

In the previous section, we have used N RNS processors in parallel to increase the throughput rate. However, there is another way to increase the throughput rate of the RNS Processor. A RNS processor has L residue subprocessors, each processing the residues of different modulus M_i 's. It is possible to increase the throughput of each modulo M_i subprocessor by using N modulo M_i residue subprocessors in parallel in place of one modulo M_i residue subprocessor. In total, there would be L subprocessor groups, each group consisting of N subprocessors and each group based on a different modulo.

In this case, a subprocessor group of N modulo M_i subprocessors can be protected with the weighted checksum technique. Therefore, we shall discuss weighted checksum architectures in which the data processors are modulo M_i arithmetic processors. The major difference from the previous section is that we allow overflows (overflow over M_i) in the data processors. This requires checksum processors to be modulo M_i arithmetic processors. Modulo M_i arithmetic should also be used in input checksum calculations and syndrome calculations.

The weight vectors for a modulo M_i single fault correction system should be chosen so that different processor failures do not generate identical syndromes. That means

$$(\phi_i \underline{w}_i + \phi_j \underline{w}_j) \bmod M_i \neq \underline{0} \quad \text{for} \quad \begin{cases} \phi_i = 1, 2, \dots, M_i - 1 \\ \phi_j = 1, 2, \dots, M_i - 1 \\ i \neq j \end{cases} \quad (7.11)$$

In order to satisfy this condition, all the weight vectors have to be pairwise linearly independent modulo M_i . Notice that this condition for the single fault correction weight vectors is the same condition as for the double fault detection weight vectors.

Since M_l is usually fairly small in RNS systems in order to speed up the computations, there is a limited number of points in the syndrome space $(s_{N+1}, s_{N+2}, \dots, s_{N+C})$. The limited syndrome space limits the number of data processors that can be protected using C checksum processors. Since each syndrome s_k can assume M_l different values $(0, 1, \dots, M_l-1)$, there are only M_l^C points in the syndrome space. Out of M_l^C points in the syndrome space, the no-fault H_0 hypothesis takes up one point ($\underline{s} = \underline{0}$). Each failure hypothesis H_k takes up $M_l - 1$ points.

$$\underline{s} = \underline{w}_{N+1,k} \phi_k \quad \text{for } \phi_k = 1, 2, \dots, M_l - 1 \quad (7.12)$$

Therefore, the maximum number of processors is equal to

$$\max(N + C) = \frac{M_l^C - 1}{M_l - 1} \quad (7.13)$$

However, this is achievable only if M_l is prime. If M_l is not prime but is equal to

$$M_l = \prod_{n=1}^{N_p} P_n \quad (7.14)$$

where P_n 's are primes, the weight vectors have to be linearly independent in all the modulo P_n 's in order to satisfy the unique decision region condition in equation 7.11. Suppose that two weight vectors $\underline{w}_i$ and $\underline{w}_j$ are linearly dependent in modulo P_n .

$$(\underline{w}_i + \alpha \underline{w}_j) \bmod P_n = \underline{0} \quad \text{for } i \neq j \quad (7.15)$$

Then for the processor error values of $\phi_i = \prod_{m \neq n} P_m$ and $\phi_j = \alpha \prod_{m \neq n} P_m$, we have

$$(\phi_i \underline{w}_i + \phi_j \underline{w}_j) \bmod M_l = \underline{0} \quad (7.16)$$

Therefore, processor error of size ϕ_i on processor i yields exactly the same syndromes as a failure of size $\alpha \phi_i$ on processor j . One cannot distinguish between these failures and cannot reliably correct the fault. Therefore, in order to satisfy the unique decision region condition in equation 7.11, the

weight vectors must be linearly independent in all the modulo P_n 's. This means that the maximum number of weight vectors is determined by the smallest P_n .

$$\max(N + C) = \frac{\min(P_n)^C - 1}{\min(P_n) - 1} \quad (7.17)$$

A simple way to construct a set of single fault correction weight vectors is to use a search method. Suppose M_l is prime. Start with a search space of all possible weight vectors $(w_{N+1}, w_{N+2}, \dots, w_{N+C})$ where $w_m = 0, 1, \dots, M_l - 1$. First, eliminate the all-zero vector $\underline{0}$ from the search space since it is not a useful weight vector. Then eliminate the checksum processor weight vectors $\underline{w}_{N+1}^T = (-1, 0, \dots, 0)$, $\underline{w}_{N+2}^T = (0, -1, \dots, 0)$, ..., $\underline{w}_{N+C}^T = (0, \dots, 0, -1)$ and the vectors that are linearly dependent to them ($\underline{w}_k \phi_k$ where $\phi_k = 0, 1, \dots, M_l - 1$ and $k = N + 1, \dots, N + C$) from the search space. Note that "-1" is equal to " $M_l - 1$ " in modulo M_l arithmetic. Then pick one of the remaining vectors as a weight vector $\underline{w}_1$ and delete it and all the linearly dependent vectors ($\underline{w}_1 \phi_1$ where $\phi_1 = 0, 1, \dots, M_l - 1$) from the search space. Then pick one of the remaining vectors as another weight vector $\underline{w}_2$ and delete it and all the linearly dependent vectors from the search space. Then another weight vector is picked from the remaining vectors and so on. This process is repeated until the search space is empty.

The chosen data processor weight vectors can be put into a more convenient form by normalizing them so that the leading non-zero coefficient is equal to 1. Suppose a chosen weight vector $\underline{w}_{m,k}$ has the leading non-zero coefficient α .

$$\underline{w}_k^T = (0, 0, \dots, 0, \alpha, \dots) \quad (7.18)$$

If M_l is prime, then α^{-1} exists. Therefore, the normalized weight vector $\tilde{\underline{w}}_k = (\alpha^{-1} \underline{w}_k) \bmod M_l$ is equal to

$$\tilde{\underline{w}}_k^T = (0, 0, \dots, 0, 1, \dots) \quad (7.19)$$

The normalized weights not only reduce the number of multiplies in the input checksum and syndrome calculations, but also provide a convenient method of detection/correction of a fault if the weight vectors $\underline{w}_k$ are in the normalized form. Suppose the syndromes $\underline{s}^T = (s_{N+1}, s_{N+2}, \dots, s_{N+C})$

H_k	$\underline{w}_k^T$	$\underline{s}_{\phi_k=1}^T$	$\underline{s}_{\phi_k=2}^T$	$\underline{s}_{\phi_k=3}^T$	$\underline{s}_{\phi_k=4}^T$
H_0		(0, 0)			
H_{N+1}	(0, 1)	(0, 1)	(0, 2)	(0, 3)	(0, 4)
H_{N+2}	(1, 0)	(1, 0)	(2, 0)	(3, 0)	(4, 0)
H_1	(1, 1)	(1, 1)	(2, 2)	(3, 3)	(4, 4)
H_2	(1, 2)	(1, 2)	(2, 4)	(3, 1)	(4, 3)
H_3	(1, 3)	(1, 3)	(2, 1)	(3, 4)	(4, 2)
H_4	(1, 4)	(1, 4)	(2, 3)	(3, 2)	(4, 1)

Table 7.1: Weight Vectors and Syndromes for Modulo 5 System ($N = 4$, $C = 2$)

have the leading non-zero coefficient β . Let $\tilde{s} = \beta^{-1}\underline{s} \bmod M_l$. Then if the failure hypothesis H_k is true, we have $\underline{w}_k = \tilde{s}$ with $\hat{\phi}_k = \beta$.

Table 7.1 lists the normalized weight vectors and the possible syndrome values for each decision region for a single fault correction system with two checksum processors using modulo 5 arithmetic. The maximum number of processors is $N + C = 6$. Notice that the data processor weight vectors are in a convenient form of $\underline{w}_k^T = (1, k)$ and the leading non-zero coefficient of the syndrome is equal to the processor error.

Another simple method of locating the faulty processor is to use a lookup table which has a failure hypothesis H_k assigned to all the possible syndromes. In conventional integer arithmetic, the syndrome space would be too large to use such a method. However, in modulo arithmetic, especially when M_l and C are relatively small, the syndrome space becomes a manageable size for such a lookup table method. The M_l is usually small in residue arithmetic in order to speed up the computation, and C does not need to be larger than two or three in a single fault correction scheme, unless N needs to be very large. If normalized weights are used, fault correction is very simple, since the leading non-zero syndrome is equal to the fault size.

The computational overhead associated with our fault detection and correction methods can be significantly lower than in the computational overhead associated with the conventional fault tolerance methods of us-

ing extra residues. Our system also uses identical copies of the residue subprocessors as checksum processors. There is no need to design the sub-processors using different and larger residues.

For the K_m fault correction system, we also have to choose the weight vectors so that different sets of up to K_m processor failures do not generate the same syndromes.

$$\left(\sum_{k=1}^I \phi_{i_k} \underline{w}_{i_k} + \sum_{k=1}^J \phi_{j_k} \underline{w}_{j_k} \right) \bmod M_l \neq \underline{0} \quad \text{for} \quad \begin{cases} 1 \leq i_k \leq N + C \\ 1 \leq j_k \leq N + C \\ I \leq K_m \\ J \leq K_m \end{cases} \quad (7.20)$$

where $(i_1, i_2, \dots, i_I)$ is not the same set of integers as $(j_1, j_2, \dots, j_J)$, and ϕ_{i_k} and ϕ_{j_k} are non-zero. This is achievable if any set of $2K_m$ weight vectors is linearly independent modulo M_l .

$$\left(\sum_{k=1}^{2K_m} \phi_{i_k} \underline{w}_{i_k} \right) \bmod M_l \neq \underline{0} \quad \text{for} \quad \begin{cases} 1 \leq i_k \leq N + C \\ \phi_{i_k} = 1, 2, \dots, M_l - 1 \end{cases} \quad (7.21)$$

Notice that the above condition for the K_m fault correction weight vectors is the same as the condition for the $2K_m$ fault detection weight vectors.

Since each failure decision region takes up $(M_l - 1)^m$ points in the syndromes space where m is the number of failed processors, the maximum number of the processors in K_m fault correction when M_l is prime is determined by the following equation.

$$M_l^C - 1 \geq \sum_{m=1}^{K_m} \frac{(N + C)!}{(N + C - m)!m!} (M_l - 1)^m \quad (7.22)$$

When M_l is not prime, but is equal to $M_l = P_1 P_2 \dots P_{N_p}$, then any set of $2K_m$ weight vectors has to be linearly independent in all the modulo P_n 's in order to satisfy the unique decision region condition in equation 7.20.

Suppose that two sets of K_m weight vectors are linearly dependent.

$$\left(\sum_{k=1}^{K_m} \alpha_k \underline{w}_{i_k} + \sum_{k=1}^{K_m} \beta_k \underline{w}_{j_k} \right) \bmod P_n = \underline{0} \quad (7.23)$$

Then for the processor error values of $\phi_{i_k} = \alpha_k \prod_{m \neq n} P_m$ and $\phi_{j_k} = \beta_k \prod_{m \neq n} P_m$, we have

$$\left(\sum_{k=1}^{K_m} \phi_{i_k} \underline{w}_{i_k} + \sum_{k=1}^{K_m} \phi_{j_k} \underline{w}_{j_k} \right) \bmod M_l = \underline{0} \quad (7.24)$$

Therefore, processor errors of sizes ϕ_{i_k} on processors $(i_1, i_2, \dots, i_{K_m})$ yield exactly the same syndromes as failures of sizes $-\phi_{j_k}$ on processors $(j_1, j_2, \dots, j_{K_m})$. One cannot distinguish between these failure modes and cannot reliably correct the faults. Therefore, in order to satisfy the unique decision region condition in equation 7.20, any set of $2K_m$ weight vectors must be linearly independent in the modulo P_n for $n = 1, \dots, N_p$. This means that the maximum number of weight vectors are determined by the smallest P_n .

$$\min(P_n)^C - 1 \geq \sum_{m=1}^{K_m} \frac{(N+C)!}{(N+C-m)!m!} (\min(P_n) - 1)^m \quad (7.25)$$

A simple way to construct a set of weight vectors is to use a similar weight vector search method as the one used in the single fault correction case. Suppose we need weight vectors for K_m fault correction when M_l is prime. Start with a search space of all possible weight vectors $(w_{N+1}, w_{N+2}, \dots, w_{N+C})$ where $w_m = 0, 1, \dots, M_l - 1$. First, eliminate the all-zero vector $\underline{0}$ from the search space. Then eliminate the checksum processor weight vectors $\underline{w}_{N+1}, \underline{w}_{N+2}, \dots, \underline{w}_{N+C}$ and eliminate all the linear combinations of up to $2K_m - 1$ checksum processor vectors from the search space (i.e. eliminate $\sum_{l=1}^{2K_m-1} \underline{w}_{k_l} \phi_{k_l}$ where $N+1 \leq k_l \leq N+C$, and $\phi_{k_l} = 0, 1, \dots, M_l - 1$). This guarantees that the linear combination of any $2K_m - 1$ checksum processor weight vectors will be linearly independent from the vectors remaining in the search space. Therefore, the next weight vector chosen will not violate the condition that any $2K_m$ weight vectors have to be linearly independent.

Then pick one remaining vector in the space as a weight vector $\underline{w}_1$ and delete it and all the linear combinations of up to $2K_m - 1$ already chosen weight vectors from the search space (i.e. delete $\sum_{l=1}^{2K_m-1} \underline{w}_{k_l} \phi_{k_l}$ from the remaining set where $\phi_{k_l} = 0, 1, \dots, M_l - 1$). Then another weight vector is chosen. This process of choosing weight vectors is repeated until the search

$M_l = 3$ $N + C = 5$	$M_l = 5$ $N + C = 6$	$M_l = 7$ $N + C = 8$	$M_l = 11$ $N + C = 8$	$M_l = 13$ $N + C = 9$	$M_l = 17$ $N + C = 9$
(0, 0, 0, -1)	(0, 0, 0, -1)	(0, 0, 0, -1)	(0, 0, 0, -1)	(0, 0, 0, -1)	(0, 0, 0, -1)
(0, 0, -1, 0)	(0, 0, -1, 0)	(0, 0, -1, 0)	(0, 0, -1, 0)	(0, 0, -1, 0)	(0, 0, -1, 0)
(0, -1, 0, 0)	(0, -1, 0, 0)	(0, -1, 0, 0)	(0, -1, 0, 0)	(0, -1, 0, 0)	(0, -1, 0, 0)
(-1, 0, 0, 0)	(-1, 0, 0, 0)	(-1, 0, 0, 0)	(-1, 0, 0, 0)	(-1, 0, 0, 0)	(-1, 0, 0, 0)
(1, 1, 1, 1)	(1, 1, 1, 1)	(1, 1, 1, 1)	(1, 1, 1, 1)	(1, 1, 1, 1)	(1, 1, 1, 1)
	(1, 2, 3, 4)	(1, 2, 3, 4)	(1, 2, 3, 4)	(1, 2, 3, 4)	(1, 2, 3, 4)
		(1, 5, 6, 2)	(1, 3, 2, 5)	(1, 3, 2, 5)	(1, 3, 2, 5)
		(1, 6, 5, 3)	(1, 4, 5, 9)	(1, 4, 5, 3)	(1, 4, 5, 9)
				(1, 7, 6, 2)	(1, 5, 4, 8)

Table 7.2: Weight Vectors for Double Fault Correction ($C = 4$)

space is empty. One can end up with normalized weight vectors by picking $\underline{w}_k$ to be in a normalized form with the leading non-zero coefficient equal to one.

Note also that for single fault correction, the bound on $N + C$ is tight and can be achieved. For $K_m > 1$, however, the bound is not tight and may not find an appropriate set of vectors spanning the whole space.

Table 7.2 and Table 7.3 show the double fault correction weight vectors for $C = 4$ and $C = 5$ found by the above search method. The number of data processors is much greater for $C = 5$ than for $C = 4$. It is clear from these examples that it is likely that more than the minimum number of checksum processors may be required in order to implement double fault correction.

$M_t = 3$ $N + C = 11$	$M_t = 5$ $N + C = 11$	$M_t = 7$ $N + C = 16$
(0, 0, 0, 0, -1)	(0, 0, 0, 0, -1)	(0, 0, 0, 0, -1)
(0, 0, 0, -1, 0)	(0, 0, 0, -1, 0)	(0, 0, 0, -1, 0)
(0, 0, -1, 0, 0)	(0, 0, -1, 0, 0)	(0, 0, -1, 0, 0)
(0, -1, 0, 0, 0)	(0, -1, 0, 0, 0)	(0, -1, 0, 0, 0)
(-1, 0, 0, 0, 0)	(-1, 0, 0, 0, 0)	(-1, 0, 0, 0, 0)
(0, 1, 1, 1, 1)	(0, 1, 1, 1, 1)	(0, 1, 1, 1, 1)
(1, 0, 1, 1, 2)	(0, 1, 2, 3, 4)	(0, 1, 2, 3, 4)
(1, 1, 0, 2, 1)	(1, 0, 1, 1, 2)	(0, 1, 5, 6, 2)
(1, 1, 2, 0, 2)	(1, 0, 2, 3, 3)	(0, 1, 6, 5, 3)
(1, 2, 1, 2, 0)	(1, 1, 0, 1, 3)	(1, 0, 1, 1, 2)
(1, 2, 2, 1, 1)	(1, 1, 1, 3, 0)	(1, 0, 2, 3, 1)
		(1, 0, 6, 5, 6)
		(1, 1, 0, 5, 1)
		(1, 1, 1, 4, 0)
		(1, 2, 0, 1, 4)
		(1, 2, 3, 2, 6)

Table 7.3: Weight Vectors for Double Fault Correction ($C = 5$)

Chapter 8

Practical Architectures

In this chapter, we shall discuss possible architectures for implementing our single fault correction multiprocessor system. We limit our discussion to the datapath of the system. Of course the clocks, controls, power sources, and other parts of the system have to be designed reliably (by triple modular redundancy or by other methods) so that their failure rates do not dominate the system failure rate.

In our proposed architectures, all the data paths, except the ones that are protected by our fault tolerant algorithm, are triplicated for reliability. If they are not triplicated, it is likely that their failure rate would dominate the system failure rate. Although it is possible to design a reliable data path without triplicating through error coding techniques, it is not clear that one can make such data paths as reliable as the triplicated ones. For example, if the error coded bus system experiences a failure in one of its bus driver chips, the entire bus system may be stuck due to a single failure. In our architectures we have also assumed that the decoder, which is the hardware module that is responsible for fault detection and the faulty output correction from the syndromes, is triplicated for reliability as well. The entire datapath is designed so that any single component failure would not cause the failure of the entire datapath.

8.1 Single Bus Architecture

The single bus architecture is a simple and yet very efficient datapath architecture for implementing our fault tolerant multiprocessor. Figure 8.1 shows the single bus datapath architecture for the single fault correction system. There are N data processors and C checksum processors in the system. It is designed so that any single component failure would not cause a system failure. The main bus is triplicated for reliability. There is also a bus guardian (BG) unit attached to each processor. The bus guardian unit is responsible for driving the bus during the output phase and isolating the bus from the bus driving circuit during the non-output phase. The bus guardian unit is designed so that a single component failure would not disable more than one of three main busses. A local bus connects the bus guardian unit to the data processor. This local bus does not need to be triplicated since it is protected by our fault tolerance algorithm. To the system, a failure in a local bus would be equivalent to and indistinguishable from a failure in the corresponding processor. For a local bus that is connected to the checksum processor, there is also a checksum calculator attached to it. The checksum calculator is responsible for calculating the input checksum and the syndrome, and has two sets of internal accumulators: one for the input checksum and one for the syndrome. These checksum calculators are not triplicated since they are also protected by our fault tolerance algorithm. A failure in a checksum calculator would appear to the system no differently from a failure in the corresponding checksum processor.

Figure 8.3 shows an example design for the bus guardian unit operated by triplicated control signals. During the input phase, the information from the triplicated main bus is voted on before being passed on to the local bus. During the output phase, the signals on the local bus drive all three main busses. The bus guardian unit is designed so that a failure in one of its components would not disable more than one of the three main busses.

Figure 8.2 shows the timing diagram for the architecture. The system processes a batch of N input data segments at a time. Each segment consists of p input data points and q output data points. As the input data comes in through the main bus, it gets loaded onto the data processors through the bus guardian unit and the local bus. The first input data segment $\underline{x}_1$ is sent to the first data processor, the second data segment

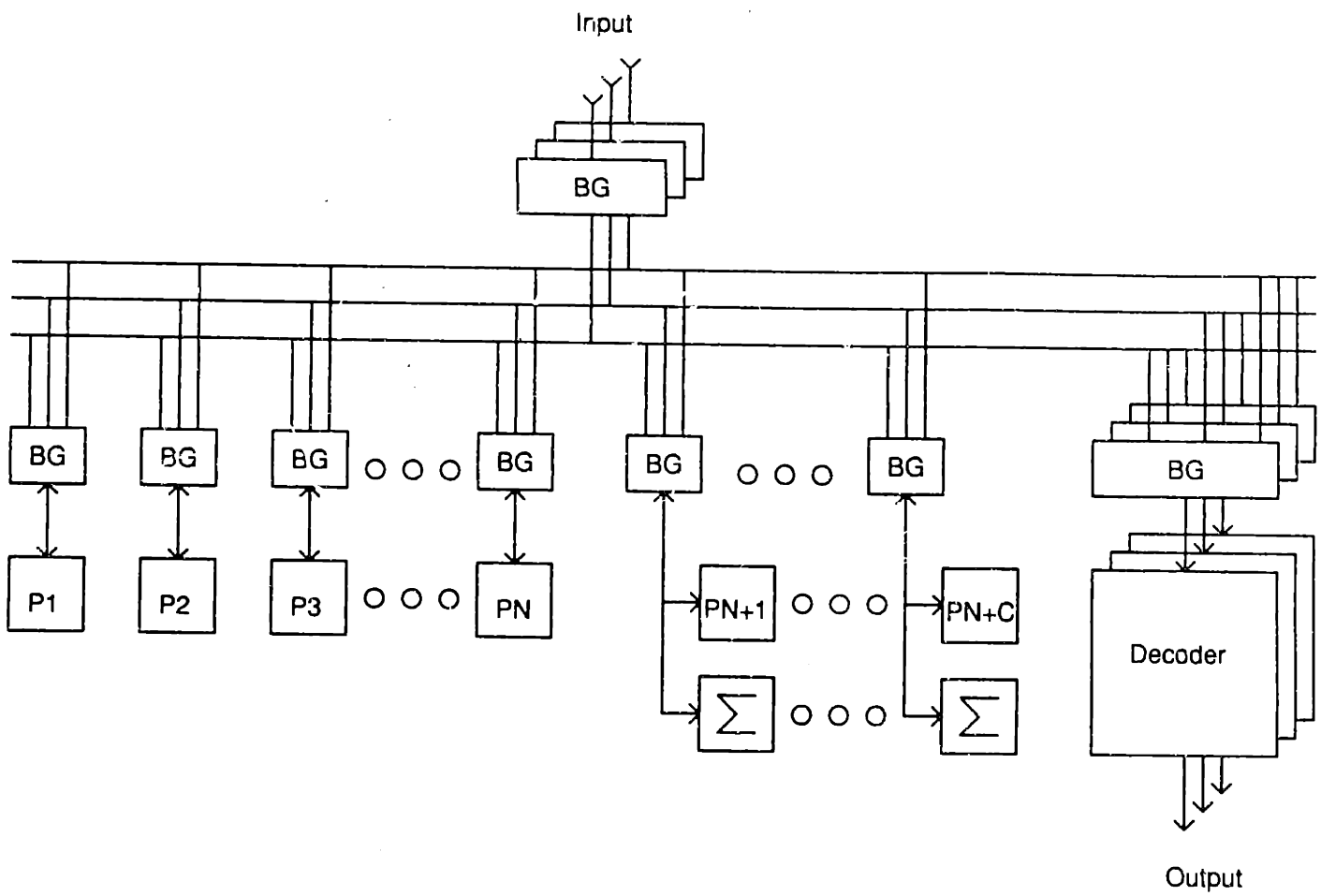


Figure 8.1: Single Bus Architecture

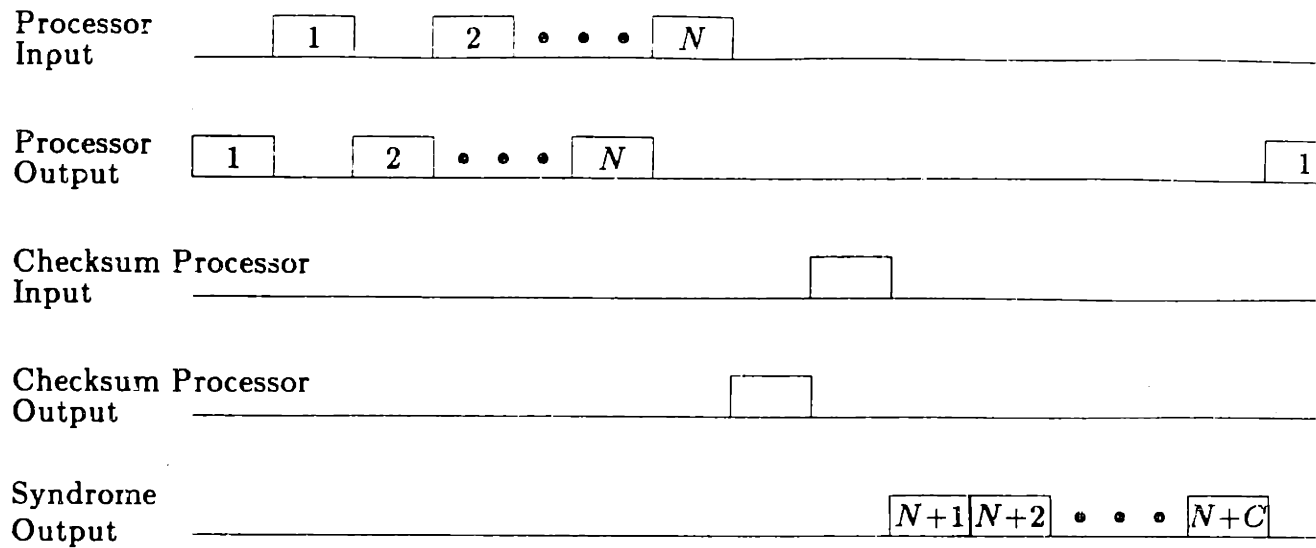


Figure 8.2: Single Bus Architecture Timing Diagram

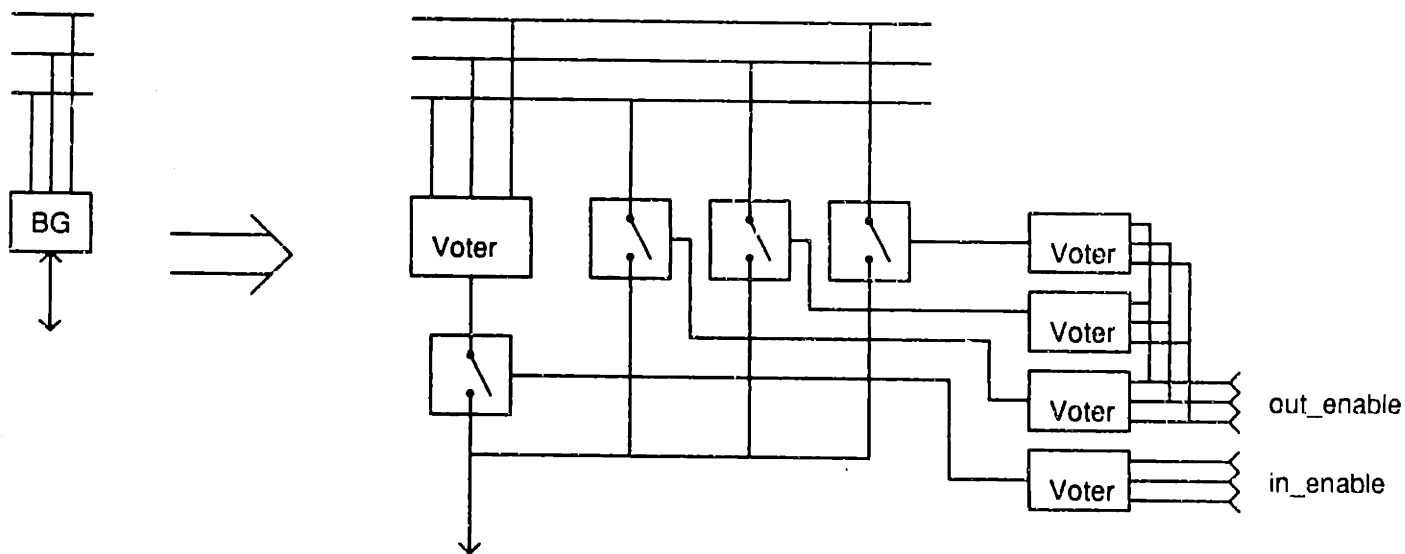


Figure 8.3: Bus Guardian Unit

$\underline{x}_1$ is sent to the second data processor, and so on. Each data processor starts processing the input segment as soon as it is received. Therefore, the data processors' computations will start in a staggered order. The checksum calculators, in the meantime, are in the process of calculating the input checksums. They take the input data $\underline{x}_k$'s off the main bus, multiply it by the appropriate weight $w_{m,k}$'s, and sum the results into the input checksum accumulators. When N^{th} input has been received and the checksum calculators have finished calculating the input checksums, the input checksums are then sent to the corresponding checksum processors through the local bus.

As the data processors finish processing the data, they send the output $\underline{y}_k$'s through the main bus to the decoder. While the outputs are being sent to the decoder, the checksum calculators compute the output checksums the same way they compute the input checksums. They take the output data $\underline{y}_k$'s off the main bus, multiply them by appropriate weight $w_{m,k}$'s, and sum the results in the syndrome accumulators. In a checksum calculator, there are two separate accumulators for the input checksum and the syndrome. When the checksum processors are finished processing the input checksums, they send the result to the checksum calculator through the local bus. The checksum calculators then compute the syndromes $\underline{s}_m$'s by subtracting the output checksums from their accumulators from the outputs of the checksum processors. The resulting syndrome is then sent to the triplicated decoder through the main bus. It takes C data transfer cycles to finish this syndrome transfer process. From the data processor outputs and the syndromes, the decoder detects and corrects the faulty processor output and then outputs the result.

This is a very heavily pipelined architecture. While a batch of input data is going into the data processors, the processed output of the previous batch is being sent to the decoder, and the fault detected/corrected results of the batch before that are being outputted from the decoder. Notice that the timing has been carefully pipelined so that the idle times of the processors are minimized. For example, the input and output phases of the data processor are interleaved so that as soon as a data processor has finished outputting the result through the main bus, it receives the next batch of input data, also through the main bus. The same goes for the checksum processors. All the checksum processors simultaneously output their result to the corresponding checksum calculators through the local

bus. As soon as the outputting is completed, the new set of input checksums are immediately sent from the checksum calculators to the corresponding checksum processors through the local bus.

The major bottleneck of this system is the bandwidth of the main bus. The main bus should have enough bandwidth to take care of all the data processor inputs and outputs as well as the syndrome transfers. If the processor computation time per batch is shorter than the time needed for data transfers, then the processors would sit idle for at least part of the time.

8.2 Unidirectional Data Flow Architecture

There are some situations when it is advantageous to make the system data flow unidirectional. For example, if the data flow is unidirectional, we can use multiplexers instead of busses with bus guardian units in some places. In multiplexers, it is easier to prevent a faulty component from jamming the entire datapath. Figure 8.4 shows an example of unidirectional data flow architecture. In the input stage, a triplicated input bus is used. Notice that simple voters (shown as "V") can be used on the receiving end instead of the complex bus guardian unit, assuming that the voter is designed such that a faulty voter does not load down more than one input bus at a time. The outputs of the processors are sent to the triplicated decoder through a triplicated multiplexer rather than through a bus structure. There are also separate hardware modules for input checksum calculation and the syndrome calculation.

The timing diagram for the unidirectional data flow architecture is in figure 8.5. The input data is loaded onto the data processors the same as in the single bus architecture. The first batch of the data x_1 goes to the first processor, the second batch x_2 to the second processor, and so on. The data processors start processing input data as soon as they receive them. As the input data is being loaded onto the data processors, the input checksum calculators are in the process of calculating the input checksums. They take the input data of the bus, multiply by appropriate weights, and sum the result into their accumulators. As soon as the input checksums are calculated, they are sent to the corresponding checksum processor, which starts processing them immediately. When the data processors are finished

with processing the data, they send the results to the decoder one after another through the multiplexer. While the data processors are outputting their results, the syndrome calculators are computing output checksums. After all the data processors have finished outputting the results, all the checksum processors send their results to the syndrome calculators. The syndrome calculators subtract the output checksums from the checksum processor outputs to compute the syndromes. The resulting syndromes are then sent to the decoder through the triplicated multiplexer. The decoder then locates and corrects the faulty output and outputs the result.

8.3 Variations

In this architecture, it is also possible to vary the number of checksum processors used depending on the requirements of the application. For example, if the application does not require immediate correction of the faulty processor output, one can use the system in a single fault detection configuration instead of the single fault correction configuration. In that case, one can use all the processors except for one of the checksum processors as the data processors. The resulting single fault detection system has $(N + C - 1)$ data processors and one checksum processor, and thus has more processing power.

It is also possible to give a checksum/syndrome calculator to every processor. This way, any processor can be a checksum processor which improves the reconfigurability. As we discussed in section 3.2.3 and 3.3.3, if the checksum processor has the same dynamic range as the data processors in the non-exact arithmetic systems, a small fault in a checksum processor is more easily detected than in the data processors. With a checksum/syndrome calculator built into every processor, one can improve the detectability of the small faults in all the processors by periodically rotating different processors to be the checksum processors.

Having to design the checksum/syndrome calculators and the decoders as different hardware modules may make the system hardware design more complex than desired. It may be possible in some cases to use processors as the checksum/syndrome calculators and the decoders. This may be especially useful if the processors used are single chip processors.

The input bus associated with the architecture in the previous section

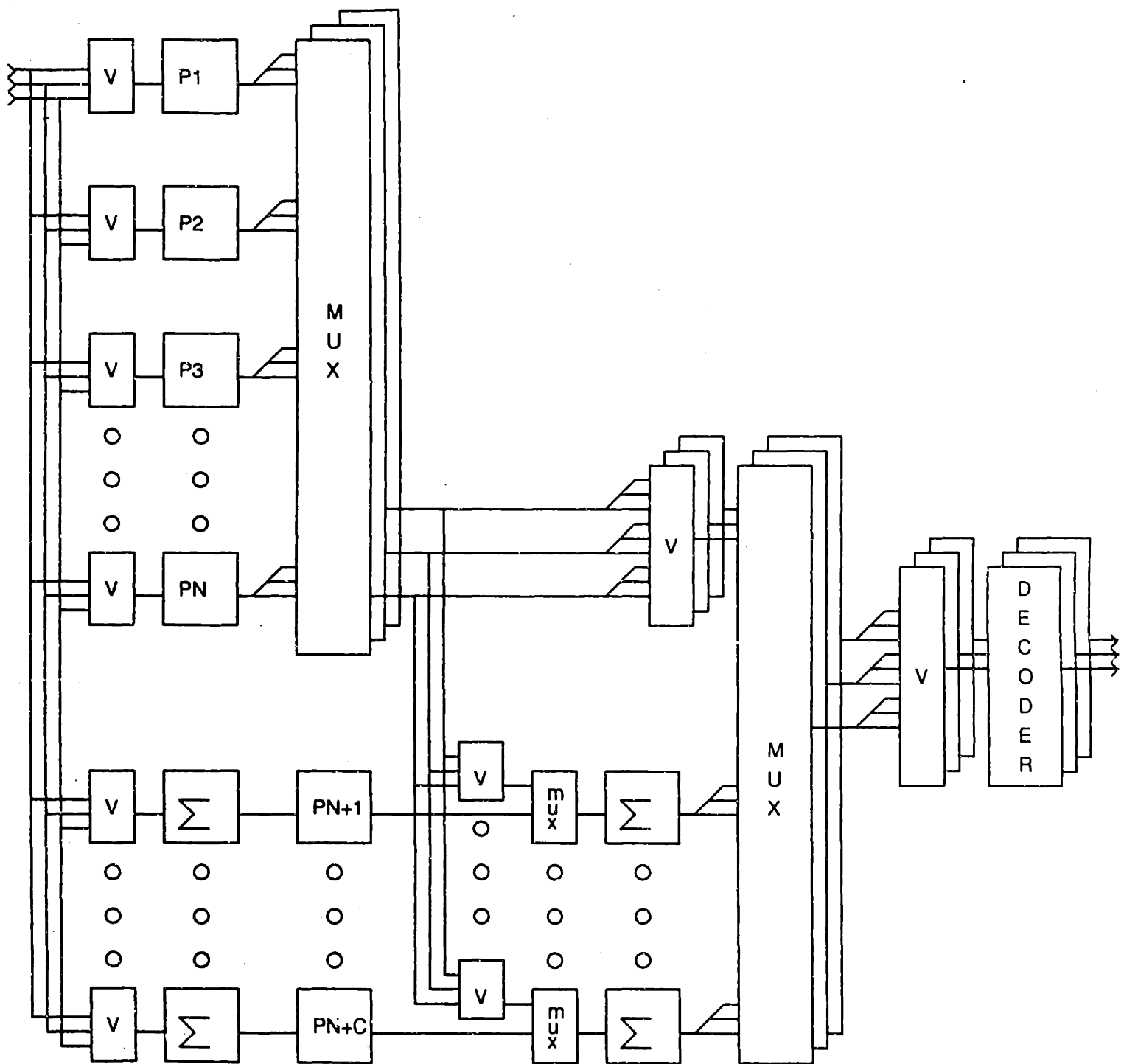


Figure 8.4: Unidirectional Data Flow Architecture

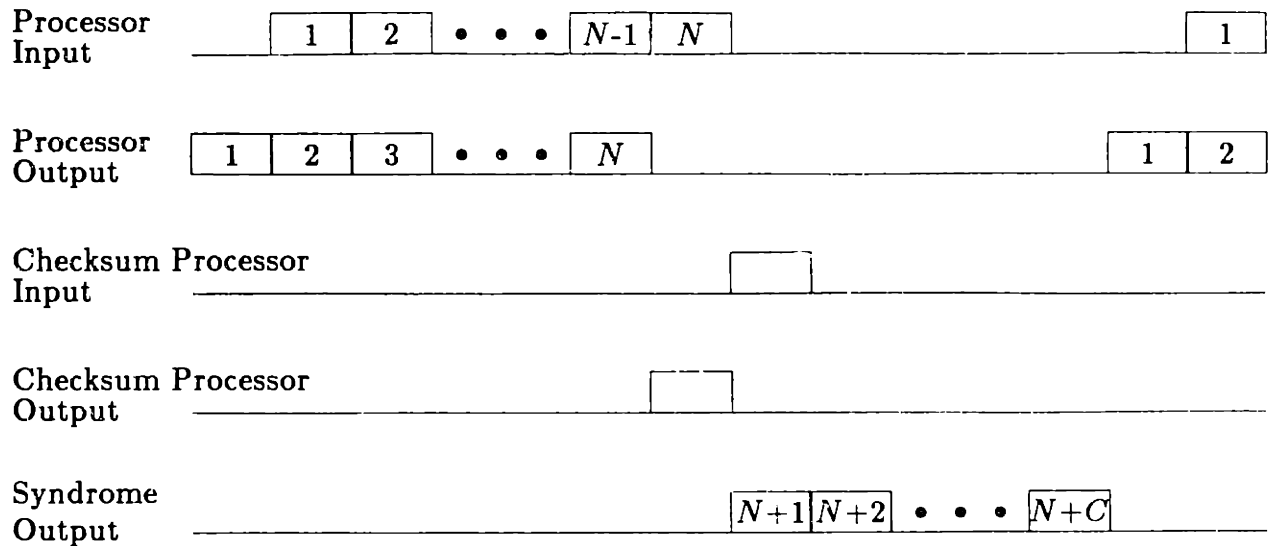


Figure 8.5: Unidirectional Data Flow Architecture Timing Diagram

can be used for a variety of purposes other than just inputting the data. One can use them to load programs, do control, etc. In order to use the input bus for these multiple tasks, one may want to build additional control structures into the system such as interrupts, flags, etc. The input bus is also one of the major bottlenecks associated with the architectures in the previous sections. This bottleneck may be relieved by using multiple input busses. However, the design of such a scheme is complicated by the fact that the input data has to be sent to all the checksum processors as well as the corresponding data processor.

It is also possible to have spare processors that would replace the faulty processors. This would be appropriate if, for example, the mean times till failure of the processors are much shorter than the rest of the system hardware. In this case, the system keeps track of the failure history of the processors, and if any processor is repeatedly diagnosed as being faulty, the system replaces it with a spare processor. In case there are no spare processors left, the system can reconfigure itself from the single fault correction configuration to the single fault detection configuration. However, the

complexity associated with such a reconfigurable system may not justify the increase in performance.

Chapter 9

Conclusion

In this thesis, we have proposed a fault-tolerant multiprocessor architecture which has much less redundant hardware associated with the fault tolerance than the Modular Redundancy techniques. The architecture uses weighted checksum techniques and is suited for linear digital signal processing applications that need to use multiple copies of identical processors in order to meet the throughput requirement.

The system consists of N identical linear data processors and C checksum processors. The input to the checksum processors are weighted sums of the data processor inputs. The fault detection/correction is done using the syndromes which are the differences between the checksum processor output and the appropriately weighted sums of the data processor outputs. When there is no fault, all the syndromes are equal to zero. When there are faults, the syndromes are linear combinations of the weight vectors of the faulty processors. One needs a minimum L_m checksum processors for L_m fault detection, and a minimum of $2K_m$ checksum processors for K_m fault correction. For L_m fault detection the linear combinations of any L_m weight vectors should be linearly independent from each other. For K_m fault correction the linear combinations of any $2K_m$ weight vectors should be linearly independent from each other. The low integer weights are good weights for single fault correction because multiplication by small integers requires less computational effort than a full multiply and also because they spread the weight vectors efficiently in the syndrome space. Good weights for the multiple fault correction are difficult to find.

When fixed point or floating point arithmetic is used, the computation

is no longer exact and the processor outputs and the syndromes contain numerical roundoff or truncation noise. In this case, the numerical noise is modeled as random variables and statistical fault detection/correction methods have been developed. For the single fault detection case, the fault is detected if the syndrome energy level exceeds the preset threshold. For the single fault correction case, the projection energy threshold method is derived for the fault diagnosis. This method projects the syndromes onto the weight vectors and performs a threshold test on the normalized projection energy. If the energy of only one projection exceeds the preset threshold, the corresponding processor is declared faulty. If more than one projection energy exceeds the threshold, the one that exceeds the threshold by the most amount is declared faulty. The projection of the syndrome onto the faulty processor weight vector is used for the fault correction, since it represents the weighted value of the most likely processor error. When the numerical noise is white, the projection method can be simplified. An efficient algorithm for the fault detection/correction computation in this case was also derived.

Since the numerical noise is modeled as a statistical process, the fault detection/correction is no longer exact. These fault diagnosis methods have certain probabilities of misdiagnosis. However, in properly designed systems, such misdiagnosis does not have fatal consequences to the system application. The net effect of the misdiagnosis in such systems is nothing more than the slight increase in the numerical noise level of some of the processor outputs. The probability of misdiagnosis decreases when the weight vectors are spread far apart in the syndrome space.

The projection fault detection/correction method is also equivalent to the result of the generalized likelihood ratio test in which the numerical noises are assumed to be Gaussian random variables. Using the generalized likelihood ratio test, one can also derive a method for using the syndromes not only to detect and correct the faults, but also to filter out some of the numerical noises of the working processors. However, the potential payoff and the practicality of such noise filtering are questionable.

A few other simple ad hoc methods for single fault correction in the presence of the numerical noise have also been derived. Some of them are quite simple and computationally more efficient than the projection threshold method, although the probability of misdiagnosis is higher.

The projection method and the generalized likelihood ratio test have

also been derived for the multiple fault correction in the presence of numerical noise. However, such a multiple fault correction system may not be very practical for the following reasons. First, it is hard to find good weight vectors for multiple fault correction. Second, it is hard to design the rest of the system hardware to be as reliable as the protected processor group without using a modular redundancy technique. Third, the computational effort required for detection/correction of the faults increases dramatically as the number of faults that must be tolerated increases, and the system can become computationally less efficient than the modular redundancy systems of comparable reliabilities.

The simulations of the single fault correction projection method have been carried out using an example fixed point system. Properties that are difficult to compute analytically, such as false alarm rate and the numerical noise in the corrected processor output, are simulated. In addition, the dependence of the fault diagnosis method on variables such as the batch size, the processor reliabilities, and the numerical noise levels were simulated. We also did a simulation to find the probability distribution of the syndrome numerical noise under the assumption that all the roundoff operations are non-correlated. In order to find out whether the roundoff operations in real systems are indeed non-correlated, a fixed point single fault detection system was simulated with the processing task of FIR filter and FFT. The roundoff operations in the FIR filtering were found to have little correlation, but the roundoff operations in FFT were heavily correlated, with the resulting noise level much lower than if they were non-correlated.

When exact integer arithmetic is used, one can achieve reduction in hardware by using modulo arithmetic in the checksum processors and in the calculations of the input checksums and the syndromes. The modulus in the form of 2^b are easy to implement. We have also developed a method of applying the error coding technique to the multiple fault detection/correction systems using integer arithmetic. This eliminates the difficulty of finding good weight vectors for the multiple fault detection/correction systems, and can also reduce the computational effort involved with multiple fault detection/correction. Our fault tolerance method can also be used in the residue number system processors very efficiently.

Practical architectures for the single fault detection/correction system are presented. Single bus architecture and unidirectional data flow architectures are discussed, along with the flexible ways they can be used to

meet the various needs of the application.

The following are suggestions for further research in this topic. The floating point numerical noise probability distribution and the effects of the floating point numerical noise in fault detection/correction algorithms need to be studied in more detail. It is also possible to extend the fault tolerant multiprocessor architecture to non-linear applications. The key idea behind the weighted checksum architecture is that the weighted checksumming process and the linear operation process $\mathbf{F}$ commute. It is likely that there are non-linear processing applications in which the desired processor operation and another operation commute in a similar way to achieve fault tolerance. The search for such operation pairs and the development of appropriate fault detection/correction algorithms are desired.

Our fault-tolerant weighted checksum multiprocessor architecture not only achieves high reliability with low hardware overhead, but is also applicable to any linear digital signal processing applications. We have developed very efficient fault detection/correction algorithms for both exact arithmetic systems and non-exact arithmetic floating point or fixed point systems. Our architectures can mask any single point failure and are flexible to meet the varying degree of fault tolerance required by the application.

Appendix A

Proof of Section 2.2.4

These appendix are taken from Musicus and Song's paper [Musicus 88]. Let $\underline{\phi}$ be the $(N + C)q$ length vector of the $N + C$ processor errors $\underline{\phi}_1, \dots, \underline{\phi}_{N+C}$. If K processors fail, $k_1, \dots, k_K$, then the corresponding error vector $\underline{\phi}$ has K non-zero block elements $\underline{\phi}_{k_1}, \dots, \underline{\phi}_{k_K}$. Let Ω_K be the set of all such possible error vectors corresponding to up to K processor failures. Let $\underline{s} = -\mathbf{W}\underline{\phi}$ be the syndromes corresponding to the errors $\underline{\phi}$. Then we require:

- a) To reliably detect up to $K_m + L_m$ failures, the syndromes must always be non-zero for any non-zero error vector in $\Omega_{K_m+L_m}$:

$$-\mathbf{W}\underline{\phi} \neq 0 \quad \text{for all } \underline{\phi} \in \Omega_{K_m+L_m}, \underline{\phi} \neq 0 \quad (\text{A.1})$$

- b) To reliably correct up to K_m failures, the syndromes corresponding to any two different such failures must be different:

$$-\mathbf{W}\underline{\phi} \neq -\mathbf{W}\underline{\tilde{\phi}} \quad \text{for all } \underline{\phi}, \underline{\tilde{\phi}} \in \Omega_{K_m}, \underline{\phi} \neq \underline{\tilde{\phi}} \quad (\text{A.2})$$

- c) To reliably distinguish a situation involving only K_m or fewer errors from one involving between K_m and $K_m + L_m$ errors, so that we can try to correct the former, we must be able to distinguish the syndromes:

$$-\mathbf{W}\underline{\phi} \neq -\mathbf{W}\underline{\tilde{\phi}} \quad \text{for all } \underline{\phi} \in \Omega_{K_m}, \underline{\tilde{\phi}} \in \Omega_{K_m+L_m}, \underline{\phi} \neq \underline{\tilde{\phi}} \quad (\text{A.3})$$

Note that $\underline{\phi} - \tilde{\underline{\phi}} \in \Omega_{\alpha+\beta}$ if $\underline{\phi} \in \Omega_{\alpha}$ and $\tilde{\underline{\phi}} \in \Omega_{\beta}$. Combining A.1, A.2, and A.3, we will therefore need:

$$-\mathbf{W}\underline{\phi} \neq \underline{0} \quad \text{for all } \underline{p} \in \Omega_{2K_m+L_m}, \underline{\phi} \neq \underline{0} \quad (\text{A.4})$$

This in turn implies that every set of $2K_m + L_m$ block columns of $\mathbf{W}$ must be linearly independent. But this implies that every set of $2K_m + L_m$ weight vectors $(w_{N+1,k} \dots w_{N+C,k})^T$ selected from $k = 1, \dots, N + C$ must be linearly independent. But this can only be possible if $C \geq 2K_m + L_m$.

Q.E.D.

Appendix B

Derivation of GLRT (Proof of Section 4.1)

To derive the formula for L_0 , we apply Bayes Rule and then substitute the Gaussian formula:

$$\begin{aligned}
 L_0 &= \log p(\underline{s}, H_0) \\
 &= \log P(\underline{s}|H_0) + \log P(H_0) \\
 &= -\frac{1}{2}\underline{s}^T \mathbf{V}^{-1} \underline{s} - \frac{1}{2} \log |2\pi \mathbf{V}| + \sum_{m=1}^{N+C} \log(1 - P_m) \quad (\text{B.1})
 \end{aligned}$$

To derive the formula for L_k for $k = 1, \dots, N + C$, we start by applying Bayes Rule to L_k , then substitute the Gaussian formula:

$$\begin{aligned}
 L_k &= \max_{\underline{\phi}_k} \log p(\underline{s}, H_k | \underline{\phi}_k) \\
 &= \max_{\underline{\phi}_k} \log p(\underline{s} | H_k, \underline{\phi}_k) + \log p(H_k) \\
 &= \max_{\underline{\phi}_k} -\frac{1}{2}(\underline{s} + \mathbf{W}_k \underline{\phi}_k)^T (\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T)^{-1} (\underline{s} + \mathbf{W}_k \underline{\phi}_k) \quad (\text{B.2})
 \end{aligned}$$

$$-\frac{1}{2} \log |2\pi(\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T)| + \log P_k + \sum_{\substack{m=1 \\ m \neq k}}^{N+C} \log(1 - P_m)$$

where $\Delta \Phi_k = \Phi_k - \bar{\Phi}_k$. To maximize over $\bar{\phi}_k$, we apply the following lemma:
Lemma A

$$\max_{\bar{\phi}_k} -(\underline{s} + \mathbf{W}_k \bar{\phi}_k)^T (\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T)^{-1} (\underline{s} + \mathbf{W}_k \bar{\phi}_k) \quad (\text{B.3})$$

$$= \max_{\bar{\phi}_k} -(\underline{s} + \mathbf{W}_k \bar{\phi}_k)^T \mathbf{V}^{-1} (\underline{s} + \mathbf{W}_k \bar{\phi}_k) \quad (\text{B.4})$$

and the maximum for both expressions is achieved at the same value of $\hat{\bar{\phi}}_k$:

$$\hat{\bar{\phi}}_k = -(\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} \mathbf{W}_k^T \mathbf{V}^{-1} \underline{s} \quad (\text{B.5})$$

Proof: Maximize the expressions above by differentiating with respect to $\bar{\phi}_k$, then setting the derivatives to zero. We find that (B.4) is maximized by (B.5), while (B.3) is maximized by:

$$\bar{\phi}_k = -(\mathbf{W}_k^T (\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T)^{-1} \mathbf{W}_k)^{-1} \mathbf{W}_k^T (\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T)^{-1} \underline{s} \quad (\text{B.6})$$

To simplify this, we use a modified form of the Woodbury ABCD lemma:

$$(\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T)^{-1} = \mathbf{V}^{-1} + \mathbf{V}^{-1} \mathbf{W}_k \Delta \Phi_k (\mathbf{I} - \mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k \Delta \Phi_k)^{-1} \mathbf{W}_k^T \mathbf{V}^{-1} \quad (\text{B.7})$$

This is most easily proved by multiplying both sides by the inverse of the left side and simplifying. (Note that it is true even if $\Delta \Phi_k$ is not invertible.) Multiplying by $\mathbf{W}_k^T$:

$$\mathbf{W}_k^T (\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T)^{-1} = (\mathbf{I} - \mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k \Delta \Phi_k)^{-1} \mathbf{W}_k^T \mathbf{V}^{-1} \quad (\text{B.8})$$

Substituting into (B.6) shows that (B.6) and (B.5) are identical. Now note that:

$$\begin{aligned}\mathbf{W}_k^T \mathbf{V}^{-1}(\underline{s} + \mathbf{W}_k \hat{\underline{\phi}}_k) &= \mathbf{W}_k^T \mathbf{V}^{-1}(\mathbf{I} - \mathbf{W}_k(\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} \mathbf{W}_k^T \mathbf{V}^{-1}) \underline{s} \\ &= \underline{0}\end{aligned}\quad (\text{B.9})$$

Substituting $\hat{\underline{\phi}}_k$ into expression (B.3) and applying the Woodbury formula again shows that:

$$\begin{aligned}& -(\underline{s} + \mathbf{W}_k \hat{\underline{\phi}}_k)^T (\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T)^{-1} (\underline{s} + \mathbf{W}_k \hat{\underline{\phi}}_k) \\ &= -(\underline{s} + \mathbf{W}_k \hat{\underline{\phi}}_k)^T \mathbf{V}^{-1} (\underline{s} + \mathbf{W}_k \hat{\underline{\phi}}_k) \\ &= -\underline{s}^T \mathbf{V}^{-1} \underline{s} + \underline{s}^T \mathbf{V}^{-1} \mathbf{W}_k (\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} \mathbf{W}_k^T \mathbf{V}^{-1} \underline{s}\end{aligned}\quad (\text{B.10})$$

Applying this lemma to L_k gives the formula in section 7.

To compute the relative likelihood, $L'_k = 2(L_k - L_0)$, we exploit equation (B.10). Performing the subtraction gives:

$$L'_k = \underline{s}^T \mathbf{V}^{-1} \mathbf{W}_k (\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} \mathbf{W}_k^T \mathbf{V}^{-1} \underline{s} + \gamma'_k \quad (\text{B.11})$$

where:

$$\gamma'_k = -\log |2\pi(\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T)| + \log |2\pi \mathbf{V}| + 2 \log \left(\frac{P_k}{1 - P_k} \right) \quad (\text{B.12})$$

To simplify this formula, note that the following matrix can be factored in two different ways:

$$\begin{aligned}\begin{bmatrix} \mathbf{I} & \mathbf{W}_k^T \\ \mathbf{W}_k \Delta \Phi_k & \mathbf{V} \end{bmatrix} &= \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{W}_k \Delta \Phi_k & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{W}_k^T \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \\ \begin{bmatrix} \mathbf{I} & \mathbf{W}_k^T \\ \mathbf{W}_k \Delta \Phi_k & \mathbf{V} \end{bmatrix} &= \begin{bmatrix} \mathbf{I} & \mathbf{W}_k^T \mathbf{V}^{-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{I} - \mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k \Delta \Phi_k & \mathbf{0} \\ \mathbf{0} & \mathbf{V} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{V}^{-1} \mathbf{W}_k \Delta \Phi_k & \mathbf{I} \end{bmatrix}\end{aligned}\quad (\text{B.13})$$

Taking the determinants of the matrices in (B.13) and equating them gives:

$$|\mathbf{I}| |\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T| = |\mathbf{I} - \mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k \Delta \Phi_k| |\mathbf{V}| \quad (\text{B.14})$$

Taking the log of both sides gives:

$$\gamma'_k = -\log |\mathbf{I} - \mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k \Delta \Phi_k| + 2 \log \left(\frac{P_k}{1 - P_k} \right) \quad (\text{B.15})$$

Appendix C

Derivation of Alternate GLRT (Proof of Section 4.2)

Under hypothesis H_0 , the processor outputs $\underline{y}$ and syndromes $\underline{s}$ can be expressed in terms of $\underline{\phi}$ as follows:

$$\begin{pmatrix} \underline{y} \\ \underline{s} \end{pmatrix} = \begin{pmatrix} \underline{\bar{y}} \\ \underline{0} \end{pmatrix} + \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\tilde{\mathbf{W}} & \mathbf{I} \end{bmatrix} \underline{\phi} \quad (\text{C.1})$$

where $\tilde{\mathbf{W}}$ is a $Cq \times Nq$ block matrix with block columns $\mathbf{W}_1, \dots, \mathbf{W}_N$. The joint distribution of $\underline{y}$ and $\underline{s}$ is thus:

$$p\left(\begin{pmatrix} \underline{y} \\ \underline{s} \end{pmatrix} | H_0, \underline{\bar{y}}\right) = N\left(\begin{pmatrix} \underline{\bar{y}} \\ \underline{0} \end{pmatrix}, \mathbf{Q}\right) \quad (\text{C.2})$$

where:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\tilde{\mathbf{W}} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \Phi_Y & \mathbf{0} \\ \mathbf{0} & \Phi_S \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\tilde{\mathbf{W}}^T \\ \mathbf{0} & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \Phi_Y & -\Phi_Y \tilde{\mathbf{W}} \\ -\tilde{\mathbf{W}} \Phi_Y & \mathbf{V} \end{bmatrix} \quad (\text{C.3})$$

where:

$$\mathbf{V} = \tilde{\mathbf{W}} \Phi_Y \tilde{\mathbf{W}}^T + \Phi_S \quad (\text{C.4})$$

and where Φ_Y is a $Nq \times Nq$ block diagonal matrix with diagonal blocks $\Phi_1, \dots, \Phi_N$, and where Φ_S is a $Cq \times Cq$ block diagonal matrix with diagonal

blocks $\Phi_{N+1}, \dots, \Phi_{N+C}$. Using Bayes Rule and substituting this Gaussian density into L_0 :

$$\begin{aligned}
L_0 &= \max_{\underline{\bar{y}}} \log p(\underline{y}, \underline{s}, H_0 | \underline{\bar{y}}) \\
&= \max_{\underline{\bar{y}}} \log p(\underline{y}, \underline{s} | H_0, \underline{\bar{y}}) + \log p(H_0) \\
&= \max_{\underline{\bar{y}}} -\frac{1}{2}(\underline{y}^T - \underline{\bar{y}}^T \underline{s}^T) \mathbf{Q}^{-1} \begin{pmatrix} \underline{y} - \underline{\bar{y}} \\ \underline{s} \end{pmatrix} - \frac{1}{2} \log |2\pi \mathbf{Q}| + \sum_{m=1}^{N+C} \log(1 - P_m)
\end{aligned} \tag{C.5}$$

To maximize this over $\underline{\bar{y}}$, we will use the following lemma:

Lemma B

Let:

$$E(\underline{\alpha}) = (\underline{\alpha}^T - \underline{\bar{\alpha}}^T \underline{\beta}^T) \begin{bmatrix} V_{\alpha\alpha} & V_{\alpha\beta} \\ V_{\beta\alpha} & V_{\beta\beta} \end{bmatrix}^{-1} \begin{pmatrix} \underline{\alpha} - \underline{\bar{\alpha}} \\ \underline{\beta} \end{pmatrix} \tag{C.6}$$

Then:

$$\min_{\underline{\bar{\alpha}}} E(\underline{\bar{\alpha}}) = \underline{\beta}^T V_{\beta\beta}^{-1} \underline{\beta} \tag{C.7}$$

and the minimum is achieved at:

$$\hat{\underline{\bar{\alpha}}} = \underline{\alpha} - V_{\alpha\beta} V_{\beta\beta}^{-1} \underline{\beta} \tag{C.8}$$

Also:

$$\begin{bmatrix} V_{\alpha\alpha} & V_{\alpha\beta} \\ V_{\beta\alpha} & V_{\beta\beta} \end{bmatrix}^{-1} \begin{pmatrix} \underline{\alpha} - \hat{\underline{\bar{\alpha}}} \\ \underline{\beta} \end{pmatrix} = \begin{pmatrix} 0 \\ V_{\beta\beta}^{-1} \underline{\beta} \end{pmatrix} \tag{C.9}$$

Proof: We can factor the matrix as follows:

$$\begin{bmatrix} V_{\alpha\alpha} & V_{\alpha\beta} \\ V_{\beta\alpha} & V_{\beta\beta} \end{bmatrix} = \begin{bmatrix} I & V_{\alpha\beta} V_{\beta\beta}^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} V_{\alpha\alpha} - V_{\alpha\beta} V_{\beta\beta}^{-1} V_{\beta\alpha} & 0 \\ 0 & V_{\beta\beta} \end{bmatrix} \begin{bmatrix} I & 0 \\ V_{\beta\beta}^{-1} V_{\beta\alpha} & 0 \end{bmatrix} \tag{C.10}$$

Thus its inverse has the form:

$$\begin{bmatrix} V_{\alpha\alpha} & V_{\alpha\beta} \\ V_{\beta\alpha} & V_{\beta\beta} \end{bmatrix}^{-1} = \begin{bmatrix} I & 0 \\ -V_{\beta\beta}^{-1}V_{\beta\alpha} & I \end{bmatrix} \begin{bmatrix} (V_{\alpha\alpha} - V_{\alpha\beta}V_{\beta\beta}^{-1}V_{\beta\alpha})^{-1} & 0 \\ 0 & V_{\beta\beta}^{-1} \end{bmatrix} \begin{bmatrix} I & -V_{\alpha\beta}V_{\beta\beta}^{-1} \\ 0 & I \end{bmatrix} \quad (\text{C.11})$$

Substituting into (C.6) gives:

$$(\underline{\alpha} - \underline{\bar{\alpha}} - V_{\alpha\beta}V_{\beta\beta}^{-1}\underline{\beta})^T [V_{\alpha\alpha} - V_{\alpha\beta}V_{\beta\beta}^{-1}V_{\beta\alpha}]^{-1} (\underline{\alpha} - \underline{\bar{\alpha}} - V_{\alpha\beta}V_{\beta\beta}^{-1}\underline{\beta}) + \underline{\beta}^T V_{\beta\beta}^{-1} \underline{\beta} \quad (\text{C.12})$$

Minimizing over $\underline{\bar{\alpha}}$ and plugging back into (C.12) completes the proof.

Q.E.D.

Applying this lemma to (C.5), the maximum over $\underline{y}$ is achieved at:

$$\hat{\underline{y}} = \underline{y} + \Phi_Y \tilde{\mathbf{W}}^T \mathbf{V}^{-1} \underline{s} \quad (\text{C.13})$$

and the likelihood equals:

$$L_0 = -\frac{1}{2} \underline{s}^T \mathbf{V}^{-1} \underline{s} + \gamma_0 \quad (\text{C.14})$$

where:

$$\begin{aligned} \gamma_0 &= -\frac{1}{2} \log |2\pi \mathbf{Q}| + \sum_{m=1}^{N+C} \log(1 - P_m) \\ &= -\frac{1}{2} \log \left| 2\pi \begin{bmatrix} \Phi_Y & \mathbf{0} \\ \mathbf{0} & \Phi_S \end{bmatrix} \right| + \sum_{m=1}^{N+C} \log(1 - P_m) \\ &= -\frac{1}{2} \sum_{m=1}^{N+C} \log |2\pi \Phi_m| + \sum_{m=1}^{N+C} \log(1 - P_m) \end{aligned} \quad (\text{C.15})$$

For $k = 1, \dots, N + C$, hypothesis H_k models the processor noise $\bar{\phi}_k$ as a non-zero mean Gaussian random variable with density:

$$P(\underline{\phi}_k | H_k) = N(\bar{\underline{\phi}}_k, \bar{\Phi}_k) \quad (\text{C.16})$$

where $\bar{\underline{\phi}}_k$ is unknown. The other processor errors are the same as under H_0 . The joint distribution of $\underline{y}$ and $\underline{s}$ given H_k and $\bar{\underline{\phi}}_k$ is then:

$$P\left(\left(\begin{array}{c} \underline{y} \\ \underline{s} \end{array}\right) | H_k, \underline{\bar{y}}, \underline{\bar{\phi}}_k\right) = N\left(\left(\begin{array}{c} \underline{\bar{y}} \\ \underline{0} \end{array}\right) + \left(\begin{array}{c} \mathbf{I}_k \\ -\mathbf{W}_k \end{array}\right) \underline{\bar{\phi}}_k, \mathbf{Q}^{(k)}\right) \quad (\text{C.17})$$

where:

$$\mathbf{Q}^{(k)} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\tilde{\mathbf{W}} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \bar{\Phi}_Y & \mathbf{0} \\ \mathbf{0} & \bar{\Phi}_S \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\tilde{\mathbf{W}}^T \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \quad (\text{C.18})$$

For $k = 1, \dots, N$, $\mathbf{I}_k$ is a $Nq \times q$ matrix which is the k^{th} block column of an $Nq \times Nq$ identity matrix, $\bar{\Phi}_Y$ is just Φ_Y with the k^{th} diagonal block replaced by $\bar{\Phi}_k$, and $\bar{\Phi}_S$ equals Φ_S . For $k = N + 1, \dots, N + C$, $\mathbf{I}_k = \mathbf{0}$, $\bar{\Phi}_Y$ equals Φ_Y , and $\bar{\Phi}_S$ is just Φ_S with the k^{th} diagonal block replaced by $\bar{\Phi}_k$. Thus applying Bayes' Rule and substituting the formula for a Gaussian:

$$\begin{aligned} L_k &= \max_{\underline{\bar{y}}} \max_{\underline{\bar{\phi}}_k} \log p(\underline{y}, \underline{s}, H_k | \underline{\bar{y}}, \underline{\bar{\phi}}_k) \\ &= \max_{\underline{\bar{y}}} \max_{\underline{\bar{\phi}}_k} \log p(\underline{y}, \underline{s} | H_k, \underline{\bar{y}}, \underline{\bar{\phi}}_k) \\ &= \max_{\underline{\bar{y}}} \max_{\underline{\bar{\phi}}_k} -\frac{1}{2} (\underline{y}^T - \underline{\bar{y}}^T - \underline{\bar{\phi}}_k^T \mathbf{I}_k^T \underline{s}^T + \underline{\bar{\phi}}_k^{-T} \mathbf{W}_k^T) (\mathbf{Q}^{(k)})^{-1} \begin{pmatrix} \underline{y} - \underline{\bar{y}} - \mathbf{I}_k \underline{\bar{\phi}}_k \\ \underline{s} + \mathbf{W}_k \underline{\bar{\phi}}_k \end{pmatrix} \\ &\quad - \frac{1}{2} \log |2\pi \mathbf{Q}^{(k)}| + \log p(H_k) \end{aligned} \quad (\text{C.19})$$

Applying lemma B to maximize over $\underline{\bar{y}}$ gives:

$$\underline{\bar{y}} = \underline{y} - \mathbf{I}_k \underline{\bar{\phi}}_k + \bar{\Phi}_Y \tilde{\mathbf{W}}^T (\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T)^{-1} (\underline{s} + \mathbf{W}_k \underline{\bar{\phi}}_k) \quad (\text{C.20})$$

Substituting back into (C.19) gives:

$$\begin{aligned} L_k &= \max_{\underline{\bar{\phi}}_k} -\frac{1}{2} (\underline{s} + \mathbf{W}_k \underline{\bar{\phi}}_k)^T (\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T)^{-1} (\underline{s} + \mathbf{W}_k \underline{\bar{\phi}}_k) \\ &\quad - \frac{1}{2} \log |2\pi \mathbf{Q}^{(k)}| + \log P_k - \sum_{\substack{m=1 \\ m \neq k}}^{N+C} \log(1 - P_m) \end{aligned} \quad (\text{C.21})$$

But applying lemma A, the maximum occurs at exactly the same value of $\hat{\underline{\phi}}_k$ as for our previous algorithm, (B.5). The likelihood at the maximum equals:

$$\begin{aligned}
L_k = & -\frac{1}{2}(\underline{s} + \mathbf{W}_k \hat{\underline{\phi}}_k)^T \mathbf{V}^{-1}(\underline{s} + \mathbf{W}_k \hat{\underline{\phi}}_k) - \frac{1}{2} \sum_{\substack{m=1 \\ m \neq k}}^{N+C} \log 2\pi \Phi_m - \frac{1}{2} \log 2\pi \bar{\Phi}_k \\
& + \log P_k + \sum_{\substack{m=1 \\ m \neq k}}^{N+C} \log(1 - P_m)
\end{aligned} \tag{C.22}$$

Substituting (B.5) into the formula for $\bar{\mathbf{y}}$ and simplifying using lemma A again:

$$\hat{\underline{y}}_m = \underline{y} - \mathbf{I}_k \bar{\underline{y}}_k + \bar{\Phi}_Y \tilde{\mathbf{W}}^T \mathbf{V}^{-1}(\underline{s} + \mathbf{W}_k \hat{\underline{\phi}}_k) \tag{C.23}$$

or:

$$\hat{\underline{y}}_m = \begin{cases} \underline{y}_m + \Phi_m \mathbf{W}_m^T \mathbf{V}^{-1}(\underline{s} + \mathbf{W}_k \hat{\underline{\phi}}_k) & \text{for } m \neq k \\ \underline{y}_k - \hat{\underline{\phi}}_k + \Phi_k \mathbf{W}_k^T \mathbf{V}^{-1}(\underline{s} + \mathbf{W}_k \hat{\underline{\phi}}_k) & \text{for } m = k \end{cases} \tag{C.24}$$

Using (B.9), the case $m = k$ simplifies:

$$\hat{\underline{y}}_m = \begin{cases} \underline{y}_m + \Phi_m \mathbf{W}_m^T \mathbf{V}^{-1}(\underline{s} + \mathbf{W}_k \hat{\underline{\phi}}_k) & \text{for } m \neq k \\ \underline{y}_k - \hat{\underline{\phi}}_k & \text{for } m = k \end{cases} \tag{C.25}$$

Now we compute the relative likelihoods as before. The quadratic terms in $\underline{s}$ are exactly the same as in the previous method, but the constants are different:

$$\begin{aligned}
L'_k &= 2(L_k - L_0) \\
&= \underline{s}^T \mathbf{V}^{-1} \mathbf{W}_k (\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} \mathbf{W}_k^T \mathbf{V}^{-1} \underline{s} + \gamma'_k
\end{aligned} \tag{C.26}$$

where:

$$\begin{aligned}
\gamma'_k &= -\log |2\pi\bar{\Phi}_k| + 2\log P_k + \log |2\pi\bar{\phi}_k| - 2\log(1 - P_k) \\
&= -\log |\bar{\Phi}_k\Phi_k^{-1}| + 2\log\left(\frac{P_k}{1 - P_k}\right)
\end{aligned} \tag{C.27}$$

Appendix D

Mean and Variance of Likelihoods

To compute the mean and variance of the relative likelihoods L'_k , we need the following lemma:

Lemma C: Suppose $\underline{\alpha}$, $\underline{\beta}$ are Gaussian random variables with means $\underline{\bar{\alpha}}$, $\underline{\bar{\beta}}$ and cross-covariance $V_{\alpha\beta}$. Then for any matrix A:

$$E[\underline{\alpha}^T A \underline{\beta}] = \text{tr} \{ A E[\underline{\beta} \underline{\alpha}^T] \} = \text{tr} \{ A [\underline{\bar{\beta}} \underline{\bar{\alpha}}^T + V_{\beta\alpha}] \} \quad (\text{D.1})$$

Proof:

$$\begin{aligned} E[\underline{\alpha}^T A \underline{\beta}] &= E[\sum_{i,j} \alpha_i A_{ij} \beta_j] \\ &= \sum_{i,j} A_{ij} E[\alpha_i \beta_j] \\ &= \sum_{i,j} A_{ij} (\bar{\alpha}_i \bar{\beta}_j + [V_{\alpha\beta}]_{i,j}) \\ &= \text{tr} \{ A [\underline{\bar{\beta}} \underline{\bar{\alpha}}^T + V_{\alpha\beta}^T] \} \\ &= \text{tr} \{ A [\underline{\bar{\beta}} \underline{\bar{\alpha}}^T + V_{\beta\alpha}] \} \end{aligned} \quad (\text{D.2})$$

Q.E.D.

With this lemma, we can compute the expected value of L'_m under each hypothesis H_k . Start with the following formula for L'_m :

$$L'_m = \underline{s}^T \mathbf{V}^{-1} \mathbf{W}_m (\mathbf{W}_m^T \mathbf{V}^{-1} \mathbf{W}_m)^{-1} \mathbf{W}_m^T \mathbf{V}^{-1} \underline{s} + \gamma'_m \quad (\text{D.3})$$

Let $\bar{\phi}_k$ be the mean of the failure of the k^{th} processor or syndrome. Using lemma C, plus the fact that $\text{tr}\{AB\} = \text{tr}\{BA\}$:

$$\begin{aligned} E[L'_m|H_0] &= \text{tr} \left\{ \mathbf{V}^{-1} \mathbf{W}_m (\mathbf{W}_m^T \mathbf{V}^{-1} \mathbf{W}_m)^{-1} \mathbf{W}_m^T \mathbf{V}^{-1} (\mathbf{V}) \right\} + \gamma'_m \\ &= \text{tr} \left\{ (\mathbf{W}_m^T \mathbf{V}^{-1} \mathbf{W}_m) (\mathbf{W}_m^T \mathbf{V}^{-1} \mathbf{W}_m)^{-1} \right\} + \gamma'_m \\ &= \text{tr}\{\mathbf{I}\} + \gamma'_m \\ &= q + \gamma'_m \end{aligned} \quad (\text{D.4})$$

For $k = 1, \dots, N$ we use the fact that:

$$\begin{aligned} E[\underline{s}|H_k] &= -\mathbf{W}_k \bar{\phi}_k \\ \text{Var}[\underline{s}|H_k] &= \mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T \end{aligned} \quad (\text{D.5})$$

Then using lemma C again:

$$\begin{aligned} E[L'_m|H_k] &= \text{tr} \left\{ \mathbf{V}^{-1} \mathbf{W}_m (\mathbf{W}_m^T \mathbf{V}^{-1} \mathbf{W}_m)^{-1} \mathbf{W}_m^T \mathbf{V}^{-1} [\mathbf{W}_k \bar{\phi}_k \bar{\phi}_k^T \mathbf{W}_k^T + \mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T] \right\} + \gamma'_m \\ &= \text{tr} \left\{ \mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_m (\mathbf{W}_m^T \mathbf{V}^{-1} \mathbf{W}_m)^{-1} \mathbf{W}_m^T \mathbf{V}^{-1} \mathbf{W}_k [\bar{\phi}_k \bar{\phi}_k^T - \Delta \Phi_k] + \mathbf{I} \right\} + \gamma'_m \\ &= \text{tr} \left\{ \mathbf{R}_{km} \mathbf{R}_{mm}^{-1} \mathbf{R}_{mk} [\bar{\phi}_k \bar{\phi}_k^T - \Delta \Phi_k] \right\} + q + \gamma'_m \end{aligned} \quad (\text{D.6})$$

where:

$$\mathbf{R}_{km} = \mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_m \quad (\text{D.7})$$

To compute the variance, we need the following lemmas:

Lemma D: Suppose $\alpha, \beta, \gamma, \lambda$ are zero-mean, jointly Gaussian random variables. Then:

$$\begin{aligned}
E[\alpha\beta\gamma\lambda] &= E[\alpha\beta]E[\gamma\lambda] + E[\alpha\gamma]E[\beta\lambda] + E[\alpha\lambda]E[\beta\gamma] \\
E[\alpha\beta\gamma] &= 0
\end{aligned} \tag{D.8}$$

Proof: A standard result (see [Helstrom 84] pp. 210)

Lemma E: Suppose $\alpha, \beta, \gamma, \lambda$ are jointly Gaussian random variables with respective means $\bar{\alpha}, \bar{\beta}, \bar{\gamma}, \bar{\lambda}$. Then:

$$E[\alpha\beta\gamma\lambda] = E[\alpha\beta]E[\gamma\lambda] + E[\alpha\gamma]E[\beta\lambda] + E[\alpha\lambda]E[\beta\gamma] - 2\bar{\alpha}\bar{\beta}\bar{\gamma}\bar{\lambda} \tag{D.9}$$

Proof: Let $\tilde{\alpha} = \alpha - \bar{\alpha}$; define $\tilde{\beta}, \tilde{\gamma}, \tilde{\lambda}$ similarly. Then $\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}, \tilde{\lambda}$ are zero mean Gaussian random variables:

$$E[\alpha\beta\gamma\lambda] = E[(\tilde{\alpha} + \bar{\alpha})(\tilde{\beta} + \bar{\beta})(\tilde{\gamma} + \bar{\gamma})(\tilde{\lambda} + \bar{\lambda})] \tag{D.10}$$

Multiplying this out gives $2^4 = 16$ terms. The expectation of the product of an odd number of zero-mean random variables is zero. Therefore, retaining only the terms with an even number of random variables gives:

$$\begin{aligned}
E[\alpha\beta\gamma\lambda] &= E[\tilde{\alpha}\tilde{\beta}\tilde{\gamma}\tilde{\lambda}] + \bar{\alpha}\bar{\beta}E[\tilde{\gamma}\tilde{\lambda}] + \bar{\alpha}\bar{\gamma}E[\tilde{\beta}\tilde{\lambda}] + \bar{\alpha}\bar{\lambda}E[\tilde{\beta}\tilde{\gamma}] \\
&\quad + \bar{\beta}\bar{\gamma}E[\tilde{\alpha}\tilde{\lambda}] + \bar{\beta}\bar{\lambda}E[\tilde{\alpha}\tilde{\gamma}] + \bar{\gamma}\bar{\lambda}E[\tilde{\alpha}\tilde{\beta}] + \bar{\alpha}\bar{\beta}\bar{\gamma}\bar{\lambda} \\
&= E[\tilde{\alpha}\tilde{\beta}]E[\tilde{\gamma}\tilde{\lambda}] + E[\tilde{\alpha}\tilde{\gamma}]E[\tilde{\beta}\tilde{\lambda}] + E[\tilde{\alpha}\tilde{\lambda}]E[\tilde{\beta}\tilde{\gamma}] \\
&\quad + \bar{\alpha}\bar{\beta}E[\tilde{\gamma}\tilde{\lambda}] + \bar{\alpha}\bar{\gamma}E[\tilde{\beta}\tilde{\lambda}] + \bar{\alpha}\bar{\lambda}E[\tilde{\beta}\tilde{\gamma}] \\
&\quad + \bar{\beta}\bar{\gamma}E[\tilde{\alpha}\tilde{\lambda}] + \bar{\beta}\bar{\lambda}E[\tilde{\alpha}\tilde{\gamma}] + \bar{\gamma}\bar{\lambda}E[\tilde{\alpha}\tilde{\beta}] + \bar{\alpha}\bar{\beta}\bar{\gamma}\bar{\lambda}
\end{aligned} \tag{D.11}$$

Factoring, and using the fact that:

$$E[\alpha\beta] = E[\tilde{\alpha}\tilde{\beta}] + \bar{\alpha}\bar{\beta} \tag{D.12}$$

proves the lemma.

Q.E.D.

Lemma F: Suppose $\underline{\alpha}, \underline{\beta}, \underline{\gamma}$, and $\underline{\lambda}$ are jointly Gaussian random variables with respective means $\underline{\bar{\alpha}}, \underline{\bar{\beta}}, \underline{\bar{\gamma}}$, and $\underline{\bar{\lambda}}$. Let A, G be arbitrary matrices. Then:

$$\begin{aligned} E[\underline{\bar{\alpha}}^T A \underline{\beta} \underline{\gamma}^T G \underline{\lambda}] &= E[\underline{\alpha}^T A \underline{\beta}] E[\underline{\gamma}^T G \underline{\lambda}] + \text{Tr} \{ A E[\underline{\beta} \underline{\gamma}^T] G E[\underline{\gamma} \underline{\alpha}^T] \} \\ &\quad + \text{tr} \{ A^T E[\underline{\alpha} \underline{\gamma}^T] G E[\underline{\gamma} \underline{\beta}^T] \} - 2 \underline{\bar{\alpha}}^T A \underline{\bar{\beta}} \underline{\bar{\gamma}}^T G \underline{\bar{\lambda}} \quad (\text{D.13}) \end{aligned}$$

Proof:

$$\begin{aligned} E[\underline{\bar{\alpha}}^T A \underline{\bar{\beta}} \underline{\bar{\gamma}}^T G \underline{\bar{\lambda}}] &= E\left[\sum_{i,j,k,l} \alpha_i A_{ij} \beta_j \gamma_k G_{kl} \lambda_l\right] \\ &= \sum_{i,j,k,l} A_{ij} G_{kl} E[\alpha_i \beta_j \gamma_k \lambda_l] \\ &= \sum_{i,j,k,l} A_{ij} G_{kl} \{ E[\alpha_i \beta_j] E[\gamma_k \lambda_l] + E[\alpha_i \gamma_k] E[\beta_j \lambda_l] + E[\alpha_i \lambda_l] E[\beta_j \gamma_k] \} \\ &\quad - 2 \sum_{i,j,k,l} A_{ij} G_{kl} \bar{\alpha}_i \bar{\beta}_j \bar{\gamma}_k \bar{\lambda}_l \quad (\text{D.14}) \end{aligned}$$

The lemma follows directly.

Q.E.D.

Now to derive the covariances of the relative likelihoods L'_m under each hypothesis. Using lemma F:

$$\begin{aligned} \text{Cov}[L'_p, L'_q | H_k] &= E[L'_p L'_q | H_k] - E[L'_p | H_k] E[L'_q | H_k] \\ &= 2 \text{tr} \left\{ \mathbf{V}^{-1} \mathbf{W}_p \mathbf{R}_{pp}^{-1} \mathbf{W}_p^T \mathbf{V}^{-1} E[\underline{s} \underline{s}^T | H_k] \mathbf{V}^{-1} \mathbf{W}_q \mathbf{R}_{qq}^{-1} \mathbf{W}_q^T \mathbf{V}^{-1} E[\underline{s} \underline{s}^T | H_k] \right\} \\ &\quad - 2 (\underline{\bar{\alpha}}_k^T \mathbf{R}_{kp} \mathbf{R}_{pp}^{-1} \mathbf{R}_{pk} \underline{\bar{\phi}}_k) (\underline{\bar{\phi}}_k^T \mathbf{R}_{kq} \mathbf{R}_{qq}^{-1} \mathbf{R}_{qk} \underline{\bar{\phi}}_k) \quad (\text{D.15}) \end{aligned}$$

But under hypothesis H_0 ,

$$E[\underline{s}\underline{s}^T|H_0] = \mathbf{V} \text{ and } \bar{\underline{\phi}}_k = \underline{0} \quad (\text{D.16})$$

Substituting into (D.15):

$$\text{Cov}(L'_p, L'_q|H_0) = 2\text{tr} \left\{ \mathbf{R}_{qp} \mathbf{R}_{pp}^{-1} \mathbf{R}_{pq} \mathbf{R}_{qq}^{-1} \right\} \quad (\text{D.17})$$

Under hypothesis H_k for $k = 1, \dots, N$, equation (D.5) implies that:

$$E[\underline{s}\underline{s}^T|H_k] = \mathbf{W}_k \bar{\underline{\phi}}_k \bar{\underline{\phi}}_k^T \mathbf{W}_k^T + \mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T \quad (\text{D.18})$$

Therefore:

$$\begin{aligned} & \text{Cov}(L'_p, L'_q|H_k) \\ &= 2\text{tr} \left\{ \mathbf{R}_{pp}^{-1} [\mathbf{R}_{pk} (\bar{\underline{\phi}}_k \bar{\underline{\phi}}_k^T - \Delta \Phi_k) \mathbf{R}_{kq} + \mathbf{R}_{pq}] \mathbf{R}_{qq}^{-1} [\mathbf{R}_{qk} (\bar{\underline{\phi}}_k \bar{\underline{\phi}}_k^T - \Delta \Phi_k) \mathbf{R}_{kp} + \mathbf{R}_{qp}] \right\} \\ & \quad - 2(\bar{\underline{\phi}}_k^T \mathbf{R}_{kp} \mathbf{R}_{pp}^{-1} \mathbf{R}_{pk} \bar{\underline{\phi}}_k) (\bar{\underline{\phi}}_k^T \mathbf{R}_{kq} \mathbf{R}_{qq}^{-1} \mathbf{R}_{qk} \bar{\underline{\phi}}_k) \end{aligned} \quad (\text{D.19})$$

The formulas for the white noise case are found by substituting:

$$\mathbf{V} = \alpha_V^2 \mathbf{I} \text{ and } \Phi_k = \alpha_k^2 \mathbf{I} \text{ and } R_{km} = \frac{\tau_{km}}{\alpha_V^2} \mathbf{I} \quad (\text{D.20})$$

Appendix E

Mean and Variance of Processor Output Estimates

First we compute the expected value of the estimates of the processor outputs under each hypothesis. Under H_0 :

$$\begin{aligned}
 E[\hat{\underline{y}}|H_0] &= E[\underline{y}|H_0] + \Phi_Y \tilde{\mathbf{W}}^T \mathbf{V}^{-1} E[\underline{s}|H_0] \\
 &= \underline{\bar{y}} + 0 \\
 &= \underline{\bar{y}}
 \end{aligned} \tag{E.1}$$

Under H_k for $k = 1, \dots, N$, taking the expectation of (C.25):

$$\begin{aligned}
 E[\hat{\underline{y}}|H_k, \underline{\bar{\phi}}_k] &= E[\underline{y}|H_k, \underline{\bar{\phi}}_k] - \mathbf{I}_k E[\hat{\underline{\phi}}_k|H_k, \underline{\bar{\phi}}_k] + \bar{\Phi}_Y \tilde{\mathbf{W}}^T \mathbf{V}^{-1} E[\underline{s} + \mathbf{W}_k \hat{\underline{\phi}}_k|H_k, \underline{\bar{\phi}}_k] \\
 &= (\underline{\bar{y}} + \mathbf{I}_k \underline{\bar{\phi}}_k) - \mathbf{I}_k \underline{\bar{\phi}}_k + \bar{\Phi}_Y \tilde{\mathbf{W}}^T \mathbf{V}^{-1} (-\mathbf{W}_k \underline{\bar{\phi}}_k + \mathbf{W}_k \underline{\bar{\phi}}_k) \\
 &= \underline{\bar{y}}
 \end{aligned} \tag{E.2}$$

Under hypothesis H_0 , we calculate the variance of $\hat{\underline{y}}$ as follows: Rewriting (C.13):

$$\hat{\underline{y}} = (\mathbf{I}\Phi_Y \tilde{\mathbf{W}}^T \mathbf{V}^{-1}) \begin{pmatrix} \underline{y} \\ \underline{s} \end{pmatrix} \quad (\text{E.3})$$

From the Gaussian formula in (C.2):

$$\begin{aligned} \text{Var}(\hat{\underline{y}}|H_0) &= (\mathbf{I}\Phi_Y \tilde{\mathbf{W}}^T \mathbf{V}^{-1}) \begin{bmatrix} \Phi_Y & -\Phi_Y \tilde{\mathbf{W}}^T \\ -\tilde{\mathbf{W}}\Phi_Y & \mathbf{V} \end{bmatrix} \begin{pmatrix} \mathbf{I} \\ \mathbf{V}^{-1}\tilde{\mathbf{W}}\Phi_Y \end{pmatrix} \\ &= \Phi_Y - \Phi_Y \tilde{\mathbf{W}}^T \mathbf{V}^{-1} \tilde{\mathbf{W}} \Phi_Y \end{aligned} \quad (\text{E.4})$$

Under hypothesis H_k for $k = 1, \dots, N + C$, each component of $\hat{\underline{y}}$ has value:

$$\begin{aligned} \hat{\underline{y}}_m &= \begin{cases} \underline{y}_m + \Phi_m \mathbf{W}_m^T \mathbf{V}^{-1} (\underline{s} + \mathbf{W}_k \hat{\underline{\phi}}_k) & \text{for } m \neq k \\ \underline{y}_m - \hat{\underline{\phi}}_k & \text{for } m = k \end{cases} \\ &= \begin{cases} \underline{y}_m + \Phi_m \mathbf{W}_m^T \mathbf{I} \mathbf{P}_k \underline{s} & \text{for } m \neq k \\ \underline{y}_k + (\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} \mathbf{W}_k^T \mathbf{V}^{-1} \underline{s} & \text{for } m = k \end{cases} \end{aligned} \quad (\text{E.5})$$

where:

$$\mathbf{P}_k = \mathbf{I} - \mathbf{W}_k (\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} \mathbf{W}_k^T \mathbf{V}^{-1} \quad (\text{E.6})$$

is a projection operator which is orthogonal to $\mathbf{W}_k$; thus $\mathbf{P}_k \mathbf{W}_k = 0$. The following subcomputations are useful in evaluating the covariance:

$$\begin{aligned} \text{Cov}(\underline{y}_m, \underline{y}_p | H_k, \bar{\underline{\phi}}_k) &= \text{Cov}(\underline{\phi}_m, \underline{\phi}_p | H_k, \bar{\underline{\phi}}_k) \\ &= \Phi_m \sigma_{mp} \text{ for } m \neq k, p \neq k \end{aligned} \quad (\text{E.7})$$

$$\begin{aligned} \text{Cov}(\underline{s}, \underline{y}_m | H_k, \bar{\underline{\phi}}_k) &= \text{Cov} \left(\left(- \sum_{l=1}^N \mathbf{W}_l \underline{\phi}_l \right), \underline{\phi}_m | H_k, \bar{\underline{\phi}}_k \right) \\ &= - \sum_{l=1}^N \text{Cov}(\mathbf{W}_l \underline{\phi}_l, \underline{\phi}_m | H_k, \bar{\underline{\phi}}_k) \end{aligned}$$

$$= \begin{cases} -\mathbf{W}_m \bar{\Phi}_m & \text{for } m \neq k \\ -\mathbf{W}_k \bar{\Phi}_k & \text{for } m = k \end{cases} \quad (\text{E.8})$$

$$\text{Cov}(\underline{s}, \underline{s} | H_k, \bar{\phi}_k) = \mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T \quad (\text{E.9})$$

Using these results, we can derive the covariances of $\hat{\underline{y}}_k$. For $m \neq k, p \neq k$:

$$\begin{aligned} \text{Cov}(\hat{\underline{y}}_m, \hat{\underline{y}}_p | H_k, \bar{\phi}_k) &= \Phi_m \sigma_{mp} - 2\Phi_m \mathbf{W}_m^T \mathbf{V}^{-1} \mathbf{P}_k \mathbf{W}_p \Phi_p \\ &\quad + \Phi_m \mathbf{W}_m^T \mathbf{V}^{-1} \mathbf{P}_k (\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T) \mathbf{P}_k^T \mathbf{V}^{-1} \mathbf{W}_p \Phi_p \\ &= \Phi_m \sigma_{mp} - \Phi_m \mathbf{W}_m^T \mathbf{V}^{-1} \mathbf{P}_k \mathbf{W}_p \Phi_p \\ &= \Phi_m \sigma_{mp} - \Phi_m [\mathbf{R}_{mp} - \mathbf{R}_{mk} \mathbf{R}_{kk}^{-1} \mathbf{R}_{kp}] \Phi_p \end{aligned} \quad (\text{E.10})$$

For $m \neq k$:

$$\begin{aligned} \text{Cov}(\hat{\underline{y}}_m, \hat{\underline{y}}_k | H_k, \bar{\phi}_k) &= \Phi_m \sigma_{mk} - \Phi_m \mathbf{W}_m^T \mathbf{V}^{-1} \mathbf{P}_k \mathbf{W}_k \bar{\Phi}_k \\ &\quad - \Phi_m \mathbf{W}_m^T \mathbf{V}^{-1} \mathbf{W}_k (\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} \\ &\quad + \Phi_m \mathbf{W}_m^T \mathbf{V}^{-1} \mathbf{P}_k (\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T) \mathbf{V}^{-1} \mathbf{W}_k (\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} \\ &= -\Phi_m \mathbf{W}_m^T \mathbf{V}^{-1} \mathbf{W}_k (\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} \\ &= -\Phi_m \mathbf{R}_{mk} \mathbf{R}_{kk}^{-1} \end{aligned} \quad (\text{E.11})$$

Finally:

$$\begin{aligned} \text{Cov}(\hat{\underline{y}}_k, \hat{\underline{y}}_k | H_k, \bar{\phi}_k) &= \bar{\Phi}_k - (\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} \mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k \bar{\Phi}_k \\ &\quad - \bar{\Phi}_k \mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k (\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} \\ &\quad + (\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} \mathbf{W}_k^T \mathbf{V}^{-1} (\mathbf{V} - \mathbf{W}_k \Delta \Phi_k \mathbf{W}_k^T) \mathbf{V}^{-1} \mathbf{W}_k (\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} \\ &= -\bar{\Phi}_k + (\mathbf{W}_k^T \mathbf{V}^{-1} \mathbf{W}_k)^{-1} - \Delta \Phi_k \\ &= \mathbf{R}_{kk}^{-1} - \Phi_k \end{aligned} \quad (\text{E.12})$$

Appendix F

Proof of $WW^T = I$ For Complete Sets of Weight Vectors

Let us construct the weight vectors in the following manner. Let each weight $w_{m,k}$ be a small integer in the symmetrical range $-L$ to $+L$. Form all possible $(2L+1)^C$ vectors of this type. Now eliminate from this set the zero vector, and any vector which is an integral multiple of another vector. Now note that:

$$WW^T = \sum_k \mathbf{W}_k \mathbf{W}_k^T \quad (\text{F.1})$$

where the $\mathbf{W}_k$ are the individual weight vectors. Examine the (i, j) block component of this matrix for any $i \neq j$. Suppose there is some weight vector $\mathbf{W}_k$ which has weight values $\alpha_i \mathbf{I}$ and $\beta_j \mathbf{I}$ in the i^{th} and j^{th} positions. By symmetry, there must be another weight vector $\mathbf{W}_m$ which either has weight values $\alpha_i \mathbf{I}$ and $-\beta_j \mathbf{I}$, or else $-\alpha_i \mathbf{I}$ and $\beta_j \mathbf{I}$ in these positions. (This is just the definition of a complete set of weight vectors.) Then the (i, j) element of $\mathbf{W}_k \mathbf{W}_k^T + \mathbf{W}_m \mathbf{W}_m^T$ is $(\alpha_i \beta_j - \alpha_i \beta_j) \mathbf{I} = 0$. Since the weight vectors can all be grouped into pairs in this manner, the (i, j) element of WW^T must be zero for $i \neq j$.

The same proof also holds if the weights are chosen to be complex integers chosen from a symmetrical range.

Bibliography

- [Breuer 83] M. A. Breuer, A. A. Ismaeel, "Roving Emulation as a Fault Detection Mechanism," Proc. 13th Fault-Tolerant Computing Symp., June 1983, pp. 206-215.
- [de Sousa 78] P. T. de Sousa, F. P. Mathur, "Sift-Out Modular Redundancy," IEEE Trans. Computer, Vol. C-27, No. 7, July 1978, pp. 624-627.
- [Etzel 80] M. H. Etzel, W. K. Jenkins, "Redundant Residue Number Systems for Error Detection and Correction in Digital Filters," IEEE Trans. on ASSP, Vol. ASSP-28, No. 5, Oct. 1980, pp. 538-544.
- [Faithi 83] E. T. Faithi, M. Krieger, "Multiple Microprocessor Systems: What, Why, and When," Computer, Vol. 18, No. 3, March 1983, pp. 23-32.
- [Hamming 50] R. W. Hamming, "Error Detecting and Error Correcting Codes," Bell System Technical J., Vol. 26, No. 2, Apr. 1950, pp. 147-160.
- [Helstrom 84] C. W. Helstrom, "Probability and Stochastic Processes for Engineers," Macmillan Publishing 1984.
- [Huang 82] K. H. Huang, and J. A. Abraham, "Low Cost Schemes for Fault Tolerance in Matrix Operations with Processor Arrays," Proc. 1982 Int'l Conf. Fault-Tolerant Computing, pp. 330-338

- [Huang 84] K. H. Huang, and J. A. Abraham, "Algorithm-Based Fault Tolerance for Matrix Operations," IEEE Trans. on Computers, Vol. c-33, No. 6, Jun. 1984.
- [Johnson 84] B. W. Johnson, "Fault-tolerant Microprocessor-based Systems," IEEE Micro Vol. 4, No. 6, Dec. 1984, pp. 44-53.
- [Jou 84] J. Y. Jou, and J. A. Abraham, "Fault-Tolerant Matrix Operations on Multiple Processor Systems Using Weighted Checksums," SPIE Vol. 495 Real Time Signal Processing, 1984, pp. 94-101.
- [Jou 86] J. Y. Jou, and J. A. Abraham, "Fault-Tolerant Matrix Arithmetic and Signal Processing on Highly Concurrent Computer Structure," Proc. IEEE, Vol. 74, No. 5, May 1986, pp. 732-741.
- [Kohavi 78] Z. Kohavi, "Switching and Finite Automata Theory," McGraw-Hill, New York, 1978.
- [Losq 76] J. Losq, "A Highly Efficient Redundancy Scheme: Self-Purging Redundancy," IEEE Trans. Computers, Vol C-25, No. 6, June 1976, pp. 569-578.
- [Musicus 88] Bruce R. Musicus, William S. Song, "Fault-Tolerant Architecture for A Parallel Digital Signal Processing Machine," Unpublished.
- [Nelson 82] V. P. Nelson, B. D. Carroll, "Fault-Tolerant Computing (A Tutorial)," AIAA Fault Tolerant Computing Workshop, Nov. 1982.
- [Oppenheim 75] A. V. Oppenheim, R. W. Schafer, "Digital Signal Processing," Prentice-Hall, Englewood Cliffs, New Jersey, 1975.
- [Siewiorek 82] D. P. Siewiorek, R. S. Swarz, "The Theory and Practice of Reliable System Design," Digital Press, Bedford, Mass., 1982

- [Song 87] William S. Song, Bruce R. Musicus, "A Fault-Tolerant Architecture for A Parallel Digital Signal Processing Machine," Proc. IEEE International Conference on Computer Design, 1987, pp. 385-390.
- [Taylor 84] F. J. Taylor, "Residue Arithmetic: A Tutorial with Examples," Computer, May 1984, pp. 50-62.
- [von Neumann 56] J. von Neumann, "Probabilistic Logics and The Synthesis of Reliable Organisms from Unreliable Components," Automata Studies, Princeton University Press, 1956, pp.43-98.
- [Wensley 78] J. H. Wensley, "SIFT: Design and Analysis of a Fault Tolerant Computer for Aircraft Control," Proc. IEEE, Vol. 66, No. 10, Oct. 1978, pp. 1240-1255.
- [Welch 69] P. D. Welch, "A Fixed-Point Fast Fourier Transform Error Analysis," IEEE Trans. on Audio and Elec., Vol. Au-17, No.2, Jun. 1969.