Summer 1995

Fault Tolerant Pipelined Adaptive Systems

Bhasker Reddy Allam
Old Dominion University

Fault Tolerant Pipelined Adaptive Systems

by

Bhasker Reddy. Allam
B.Tech. (ECE), July 1993, Jawaharlal Nehru Technological University, India

A Thesis Submitted to the Faculty of
Old Dominion University in the Partial Fulfillment
of the Requirements of the Degree of

MASTER OF SCIENCE
ELECTRICAL ENGINEERING

OLD DOMINION UNIVERSITY
August, 1995

Approved by:

Dr. Martin D. Meyer Director

Dr. James F. Leathrum

Dr. John W. Stoughton

ABSTRACT

Fault Tolerant Pipelined Adaptive Systems

Bhasker Reddy. Allam
Old Dominion University, 1994
Director: Dr. Martin D. Meyer

A fault tolerant pipelined architecture for high sampling rate adaptive filters is presented in this thesis. The architecture is based on the computational requirements of delayed LMS and lattice adaptive filters. It offers robust performance in the presence of single hardware faults, and software faults resulting from numerical instability. Two different architectures are proposed. One allows a graceful degradation in system performance in the event of a fault, and the other uses a module replacement strategy to recover from faults without decreasing performance. Analysis of the steady state error increase due to filter reconfiguration is presented. The reliability of the proposed system is analyzed and compared to existing implementation strategies. Methods for fault detection, fault location and recovery via hardware reconfiguration are also discussed. Simulation results illustrating recovery from processor faults are presented.

To my parents and Family

Acknowledgment

I would like to express my sincere gratitude to my advisor, Dr. Martin D. Meyer, for his useful guidance and genuine personal warmth. He has been a constant source of encouragement for me in my research work and in my studies here at ODU. I would like to thank my committee members, Dr James F. Leathrum and Dr John W. Stoughton for their time and constructive criticism. I would also like to thank all my friends at ODU for their friendship and encouragement.

Special thanks are due to my family for the constant support and encouragement they have given me all along my career.

Table of Contents

1 Introduction **1**

1.1 Research Focus 1

1.2 Overview 1

1.3 Research Objective 3

1.4 Thesis organization 5

2 Theory of Adaptive Filters **6**

2.1 Introduction 6

2.2 Normal Equations 9

2.3 LMS Algorithm 11

2.4 Delayed LMS Algorithm 12

2.5 Lattice Filter 17

2.5.1 Gradient Lattice Filter 19

2.5.2 Least Squares Lattice Filter 22

2.6 Application (Waveform coding) 24

3 Fault Tolerance Concepts, Algorithm and Implementation **29**

3.1 Fault Tolerance Concepts 29

3.1.1 Series System 32

3.1.2 Parallel Systems 32

3.2 Fault-Tolerant Algorithm and Implementation 33
3.3 Implementation 39
3.3.1 Fault tolerance by gracefully degrading the system performance 39
3.3.2 Fault-Tolerance using redundant processing modules 41

4 Performance and Simulation Results **43**
4.1 Introduction 43
4.2 Increase in the minimum MSE with the decrease in filter order 44
4.3 Performance 52

5 Conclusions **58**
5.1 Summary 58
5.2 Future research 59

Bibliography **61**

Appendix **64**

List of Figures

2.1 Identification of unknown system. 8

2.2 Structure of adaptive transversal filter. 13

2.3 LMS adaptive filter used as a linear predictor. 14

2.4 Pipelined adaptive filter. 18

2.5 Cascade of Nth order gradient lattice adaptive filter. 20

2.6 Single stage of a gradient adaptive lattice filter. 21

2.7 Performance of comparision of various adaptive filters (Order = 2). . 25

2.8 Adaptive differential pulse code modulation. 28

3.1 Divergence of a pipelined adaptive filter in the event of a fault. 36

3.2 Fault tolerant pipelined adaptive filter 38

3.3 The switching hardware and the latch. 40

3.4 Fault tolerant pipelined adaptive filter structure with redundant processors 42

4.1 Simulation of LMS adaptive filter 45

4.2 Simulation of DLMS adaptive filter 46

4.3 Simulation of a hardware fault and subsequent recovery 48

4.4 Illustration of increase in steady state error with the decrease in filter order 49

4.5 Misadjustment curve with respect to the order of the filter. 53

4.6 Reliability curves for pipelined systems with and without fault tolerance 55

4.7 Reliability curves for pipelined systems with and without fault tolerance 56

4.8 Reliability curves for pipelined systems with and without fault tolerance 57

CHAPTER 1

Introduction

1.1 Research Focus

In this thesis a **fault tolerant algorithm** for adaptive filtering is proposed, and an architecture implementing the proposed algorithm is presented. The pipelined architecture model proposed by Meyer et.al [13] for high sampling rate adaptive filters forms the theoretical basis for this research. The architecture offers robust performance in the presence of single hardware faults, and software faults resulting from numerical instability. The algorithm for fault detection, fault location and recovery via hardware reconfiguration is presented. Simulation results illustrating the recovery from a processor fault are presented.

1.2 Overview

The term *filter* is often used to describe a device, in the form of a piece of physical hardware or computer software, that is applied to a set of noisy data to extract information about the prescribed quantity of interest. A filter is said to be linear if the filtered or the predicted quantity of interest at the output of the filter is a linear function of the filter input. The solution to the *linear filtering problem* is

based on the assumed availability of certain statistical parameters (such as mean and correlation functions) of the input data. The requirement is to design a filter which accepts noisy data as input and minimizes the effects at the filter output according to some statistical criterion. For stationary inputs, the resulting solution is known as the **Wiener filter** which is optimum in the minimum mean square sense [31], [7]. The design of Wiener filters requires a *priori* information about the statistics of the data to be processed. When this information is not known completely however, it may not be possible to design an optimum Wiener filter. Hence it is necessary to develop a device that is *self designing*; that is, a device that adapts its parameters to the changing statistical characteristics. A device that performs satisfactorily in an environment where complete knowledge of the signal statistics is not known is the **adaptive filter** [31], [28]. The adaptive filter relies on a *recursive algorithm* for its operation. The algorithm starts from a predetermined set of initial conditions in an environment where signal statistics are not known completely, and converges to the optimum Wiener solution after successive iterations in some statistical sense.

Adaptive signal processing techniques have been successfully used recent years [7], [2], [9]. Advances in signal processing theory, coupled with the experiences gained from applications, have caused these techniques to become more and more refined and sophisticated. Consistent effort by many researchers in this field has improved the understanding of adaptive filters to a great extent. These efforts have resulted in the development of some very fast and computationally efficient adaptive algorithms [31], [9], [7]. The ability of an adaptive filter to operate effectively in an unknown environment and also track time variations of the input statistics has rendered it a powerful device. This fact is illustrated by applications in such diverse fields as communications, control, sonar, and seismology [7], [9], [30].

One of the most well known adaptive algorithms is the least mean squares(LMS) algorithm [31]. In this algorithm, the estimated signal for each input sample is computed and then subtracted from a desired signal. The error signal, which is the difference signal between the estimated and the desired signal, is used to update the tap coefficients of the transversal filter. However, the inherent delay associated with the computation of the tap coefficients in the LMS algorithm imposes a critical limit on its potential throughput.

Over the years, many techniques have been developed to accommodate high sampling rates [19]. These techniques however, were based on the adaptive lattice filter, which due to its recursive nature lends itself easily to pipelining [19], [14], [27]. The least mean squares algorithm however, does not possess this recursive structure making it difficult to pipeline. Recently however, it has been demonstrated that it is possible to introduce some fixed delay in the coefficient adaptation [23]. This introduction of delay into the LMS coefficient update equations results in an order recursive structure, thus allowing the realization of high sampling rate pipelined adaptive filters [13].

1.3 Research Objective

Adaptive filters are used in many important applications, such as removing radar clutter, adaptive beamforming, sonar, channel equalization and biomedical signal processing. Many of these applications are critical real-time applications. These applications use high bit rates and require the systems to be highly reliable. For example, consider the *waveform coding* application where the ADPCM encoder/decoder is mounted on a satellite. In this case, the specifications will dictate that the coder/decoder be able to support high bit rates and at the same time

it should also be rugged. That is, it should be able to withstand the hostile environment and still perform satisfactorily. Consequently, it is desirable that such a system incorporate some form of fault tolerance in order to provide reliable performance over a long life span.

The realization of transversal adaptive filters in real time is important in many of the above mentioned applications. Because of the poor performance of the LMS algorithm in real time, recent attention has been given to implementations which introduce adaptation delay and use multiple processing elements and pipelining to achieve the required processing speed. However, the problem associated with the pipelined implementations of adaptive filters(transversal or lattice) is that if an intermediate *processing module* fails, then all of the subsequent processing modules in the pipeline are rendered ineffective. This causes the filter to diverge. Moreover, as will be shown later, the reliability of a series system decreases exponentially depending on the number of processing modules present in the system. Also, the delay introduced to facilitate pipelining can increase roundoff error leading to problems of numerical instability, particularly when finite word lengths are used. This is particularly troublesome in the least squares lattice filter which suffers from known numerical instability. The objective of this research was to develop an efficient and effective fault tolerant system to reliably implement the pipelined adaptive filter structure.

The fault tolerant algorithm developed in this thesis addresses the problem of failure of an intermediate processing module through hardware reconfiguration. The objective of the algorithm is to detect the occurrence of a fault, locate the faulty processing module which was responsible for the fault, and to eliminate the faulty processing module from the pipeline. In this thesis, two different approaches were

pursued to achieve fault tolerance. In the first method this objective is achieved by making use of redundant communication links among the neighboring processing modules(PM). The redundant links are used to bypass the faulty PM in case of a fault. The *switching mechanism* which accomplishes this task is also presented. The analysis of the increase in the minimum mean squared error due to the decrease in the order of the filter is also presented. The second method makes use of redundant processing modules and redundant links. The faulty processor is replaced with a redundant processor, thus avoiding any degradation in performance after the occurrence of a fault. The reconvergence of the adaptive filter after the occurrence of a fault is illustrated by simulations.

1.4 Thesis organization

Chapter two presents the theoretical background on adaptive filters. It describes the Wiener-Hopf equations, the LMS and the DLMS algorithms [31], [23]. Chapter three discusses the basic theory of fault tolerance with respect to multiprocessor systems [20], [22], [11]. The necessity for fault tolerance in pipelined adaptive filters is presented. The fault tolerant algorithm, its description, and the assumptions made are also presented. Finally at the end of the chapter the complete fault tolerant system is presented. In chapter four, the analytical proof illustrating the increase in the minimum mean squared error with respect to decrease in the order of the filter is presented. Simulations illustrating the reconvergence after reconfiguration are also presented. A summary of results and topics for future research are presented chapter five.

CHAPTER 2

Theory of Adaptive Filters

This chapter gives a brief introduction to the concepts of adaptive filters. The definitions of the terms used in the later chapters are presented and illustrated here. Section 2.2 presents the fundamental theory of adaptive filters and the mathematics involved in the derivation of the normal equations. Sections 2.3, 2.4 and 2.5 describe various adaptive algorithms in use. Section 2.3 talks about the well known least mean squares (LMS) adaptive algorithm, section 2.4 talks about delayed least mean squares(DLMS) algorithm and finally in section 2.5 a discussion of lattice filters is presented. Section 2.6 describes an important application of adaptive filters called *waveform coding*.

2.1 Introduction

A filter, in general, is defined as a system which is used to extract information about a quantity of interest from a set of noisy data. Most filters which one encounters in the field of communications and signal processing, are filters which have an internal structure that does not change with time. An **adaptive filter** on the other hand, is a filter whose internal parameters allow us to control the transfer function over a useful range. It uses an *adaptation algorithm* to enable the filter transfer func-

tion to track some important feature of the external environment. Adaptive filters have been put to use in many fields such as telecommunications, biomedical signal processing, geophysical signal processing, sonar processing and in the elimination of radar clutter. The factor that dictates the use of adaptive filters in the above mentioned fields is that some element of the problem is unknown and must therefore be learned, or some parameter of the system is changing with time in an unknown manner and hence must be tracked. In this thesis, the *systems identification* application has been used as a model to explain various concepts of adaptive signal processing. However, the ideas presented here apply to a wide variety of applications. Also, a brief discussion about an application which uses adaptive an filter for the purpose of predicting the input signal has been included at the end of this chapter.

Systems identification has been a focus of attention in the fields of control and signal processing for many years. The procedure of specifying an unknown system in terms of experimental evidence using a set of measurements of the input or output signals is called *identification.* Adaptive identification involves the procedure of updating our knowledge of the system based on the most recent information about the system. Adaptive signal processing algorithms attempt to optimize a performance measure that is a function of the unknown parameters to be identified.

There are two broad categories of adaptive signal processing: (1) stochastic, and (2) exact. Category (2) refers to adaptive algorithms which are based upon the actual or exact data signals acquired. Adaptive techniques which fall into category (1) are based upon algorithms that use the statistical properties of the data signals. The primary statistical measure used is the ensemble average or the mean of the squared prediction error function. This performance measure is shortened simply to the mean-square error (MSE).

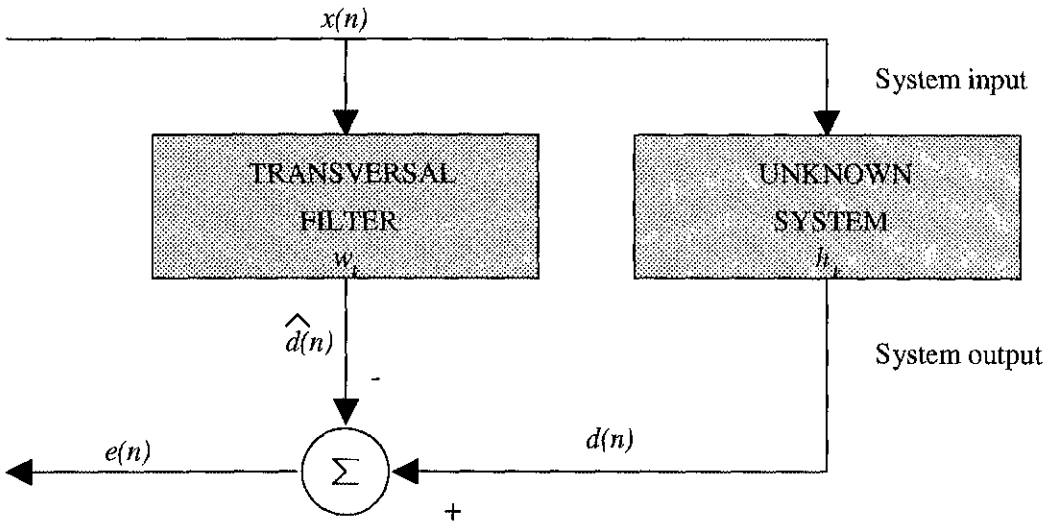

Figure 2.1 Identification of unknown system.

2.2 Normal Equations

In many engineering problems it is necessary to predict the output sample of an unknown system *d(n)* (also called as the desired signal), using the input sample x(n) at time *n*. This type of systems identification is shown in figure 2.1. Since it is not possible to predict the desired signal d(n) exactly, a prediction error signal is generated at time *n*. This error signal e(n) is given by,

$$e(n) = d(n) - \hat{d}(n) \tag{2.1}$$

where $\hat{d}(n)$ is the predicted signal. In the Wiener theory, the *minimum mean-square error* criterion is used to optimize the prediction filter rather than just the *estimation error e(n)*. Since the desired signal may be the result of a random or stochastic process, the error signal is also stochastic. The statistical property well suited as a performance measure is the ensemble average or the expectation of the squared error sequence. Hence the mean-squared error (MSE) is defined as,

$$\varepsilon(n) = E\{e^2(n)\} = E\{[d(n) - \hat{d}(n)]^2\} \tag{2.2}$$

By minimizing $\varepsilon(n)$, we can obtain an optimum linear filter in the minimum mean-squared sense.

For a given set of N filter coefficients $\mathbf{w}(n)$, and a data sequence x(n), the prediction of the desired signal may be computed as,

$$\hat{d}(n) = \sum_{i=0}^{N-1} w_i(n)x(n-i) = \mathbf{w}_N^T(n)\mathbf{x}_N(n) \tag{2.3}$$

$$\mathbf{w}_N^T = [w_o(n), w_1(n), ..., w_{N-1}(n)] \tag{2.4}$$

$$\mathbf{x}_N^T = [x(n), x(n-1), ..., x(n-N+1)] \tag{2.5}$$

where $\mathbf{w}_N^T$ and $\mathbf{x}_N^T$ are the weight coefficients and data samples respectively. Substituting for $\hat{d}(n)$ into the definition of MSE (2.2), the MSE becomes,

$$\varepsilon(\mathbf{w}_N) = E\{e^2(n)\} = E\{[d(n) - \mathbf{w}_N^T \mathbf{x}_N(n)]^2\} \tag{2.6}$$

where $\mathbf{w}_N = \mathbf{w}_N(n)$. The MSE in (2.6) is denoted as a function of $\mathbf{w}_N$ because for each value of the filter coefficient there results a corresponding value of MSE. Expanding equation (2.6) we can write,

$$\varepsilon(\mathbf{w}_N) = E\{d^2(n)\} + 2\mathbf{w}_N^T E\{d(n)\mathbf{x}_N(n)\} + \mathbf{w}_N^T E\{\mathbf{x}_N(n)\mathbf{x}_N^T(n)\}\mathbf{w}_N \tag{2.7}$$

$$\varepsilon(\mathbf{w}_N) = \sigma_d^2 - 2\mathbf{w}_N^T \mathbf{p}_N + \mathbf{w}_N^T \mathbf{R}_{NN} \mathbf{w}_N \tag{2.8}$$

where

$$\sigma_{\mathbf{d}}^{\mathbf{2}} = E\{d^2(n)\} \tag{2.9}$$

$$\mathbf{p}_N = E\{d(n)\mathbf{x}_N(n)\} \tag{2.10}$$

$$\mathbf{R}_{NN} = E\{\mathbf{x_N}(n)\mathbf{x}_N^T(n)\} \tag{2.11}$$

are the mean squared power of the desired signal, the cross-correlation vector, and the auto-correlation vector respectively [2]. From equation (2.7) it is evident that the input vector $\mathbf{x}_N$ and the desired signal *d(n)* are jointly stationary, and the mean-squared error $\varepsilon(\mathbf{w}_N)$ is a second order function of the tap-weight vector $\mathbf{w}$. Hence the dependence of the MSE $\varepsilon(\mathbf{w}_N)$ on the elements of the tap-weight vector $\mathbf{w}$ may be visualized as a bowl shaped surface with a *unique bottom.* This surface is called the *error performance* surface of the transversal filter. The objective of the adaptive filter is to track the *bottom or the minimum point* on the error performance surface, where the MSE $\varepsilon(\mathbf{w}_N)$ attains the *minimum value* (ε_{min}) and the tap-weight vector attains an optimum value $\mathbf{w_o}$. The resultant transversal

filter is said to be optimum in the mean-square sense. The gradient of the error performance surface wiht respect to the wieght vector is given by,

$$\nabla_{\mathbf{w}}[\varepsilon] = \frac{\partial \varepsilon}{\partial \mathbf{w}_N}. \tag{2.12}$$

Substituting for ε in (2.12) we get,

$$\nabla_{\mathbf{w}}[\varepsilon] = -2\mathbf{p}_N + 2\mathbf{R}_{NN}\mathbf{w}_N. \tag{2.13}$$

At the optimum tap-weight vector, the gradient vector equals the null vector. Therefore, equating $\nabla_{\mathbf{w}}[\varepsilon]$ in (2.13) to zero we have

$$\mathbf{R}\mathbf{w}_o = \mathbf{p}. \tag{2.14}$$

Equation (2.14) is the discrete form of the Wiener-Hopf equation or the *normal equation.*

2.3 LMS Algorithm

By definition, the gradient (2.12) points in the direction of the maximum rate of change of the surface at a point on the error-performance surface. That is, the gradient points in the direction of *steepest descent.* The above property implies that given a position on the MSE surface at a time n, one can determine the next position at time *(n+1)* by moving along the direction opposite to the gradient. Applying this property to (2.14) and solving for $\mathbf{w}$,

$$\mathbf{w}_N(n+1) = \mathbf{w}_N(n) - \mu\nabla_{\mathbf{w}}[\varepsilon(n)] \tag{2.15}$$

where μ is a constant. But,

$$\nabla_{\mathbf{w}}[\varepsilon(n)] = \frac{\partial}{\partial \mathbf{w}_N} E\{e^2(n)\} = 2E\{e(n)\frac{\partial e(n)}{\partial \mathbf{w}_N}\} \tag{2.16}$$

By expanding for $e(n)$ according to (2.1) and taking the gradient of $e(n)$ with repect to $\mathbf{w})_N$ is given by

$$\frac{\partial}{\partial \mathbf{w}_N} e(n) = -\mathbf{x}_N(n) \tag{2.17}$$

Substituting equations (2.16) and (2.17) in (2.15)

$$\mathbf{w}_N(n+1) = \mathbf{w}_N(n) + \alpha \hat{E}\{e(n)\mathbf{x}_N(n)\} \tag{2.18}$$

where $\alpha = 2\mu$, and $\hat{E}\{.\}$ is an estimate of the expected value.

The equation given in (2.18) cannot be used in practice because it is not to possible compute the ensemble average given in equation in real-time. In practice, in order to overcome this problem the expectation in (2.18) is approximated with the instantaneous value of the quantity inside the brackets. That is, let $\hat{E}$ be simply given as

$$\hat{E}\{e(n)\mathbf{x}_N(n)\} = e(n)\mathbf{x}_N(n). \tag{2.19}$$

Substituting (2.19) into (2.18) we get

$$\mathbf{w}_N(n+1) = \mathbf{w}_N(n) + \alpha e(n)\mathbf{x}_N(n). \tag{2.20}$$

Equation (2.20) is called the LMS algorithm [31]. Due to its simplicity, the above algorithm has found wide usage in applications that deal with nonstationary data. Figure 2.2 illustrates the structure of an adaptive transversal filter and figure 2.3 illustrates an LMS adaptive filter used in a linear prediction application.

2.4 Delayed LMS Algorithm

As mentioned in the previous, subsection LMS is the most commonly used adaptive transversal filter. In this algorithm the error signal is needed to to update

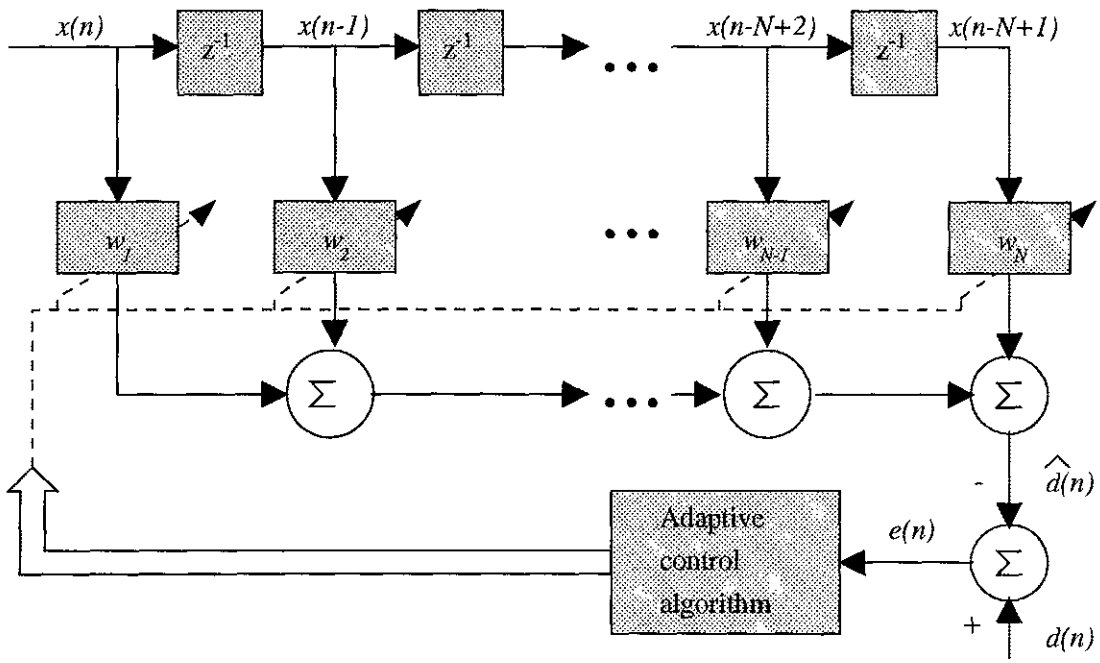

Figure 2.2 Structure of adaptive transversal filter.

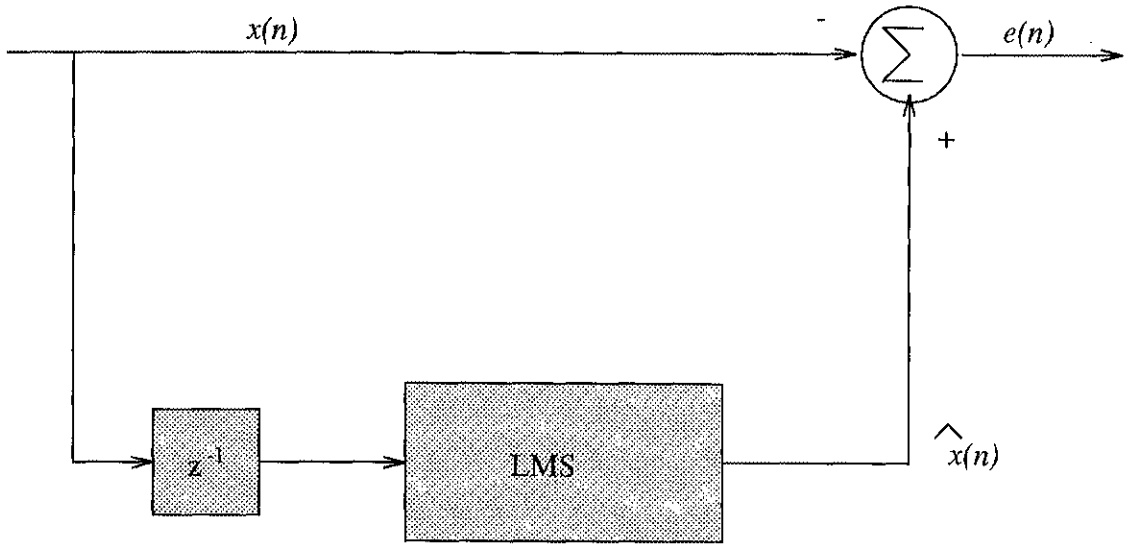

Figure 2.3 LMS adaptive filter used as a linear predictor.

the filter coefficients before the next sample arrives. However, in some real-time applications this imposes a critical limit on the implementation of the LMS algorithm. This problem was studied by Proakis et.al [23], where the introduction of some delay into the coefficient adaptation was proposed. The resulting algorithm, called the *delayed least mean square* (DLMS) algorithm, has been shown to guarantee stable convergence characteristics provided an appropriate adaptation step size is chosen.

Based on a time-shifted version of the DLMS algorithm, Meyer et al. [13]-[14] developed order recursive coefficient update equations which were mapped to a pipeline of application specific processing modules (PMs). In this new algorithm, the computations are structured to be order recursive, resulting in a highly modular pipelined implementation that can provide high sampling rates independent of the order of the filter. Also, the weights are updated locally within each stage. The DLMS algorithm is defined by the following equations:

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \alpha e(n-D)\mathbf{x}(n-D) \tag{2.21}$$

$$e(n) = d(n) - \hat{d}(n) \tag{2.22}$$

$$\hat{d}(n) = \mathbf{x}^T(n)\mathbf{w}(n-1) \tag{2.23}$$

where D is the delay in the weight adaptation, and $\mathbf{w}(n)$ and $\mathbf{x}(n)$ are the weight and input vectors respectively as defined in equations (2.4) and (2.5). The order used here is *(N+1)* in order to simplify the notation in the derivation. Since the filter has *(N+1)* stages, N extra units of delay are introduced into the system. Hence, the delay is made equal to the number of stages (D = N). The time-shifted weight vector can then be defined as follows:

$$\hat{\mathbf{w}}(n) = \begin{pmatrix} w_o(n) \\ w_1(n-1) \\ \vdots \\ w_N(n-N) \end{pmatrix} \tag{2.24}$$

Substituting this time shifted vector into the DLMS algorithm, we have

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}(n-1) + \beta \mathbf{E}(n-N)\hat{\mathbf{x}}(n-N) \tag{2.25}$$

where $\hat{\mathbf{x}}(n-N)$ is the modified input vector, which is defined as

$$\hat{\mathbf{x}}(n-N) = \begin{pmatrix} x(n-N) \\ x(n-N-2) \\ x(n-N-4) \\ \vdots \\ x(n-3N) \end{pmatrix} \tag{2.26}$$

and **E**(n - N) is the *(N+1)x(N+1)* diagonal error matrix. The scalar update equations for the *i*th filter weight can therefore be written as

$$w_i(n-i) \;=\; w_i(n-i-1) + \beta e(n-N-i)x(n-N-2i). \tag{2.27}$$

Since we are now delaying the input to each filter weight, the output at each filter weight is in effect a partial sum of the outputs from the preceding weights. This shifted vector of partial sums of the output may be defined as

$$\mathbf{y}(n) = \begin{pmatrix} y_o(n) \\ y_1(n-1) \\ y_2(n-2) \\ \vdots \\ y_N(n-N) \end{pmatrix} \tag{2.28}$$

Each partial sum $y_i(n-i)$ corresponds to the output of a transversal filter of order i, whose current input is $x(n-i)$, and whose coefficients are $w_o(n-i-1), ..., w_i(n-i-1)$; that is,

$$y_i(n-i) = \sum_{k=0}^{i} x(n-i-k)w_k(n-i-1). \tag{2.29}$$

Equation (2.29) can be split into two parts:(1) sum of product of the terms of the previous it (i-1) samples plus, (2) the sum of product of the it ith sample.

$$y_i(n-i) = [\sum_{k=0}^{i-1} x(n-i-k)w_k(n-i-1)] + x(n-2i)w_i(n-i-1) \tag{2.30}$$

The summation term in (2.30) is equal to $y_{i-1}(n-i)$, and the order recursion for the filter output is finally given by

$$y_i(n-i) = y_{i-1}(n-i) + x(n-2i)w_i(n-i-1). \tag{2.31}$$

Equation (2.31) and (2.27) form the scalar updates to be performed for each weight by processor module i at sample time n. A detailed derivation of these equations is presented in [13]. The corresponding filter structure is given in figure 2.4.

2.5 Lattice Filter

There are numerous ways of solving equation (2.14), and it has been shown in the above discussion that LMS algorithm is one of the ways of finding a solution to (2.14) in terms of **w**. An alternative structure that is used to solve equation (2.14) is the lattice filter. There are two types of lattice filters (1) gradient lattice filters and (2) the least squares lattice filters. The gradient lattice is based on the adaptive techniques considered so far. That is, it attempts to minimize the statistical error measure derived from the data. The least squares lattice on the other hand is based on the error measures derived from the exact data signals acquired.

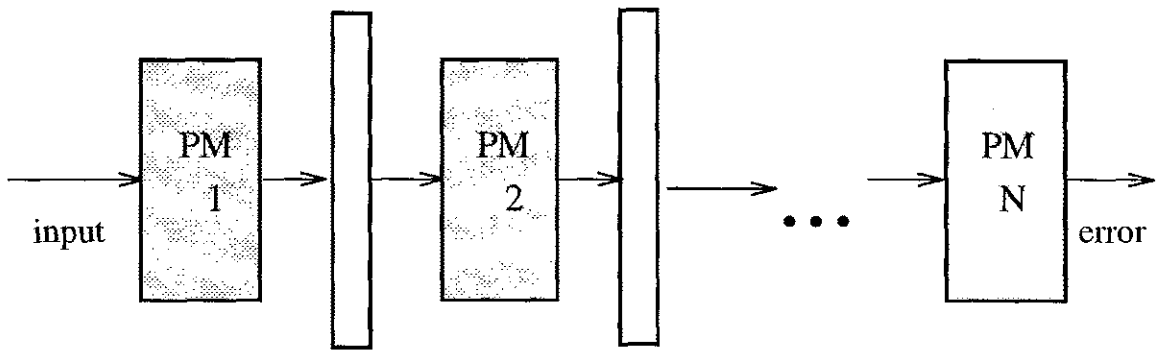

Figure 2.4 Pipelined adaptive filter.

2.5.1 Gradient Lattice Filter

The gradient lattice formulation is a result of using Durbin's algorithm for solving the normal equations. The procedure used for processing the actual data by the lattice filter is to first estimate the autocorrelation coefficients and then use Durbin's recursion for computing the reflection coefficients (k_p), where $1 \leq p \leq N$. However, since the estimation of the autocorrelation is an error prone process, an alternative method used here is to directly estimate the autocorrelation coefficients from the data and carry out the updating on a sample-per-sample basis, as was done in the LMS algorithm.

An Nth order prediction filter consists of N cascaded stages as shown in figure 2.5. The operation of the pth stage of the lattice is shown in figure 2.6 and is defined by the following relations,

$$e_p^f(n) = e_{p-1}^f(n) - k_p e_{p-1}^b(n-1), \tag{2.32}$$

$$e_p^b(n) = e_{p-1}^b(n-1) - k_p e_{p-1}^f(n) \tag{2.33}$$

where $e_p^f(n)$ and $e_p^b(n)$ are the pth order forward prediction error (FPE) and backward prediction error (BPE), respectively. The key idea in the lattice filter is to choose k_p that would minimize the FPE and/or BPE at each stage. However if the input has a non-stationary character, it is necessary that k_p be updated adaptively for each sample. The equation for updating the reflection coefficients is,

$$k_p(n+1) = \frac{2[N_p(n) + e_{p-1}^f(n+1)e_{p-1}^b(n)]}{D_p(n) + [e_{p-1}^f(n+1)]^2 + [e_{p-1}^b(n)]^2} \tag{2.34}$$

$$N_p(n) = N_P(n-1) + e_{p-1}^f(k)e_{p-1}^b(k-1), \tag{2.35}$$

$$D_p(n) = D_p(n-1) + [e_{p-1}^f(k)]^2 + [e_{p-1}^b(k-1)]^2 \tag{2.36}$$

where $N_p(0)$ and $D_p(0)$ are assumed to be zero.

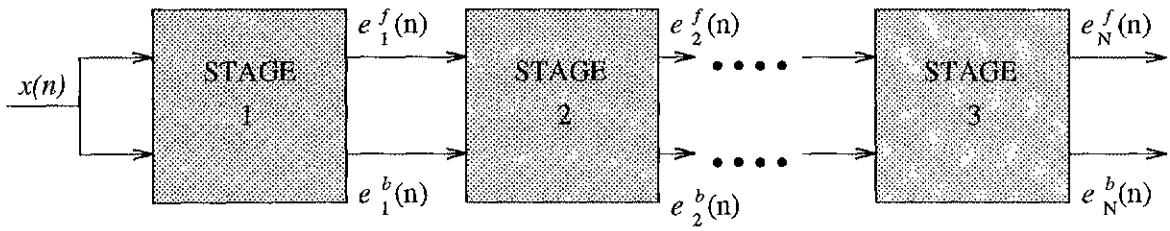

Figure 2.5 Cascade of Nth order gradient lattice adaptive filter.

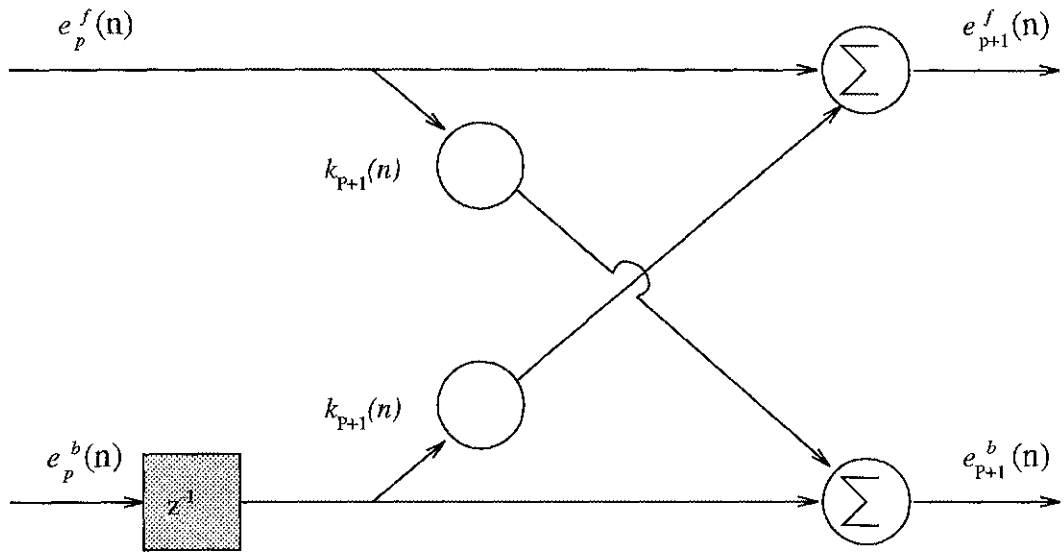

Figure 2.6 Single stage of a gradient adaptive lattice filter.

2.5.2 Least Squares Lattice Filter

As mentioned in the earlier subsection, recursive least squares (RLS) or simply least squares prediction filters are based upon the error measure derived from the exact data signals acquired. In this subsection, the basic RLS filtering problem and the least squares lattice (LSL) filter are briefly discussed. The important differences between the gradient adaptive techniques and LSL filters are also highlighted.

The least squares filtering problem is defined as follows: if d(n) is the desired signal and x(n) is the acquired signal and if a linear prediction filter $\mathbf{w}_N(n)$ is used to predict the desired signal d(i) of the ith sample, then the error made in predicting the desired sample is

$$e(i|n) = d(i) - \mathbf{x}_N^T(i)\mathbf{w}_N(n). \tag{2.37}$$

Here $\mathbf{x}_N(i)$ is the N-length data vector used in the prediction of the desired signal *d(i)* at time i and is defined by

$$\mathbf{x}_N(n) = [x(i), x(i-1), ..., x(i-N+1)]^T \tag{2.38}$$

and $\mathbf{w}_N$ is the weight vector

$$\mathbf{w}_N(n) = [w_o(n), w_1(n), ..., w_N - 1(n)]^T. \tag{2.39}$$

The RLS problem is to find a set of prediction coefficients $\mathbf{w}_N(n)$ such that the *cumulative squared error measure*

$$E(n) = \sum_{i=1}^{n} \lambda^{n-i} e^2(i|n) \tag{2.40}$$

is minimized. The constant λ $(0 \leq \lambda \leq 1)$ used in equation (2.40) is data-weighting factor that may be used to weight recent data more heavily in the RLS computations.

The important difference between the gradient adaptive techniques and the least squares prediction filters is that the MSE measure used in the gradient adaptive

techniques is not actually a function of the data acquired by the processor, but instead depends upon the statistical characterization of the data. However it is obvious from equations (2.37) and (2.40) that the cumulative squared error criterion is in fact a function of the actual data vectors $\mathbf{x}_N(n), \mathbf{x}_N(n-1), ..., \mathbf{x}_N(1)$. Therefore we see that RLS techniques provide prediction filters that are exactly optimal for the acquired data rather than being statistically optimal for a class of data [2].

As mentioned in the previous subsection of gradient lattice filter, the lattice structure results as a consequence of computing the *(m + 1)*th order LS forward linear predictor based upon a knowledge of the *m*th order LS linear predictor. Extending the above concept to lower orders, we finally end up with a single order filter whose output is to be estimated based upon the present input sample. The least squares lattice filter is similar to the gradient lattice filter, as is its implementation shown in figure 2.6. The forward prediction error (FPE) and the backward prediction error (BPE) for an $(m + 1)$th order filter are given by

$$e^f_{m+1}(n) = e^f_m(n) - k^b_{m+1}(n)e^b_m(n-1) \tag{2.41}$$

$$e^b_{m+1}(n) = e^b_m(n) - k^f_{m+1}(n)e^f_m(n) \tag{2.42}$$

where $k^b_{m+1}(n)$, called the backward reflection coefficient, is defined as

$$k^b_{m+1}(n) = \frac{\Delta_{m+1}(n)}{\varepsilon^b_m(n-1)}. \tag{2.43}$$

Similarly $k^f_{m+1}(n)$, called the forward reflection coefficient, is defined as

$$k^f_{m+1}(n) = \frac{\Delta_{m+1}(n)}{\varepsilon^f_m(n)}. \tag{2.44}$$

$\varepsilon^b_m(n-1)$ and $\varepsilon^f_m(n)$ in equations (2.43) and (2.44) are called the BPE and the FPE residuals, and are a measure of the FPE and BPE energy at the *m*th stage of the lattice. They are defined by the following equations

$$\varepsilon^f_{m+1}(n) = \varepsilon^f_m(n) - \frac{\Delta^2_{m+1}(n)}{\varepsilon^b_m(n-1)} \tag{2.45}$$

$$\varepsilon^b_{m+1}(n) = \varepsilon^b_m(n-1) - \frac{\Delta^2_{m+1}(n)}{\varepsilon^f_m(n)}. \tag{2.46}$$

Also, $\Delta_{m+1}(n)$ in equations (2.43) and (2.44) is the partial correlation (PARCOR) coefficient and is defined by the following equation

$$\Delta_{m+1}(n) = \Delta_{m+1}(n-1) + \frac{e^f_m(n)e^b_m(n-1)}{\gamma_m(n-1)} \tag{2.47}$$

where $\gamma_m(n-1)$ is the angle parameter defined by

$$\gamma_{m+1}(n-1) = \gamma_m(n-1) - \frac{[e^b_m(n-1)]^2}{\varepsilon^b_m(n-1)}. \tag{2.48}$$

Equations (2.41) to (2.48) define all the recursions needed to implement the least squares lattice algorithm.

The main advantage of lattice filters is the convergence speed compared to the LMS algorithm. However, this speed-up is at the cost of added computational operations and extra storage requirements. Another important advantage of the lattice filter is that each stage of the filter is modular and hence becomes an excellent candidate for pipelining, in order to support high sampling rates. Figure 2.7 compares the performance of the LMS, gradient lattice and the least squares lattice filters when they were used in the linear prediction mode. Figure 2.7 shows how fast each type of filter converges to the predetermined set of weights of the unknown system.

2.6 Application (Waveform coding)

An application of adaptive filtering is the efficient encoding of analog signals(such as speech) in digital form. There are two basic ways in which this can be done. The first one is the *linear predictive coding(LPC)* approach in which the speech generation process is characterized by a simple model, and the input is used

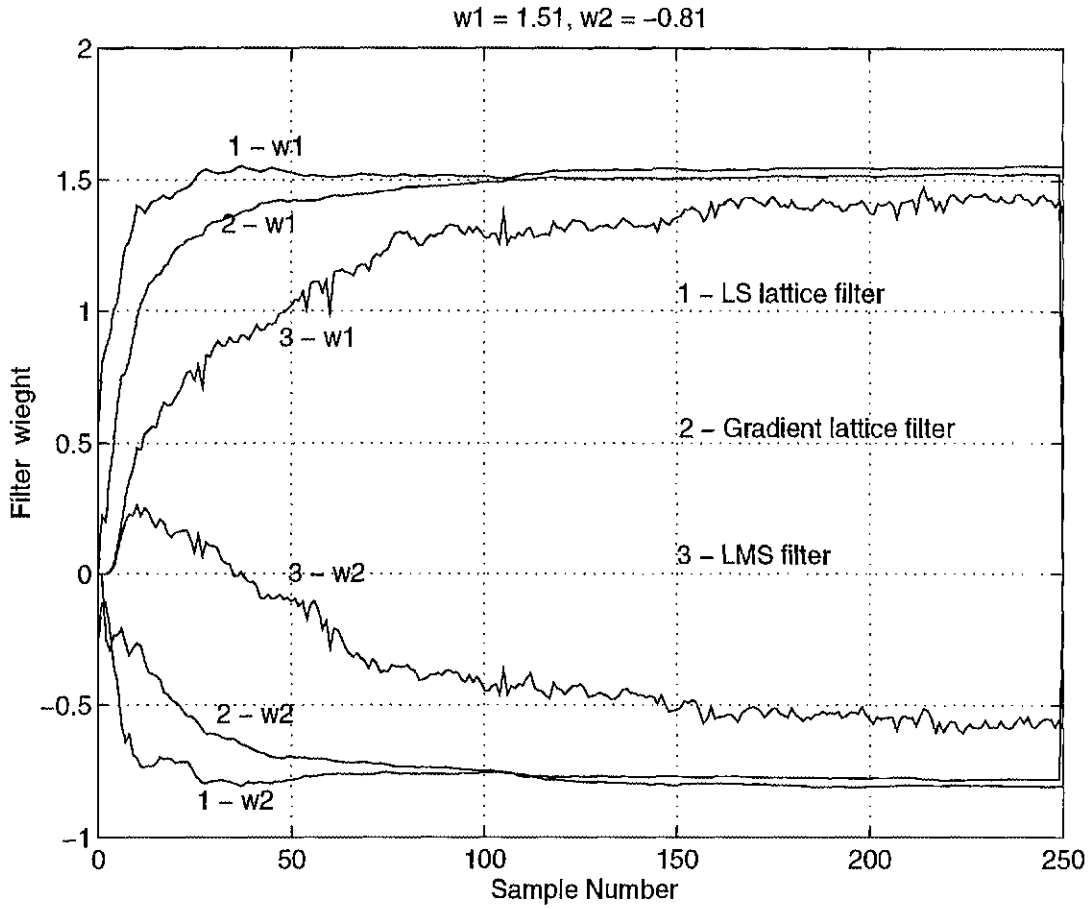

Figure 2.7 Performance of comparision of various adaptive filters (Order = 2).

to estimate the parameters of that model. The second one is called *waveform coding* [10]. In this approach the speech waveform is quantized directly. In the decoder the waveform is reconstructed from the quantized samples. Waveform coding is one of the most widely used forms of encoding, and it is also the one that is discussed here.

One of the approaches used to sample and quantize the input waveform directly is *pulse code modulation(PCM)* [10]. A slight modification of this technique gives us another very useful modulation technique called *differential pulse code modulation(DPCM)*. This technique uses a linear predictor in a feedback loop and quantizes the prediction error rather than the input signal. The quantized error is the signal that is sent to the receiver. If $y(n)$ is the input sample, and $\hat{y}(n)$ is the predicted value, then $e_f(n)$ is the prediction error. This error signal is obtained prior to quantization and it is this signal that is quantized in the encoder. At the receiver, the predicted value $\hat{y}(n)$ is added to the error signal to obtain the original signal. Since perfect reconstruction is not possible, we call this signal $Y(n)$. Therefore we have,

$$e_f(n) = y(n) - \hat{y}(n) \tag{2.49}$$

$$Y(n) = \hat{y}(n) + E_f(n) \tag{2.50}$$

where $E_f(n)$ is the quantized version of the prediction error available at the decoder.

If the prediction of $y(n)$ is good, then the prediction error $e_f(n)$is quite small compared to $y(n)$. Hence it is possible to encode the error signal using fewer bits, or increase the resolution by reducing the step size for a given sampling rate. The overall quantization error can be reduced by using a smaller step size. This is one of the major advantages offered by the adaptive digital pulse code modulation (ADPCM) encoder.

From equation (2.50) we see that the decoder needs the predicted value of the signal $y(n)$, i.e, $\hat{y}(n)$ to reconstruct the samples of $Y(n)$. But, $\hat{y}(n)$ is generated by forming a weighted sum of the past samples of $y(n)$. In this case the input samples are not available at the decoder. In order to overcome this problem $\hat{y}(n)$ is constructed from the past quantized samples of $Y(n)$ in both the encoder and the decoder. The equation describing the above is

$$\hat{y}(n) = \sum_{j=1}^{N} w_i Y(n-j) \tag{2.51}$$

The above principle is illustrated in figure 2.8. During the initial adaptation period, some arbitrary values are assumed for $Y(n)$ and as the predictor begins to adapt itself, the transmitter and receiver become synchronized.

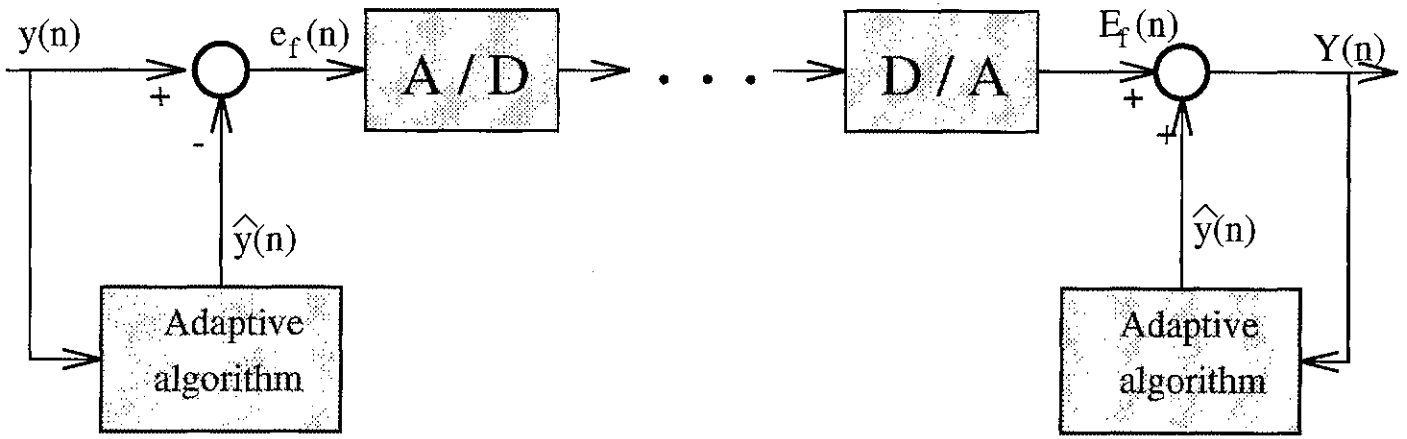

Figure 2.8. Adaptive differential pulse code modulation

CHAPTER 3

Fault Tolerance Concepts, Algorithm and Implementation

3.1 Fault Tolerance Concepts

Due to the tremendous advancements in technology in recent years, digital systems have shrunk drastically in size. These advance have made complex implementations such as special purpose DSP architectures, systolic arrays and highly pipelined systems increasingly attractive and realistically feasible. Unfortunately, with any increase in complexity comes an inevitable decrease in system system reliability. As a result, the discipline of *fault-tolerance* attracted a great deal of research interest over the past few years. A system is said to have failed if it no longer provides the services for which it was designed. A system is said to be fault tolerant if it continues to perform the desired functions correctly even in the presence of software or hardware faults.

The occurrence of faults in a system is assumed to be random, hence the occurrence of a fault in a system is a chance event. The parameters that are usually used to gauge to performance of a system and take into account the randomness of the occurrence of faults are reliability, availability, performability and maintainabil-

ity. **Reliability** R(t) of a system is defined as the probability that the system will perform correctly in a given interval of time$[t_o, t_1]$. **Availability** A(t)is defined as the probability that a system is operating correctly and will be available to perform a specified function at a given time t. **Performability** of a system is defined as an attribute of the system, by virtue of which the performance of the system is always at or above some level **K**. **Maintainability** of a system is defined as the ease with a fault can be located and diagnosed. Fault tolerance is a technique that is used to ensure that a system can fulfill the above requirements. There are many other parameters that may have to be considered in the evaluation of the performance of the system, but the above defined parameters are some of the most important ones that have to be considered in terms of fault tolerance.

There are three techniques which are commonly used to improve the performance of a system in the presence of faults. They are: fault avoidance, fault masking and fault tolerance. **Fault avoidance** is a technique wherein we attempt to prevent the occurrence of faults in a system by using highly reliable components, carrying out prescreening and testing of the components. **Fault masking** is a technique which prevents the faults from introducing errors into the system which will eventually cause system failure [11]. For example, if we consider a TMR system, the output from the faulty module is masked by the remaining two good modules. **Fault tolerance**, as defined in the beginning paragraph of this chapter, is the ability of the system to perform satisfactorily even after the occurrence of faults. Fault tolerance in general, encompasses both fault avoidance and fault masking techniques and adds some more attributes of its own. One of the commonly used approachs to achieve fault tolerance is the **reconfiguration** technique. In this thesis, the reconfiguration technique was used to achieve fault tolerance in pipelined filter structures.

This technique requires attention to issues such as fault detection, fault location and reconfiguration of the system by isolating the faulty module.

The technique chosen to improve the performance of a system depends purely on the application. For example, if we consider a spaceborne satellite, some of the primary concerns in the design of the satellite are its weight and power consumption. We would like to design a satellite that is as light as possible, and consumes as little power as possible. In this case, it may not be possible to use the fault masking technique because it dictates the use of lots of redundant hardware, which implies an increase in weight and power consumption. However, if we consider a typical earth-based military application any degradation in the performance of the system, even for a short period of time, is not tolerable because it could be destructive to human life and property. In this case redundant hardware might be justified in order to mask the faults from causing performance degradation.

Faults are of two types; permanent and transient. In this thesis we attempt to address both these types with respect to the pipelined adaptive filters. A fault-tolerant system is usually designed to tolerate a given class of faults. The given class indicates the relative level at which fault tolerance is being incorporated; the gate level, the transistor level or at a much higher level such as the module level. At the module level, the two most commonly considered systems are series and parallel systems. Because many systems can be modeled as some combination of these types, a little more insight into the reliability analysis of series and parallel systems is useful.

3.1.1 Series System

A series system may be defined as a system in which a failure in any one of the subsystems would cause a system failure. If we assume that the failure of each of the subsystems is independent and let R_i be the reliability of a subsystem i, then the overall system reliability is [11], [22], [5],

$$R_{ov} = \prod_{i=1}^{N} R_i \tag{3.1}$$

where N is the total number of subsystems. If we also assume that the failure rate of each of the subsystems is constant, then we have $R_i = e^{(-\lambda_i t)}$ and

$$R_{ov} = \prod_{i=1}^{N} e^{-\lambda_i t} \tag{3.2}$$

$$R_{ov} = e^{(\sum_{i=1}^{N} \lambda_i t)} \tag{3.3}$$

where λ_i is the failure rate of a subsystem and t is the time at which the failure rate was computed. If we further assume that all the subsystems have identical failure rates, then we have $\lambda_i = \lambda$ and $R_i = R$. Therefore, the overall reliability of a series system can be written as

$$R_{ov} = e^{-N\lambda t} \tag{3.4}$$

$$R_{ov} = R^N. \tag{3.5}$$

Equation (3.5) implies that the overall reliability of a series system decreases exponentially with respect to N. Therefore, for a series system to have high reliability, it is necessary that N be smaller or that the subsystems have very high reliability.

3.1.2 Parallel Systems

A parallel system, assuming all the subsystems are identical, in the context of fault tolerance and reliability, can be defined as a system which can fail only if

all of its subsystems have failed. Also, assuming that the subsystem failures are independent and R_i is the reliability of a subsystem i, the overall reliability of the system can be given by

$$R_{ov} = 1 - \prod_{i=1}^{N}(1 - R_i). \tag{3.6}$$

If it is further assumed that all the subsystems are identical and have a constant failure rate λ, then

$$R_{ov} = 1 - (1 - R)^N. \tag{3.7}$$

In general, most practical systems are made up of both series and parallel systems, whose reliabilities can be computed using equations (3.5) and (3.7).

In this thesis we deal primarily with the series system. This is because the two systems considered here, the DLMS pipelined structure and the lattice structure, are both series systems (figures 2.4 and 2.5) . In the following section, a flexible fault-tolerant adaptive filter structure is presented. The proposed structure is suitable for implementation of a variety of adaptive algorithms, including the delayed LMS, least squares lattice and gradient lattice filters. Fault tolerance is achieved by introducing redundancy into the system or the communication links, and employing a fault detection scheme which exploits the natural fault tolerance inherent in the adaptive filter.

3.2 Fault-Tolerant Algorithm and Implementation

In most of the work on fault-tolerance, the objective is to mask failures and if possible recover completely after the occurrence of a fault. In practice, fault-tolerance in a system is achieved by either replacing the faulty module with a stand-

by module or by reducing its functional capabilities. In some mission-critical systems, due to space and cost constraints, it is not always advisable to use redundancy to deal with the failures. In such cases, it might be desirable that the system be able to gracefully downgrade its performance. In this thesis both the methods are employed to achieve fault-tolerance for pipelined adaptive filter structures. Both these methods use the reconfiguration technique discussed in section I of this chapter.

Before addressing the issues of fault location, fault detection, and hardware reconfiguration, certain assumptions must be made. These assumptions are consistent with those made in the literature [11], [20], [29].

Assumption 1. No two faults can occur concurrently, i.e, no two faults occur at the same time.

Assumption 2. The switching logic is simple in comparison to the processing modules and hence reliable.

Assumption 3. Each processing module has the ability to perform a predetermined self-test sequence, but cannot be trusted to interpret the results.

Another issue that needs to be addressed before proceeding further is the concept of *scaling.* The function of the adaptive filter is to compute a finite set of weights based on the input samples. These computations can be performed by using dedicated hardware or by a general purpose computer/processor. In this context it is possible to develop dedicated hardware which computes one weight per module. Alternately, it is possible to partition the filter and map the computations associated with several weights onto a single module. For a programmable DSP module this can be done even more easily by modifying the relevant software.

A faulty processing module in a pipelined adaptive filter will cause a block of weights to take on unpredictable values. The number of weights affected depends on the order of the filter and the number of processors in the system. The particular block affected may vary, but the end result is always a divergence of the filter's error signal as shown in figure 3.1. In this figure the filter initially converges to an optimal solution by sample 200. But a fault occurs at sample 400 causing the filter to diverge. This provides the mechanism for fault detection. That is, it is possible to detect the occurrence of a fault (software or hardware) by monitoring the error signal. But there is one problem associated with this form of error detection. The convergence of an adaptive algorithm to an optimal solution is an iterative process. Since the input applied to an adaptive filter is random (though it is not completely white), the filter tries to track the input signal in the minimum mean square sense. When tracking the input in this manner the adaptive filter might sometimes over predict, which causes the error signal to have a higher value than its value for the previous sample. Using the error signal on a sample per sample basis for error detection might then cause the reconfiguration of the system even when there are no failures in the system. In order to avoid this situation, the error signal is *windowed.* By windowing, the error signal can be measured over **M** samples, where **M** is an integer. By adjusting the value of **M**, we can make sure that the error signal over **M** samples is always less than a predetermined threshold, when the system is in operating condition.

Fault location is a harder problem, and one which has received some attention outside the context of adaptive filtering [24], [12], [29]. The method proposed here for fault location is specific to the pipelined filter structure, and utilizes a processor self-test sequence. In this method, the error signal is continuously monitored by all

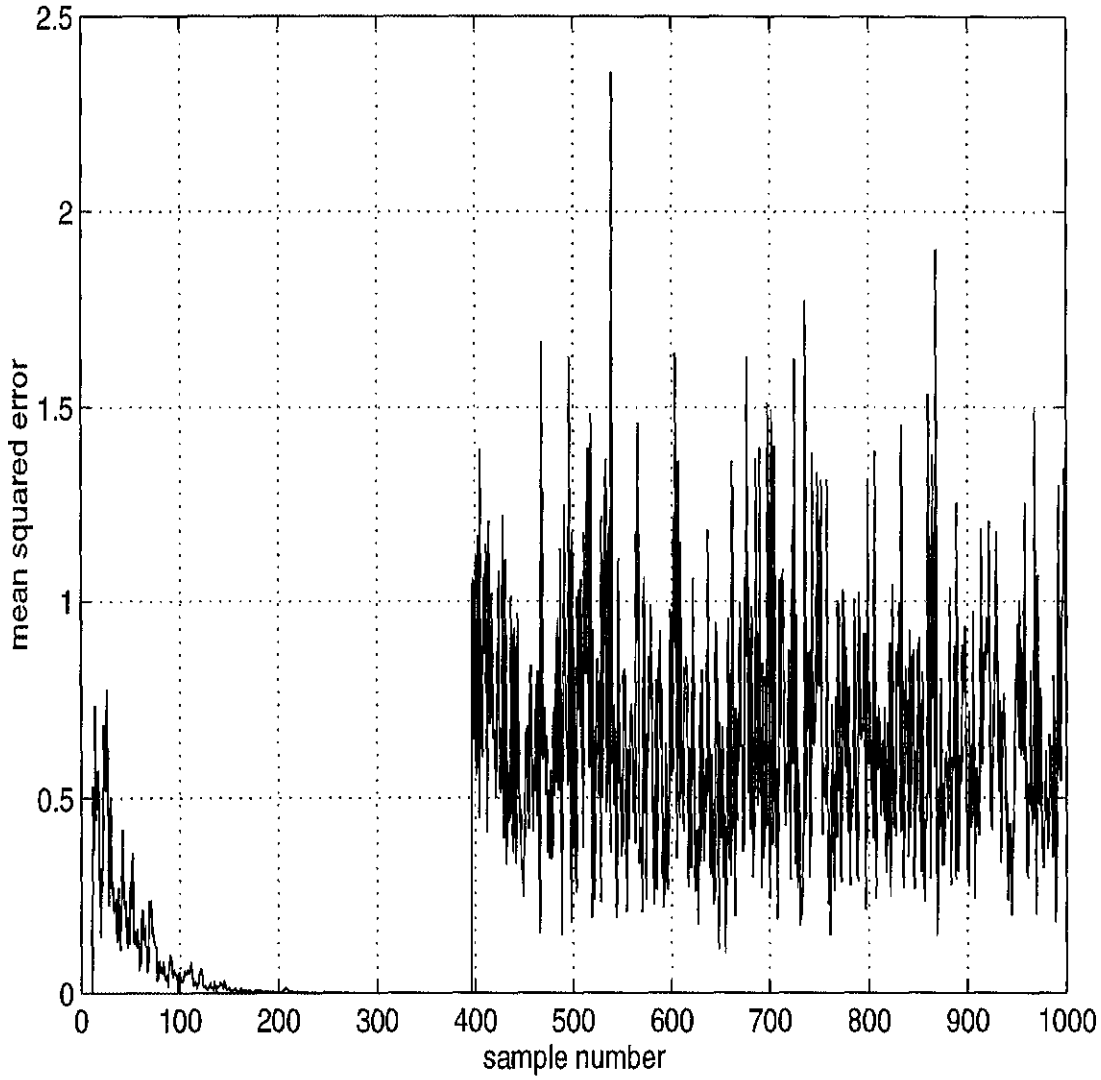

Figure 3.1 Divergence of a pipelined adaptive filter in the event of a fault.

the processing modules. If the error signal increases past a predetermined threshold for **M** consecutive samples, the processors go into diagnostic mode. Since a faulty processor cannot be trusted to evaluate its own integrity, test results are passed to the right and are evaluated by the subsequent processors. In figure 3.2, t_i is the evaluation of test results from PM_{i-1} which have been passed to PM_i. PM_i outputs $t_i = 1$ if it determines PM_{i-1} has failed its self test, and $t_i = 0$ if PM_i has completed its test successfully. The switching hardware dynamically reconfigures the pipeline, eliminating the faulty module. The switching hardware is discussed in more detail in the following subsection.

The algorithm for fault detection, fault location and reconfiguration is summarized below. The algorithm discussed here results in graceful performance degradation. An alternative technique, which involves the replacing of the faulty processor with a stand-by, is similar to the above mentioned one algorithmically. The major differences between the two are in the way they are implemented, which will be discussed in more detail in the next section. The algorithm is as follows:

(1) Proceed with filter computations of error and weight updates.

(2) Monitor the squared error $e^2(n)$ over **M** samples. If $e^2(n)$ exceeds a predetermined threshold for more than M consecutive samples periods, then a fault has occurred (either hardware or software), proceed to step (3), else proceed to step 1.

(3) Processors enter test mode, and pass test results to the right.

(4) Switch out the faulty processor by setting the input to each switch S_i to be $S_i = (t_i)(t_{i+1})'$.

(5) Return to step 1.

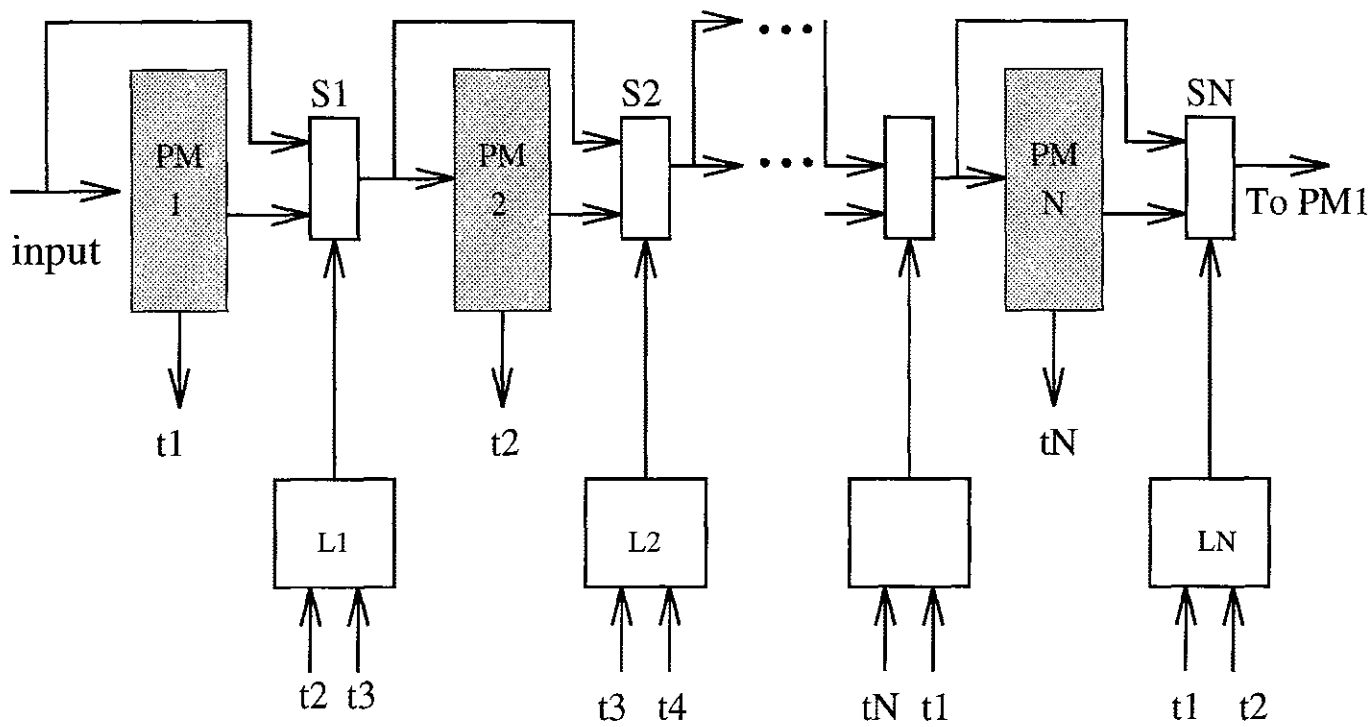

Figure 3.2 Fault tolerant pipelined adaptive filter.

In this way, no faulty processor may erroneously switch out a good one. That is, if processor 1 is faulty, then processor 3 must determine that processor 2 is fault-free before processor 2 is able to switch out processor 1. If no processor is determined to be faulty, the end result is a system restart. In this way, the divergence caused by numerical instability or any hardware transients in the system are handled within the proposed framework. The sampling rate is not reduced due to reconfiguration, however in the case of a processor fault, misadjustment will increase slightly due to a reduction in the number of filter taps being implemented.

3.3 Implementation

3.3.1 Fault tolerance by gracefully degrading the system performance

The implementation of this form of fault-tolerant pipelined adaptive filter is shown in figure 3.2. The boxes labeled **PM** are the processing modules, and the boxes labeled **S1, S2, ... , SN** are the switches that switch out the faulty module and route the data from the previous processor to the next one. For example, in figure 3.2 if PM_2 fails switch S2 will switch out PM_2 and will pass the data from $\mathbf{PM_1}$ to $\mathbf{PM_3}$. The switches considered here are an array of two input multiplexers, which are all controlled by a single control signal. The boxes labeled **L1, L2, ... , LN** are the latches that control the multiplexers based on the input they receive after the self-test evaluation. This is initialized to **1**. As shown in figure 3.3, every latch gets its input from a NAND gate which outputs a zero when the processor has to be switched.

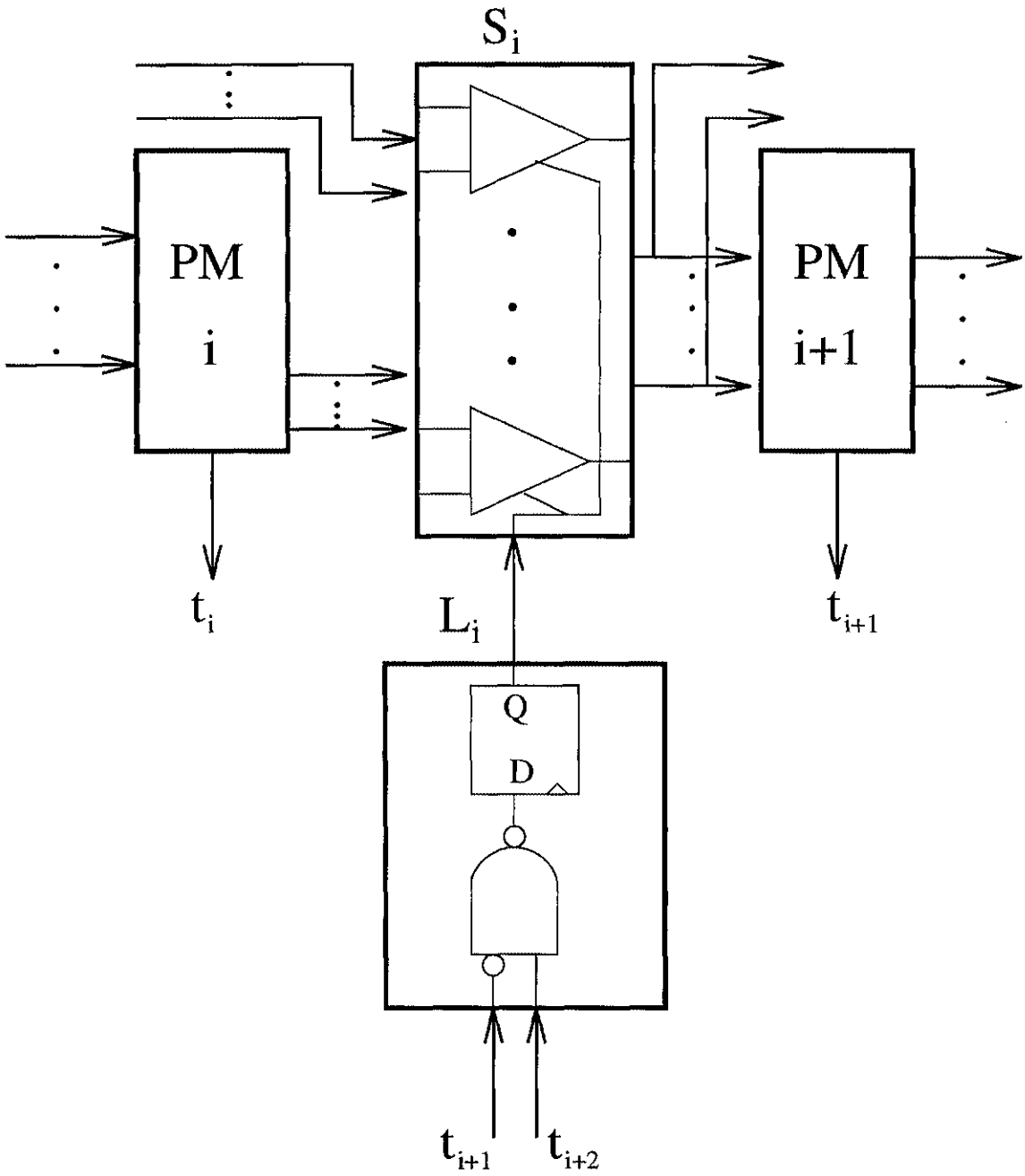

Figure 3.3 The switching hardware and the latch.

3.3.2 Fault-Tolerance using redundant processing modules

It is sometimes necessary to design a system that would function perfectly at all times. In such cases it is imperative that we use redundant processing modules which can be switched in when a fault occurs. This type of design for a pipelined adaptive filter is shown in figure 3.4. The switching circuit consists of an encoder, a multiplexer and a demultiplexer. The encoder inputs are the same signals that are passed to the switches $\mathbf{S}_1$, $\mathbf{S}_2$, ... etc. The output of the encoder acts as the control line for the multiplexers. The multiplexer selects the input from one of the processors to be passed on as input to the redundant processor. The same control signal is used to demultiplex the redundant processor output. This type of arrangement has the advantage of flexibility. The number of processors that a redundant processor can serve can be varied to provide the desired amount of fault tolerance.

For the case shown in figure 3.4, it is assumed that there is one redundant processor for every two processors in the system. The outputs of PM_1, PM_2 are passed to the multiplexers as well as to the processor in line via the switches $\mathbf{S}_1$ and $\mathbf{S}_2$. Based on the signals $\mathbf{L}_1$ and $\mathbf{L}_2$ the output of PM_1 and PM_2 are selected as input to the redundant processor. The output of the redundant processor R_1 is demultiplexed between the processors PM_2 and PM_3.

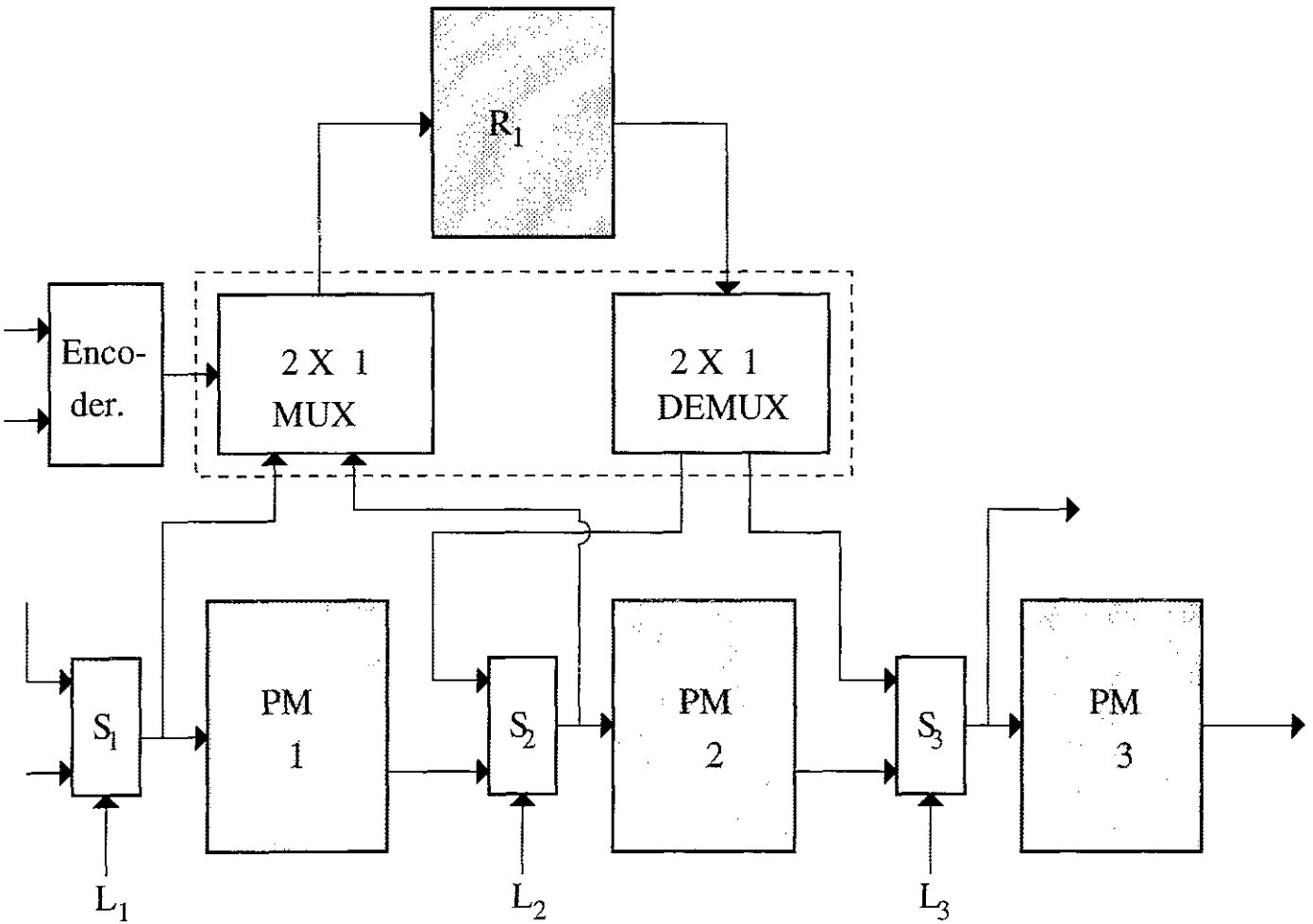

Figure 3.4 Fault-tolerant pipelined adaptive filter structure with redundant processors.

CHAPTER 4

Performance and Simulation Results

4.1 Introduction

As described in chapter II, the delayed LMS algorithm can be realized by introducing a fixed delay into the coefficient adaptation. Because of the introduction of delay, the number of samples required by the delayed LMS algorithm to converge to an optimal solution is greater than the number of samples required by the LMS algorithm. This behavior is illustrated in figures 4.1 and 4.2, which show the learning curves of these systems for $\mathbf{N} = 16$.

The inputs used in the simulations were random sequences generated using Matlab's random generator function. All the simulations presented in this thesis were written in Matlab. To obtain a colored input sequence, a random signal was passed through a fifth order lowpass Butterworth filter. Additional noise was included to obtain a SNR ratio of 20dB. The value of β used was .015. The predicted signal was subtracted from the filter output to obtain the error signal. This error signal was then used to update the filter coefficients. The curves (figures 4.1 and 4.2) were obtained by connecting filters in systems identification mode and averaging the results over 20 independent trials. The prediction error signal is plotted as

a function of the number of samples. The Matlab simulation code used to perform all the simulations given in this thesis are compiled in the appendix.

From figures 4.1 and 4.2 it appears that the LMS algorithm is superior to the DLMS algorithm in terms of convergence. This is because these graphs fail to convey any information regarding the rate at which the samples are taken in both the systems. When sampling rate is taken into consideration, it can be argued that the DLMS algorithm converges much faster than the LMS algorithm. This is because in DLMS, unlike LMS, it is not necessary to compute all weights before accepting a new input sample. Each processor in the pipeline updates a single weight or a small block of weights. The DLMS filter can be put to good use in applications which require high sampling rates. Another point worth noting about the pipelined DLMS filter is that, although the speed-up varies as a function of the filter order, the sampling rate is independent of the filter order. This is one of the most attractive features of a pipelined filter structure.

4.2 Increase in the minimum MSE with the decrease in filter order

To illustrate the proposed reconfiguration scheme, the delayed LMS algorithm, implemented as a pipelined filter, was selected for simulation. As mentioned earlier, a 16-tap filter is configured in system identification mode and mapped onto an array of four processing modules. A colored input sequence is applied to the system, and noise is included to give SNR of 20dB. Each processor is responsible for computing the weight updates and partial product updates for a block of four weights. The

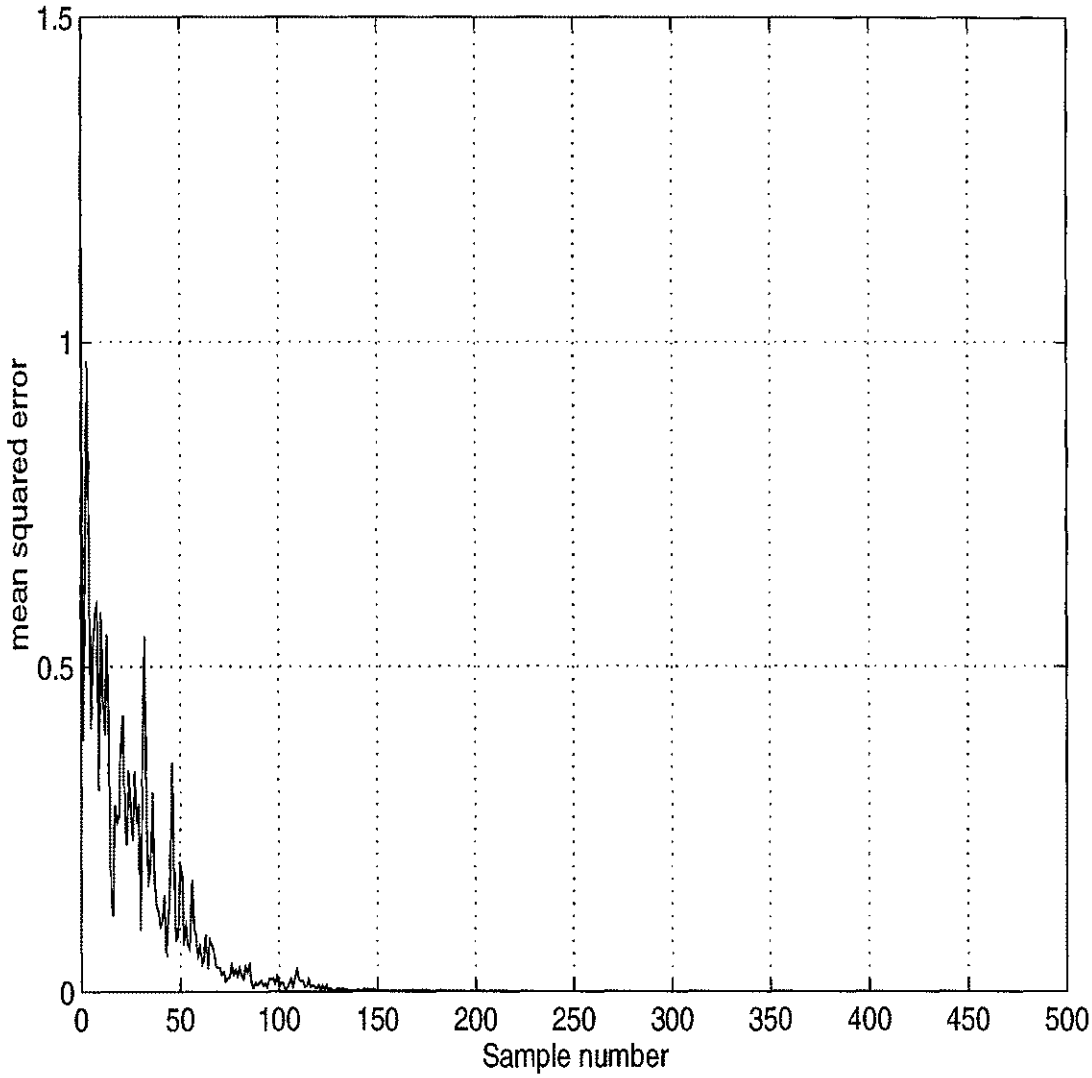

Figure 4.1 Simulation of LMS adaptive filter

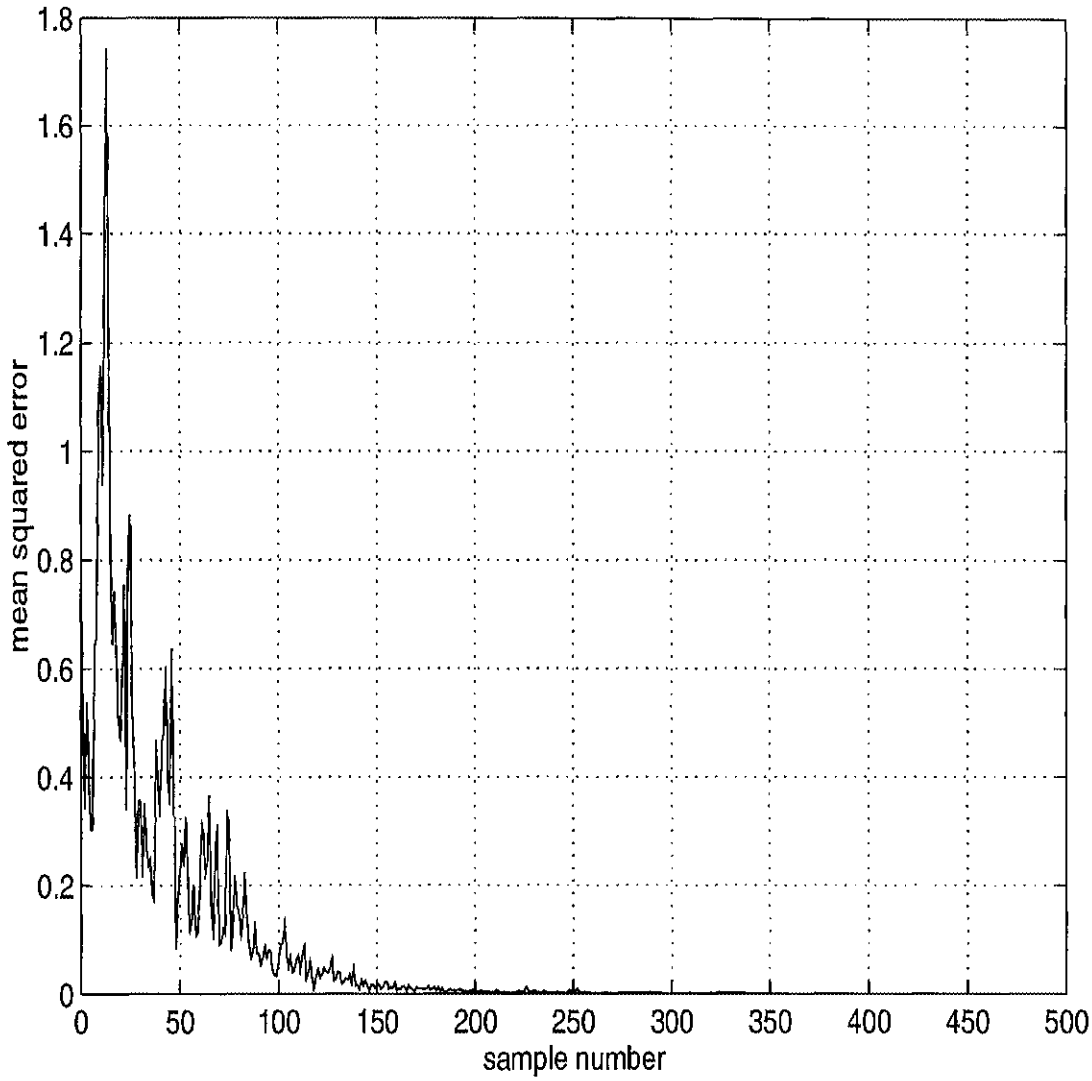

Figure 4.2 Simulation of DLMS adaptive filter

scalar updates to be performed for each weight by processor i at sample n, are shown below.

$$w_i(n-i) = w_i(n-i-1) + \beta e(n-N-i)x(n-N-2i) \tag{4.1}$$

$$y_i(n-i) = y_{i-1}(n-i) + x(n-2i)w_i(n-i-1). \tag{4.2}$$

Here x, y, e, w, and β have their usual meaning. Figure 4.3 shows the simulation of this system in response to a hardware fault. The stable region of convergence continues for 400 samples until a fault occurs in one of the processing modules at sample 400, causing the weights of that particular processor to take on random values. The result is error divergence. By sample 550 the system has reconfigured by switching out the faulty processor and the resulting 12-tap filter reconverges.

Since the reconfigured filter contains fewer weights than the original filter, the steady state misadjustment should be expected to increase. This behavior is illustrated in figure (4.4), which displays the same simulation shown in figure (4.3), where the axes have been scaled to focus on the region of convergence. In order to model this behavior mathematically, an analysis of the steady state error with respect to the order of the filter is required.

Before proceeding with the analysis of the steady state error with respect to the order of the filter, it is important to examine some important properties of eigenvectors which will be used in the derivation to follow [7].

Property 1: If $\lambda_1, \lambda_2, \ldots, \lambda_N$ denote the eigenvalues of the correlation matrix $\mathbf{R}$, then the eigenvalues of matrix $\mathbf{R}^k$ are $\lambda_1^k, \lambda_2^k, \ldots, \lambda_N^k$.

For any arbitrary correlation matrix $\mathbf{R}$, we have

$$\mathbf{Rq} = \lambda\mathbf{q} \tag{4.3}$$

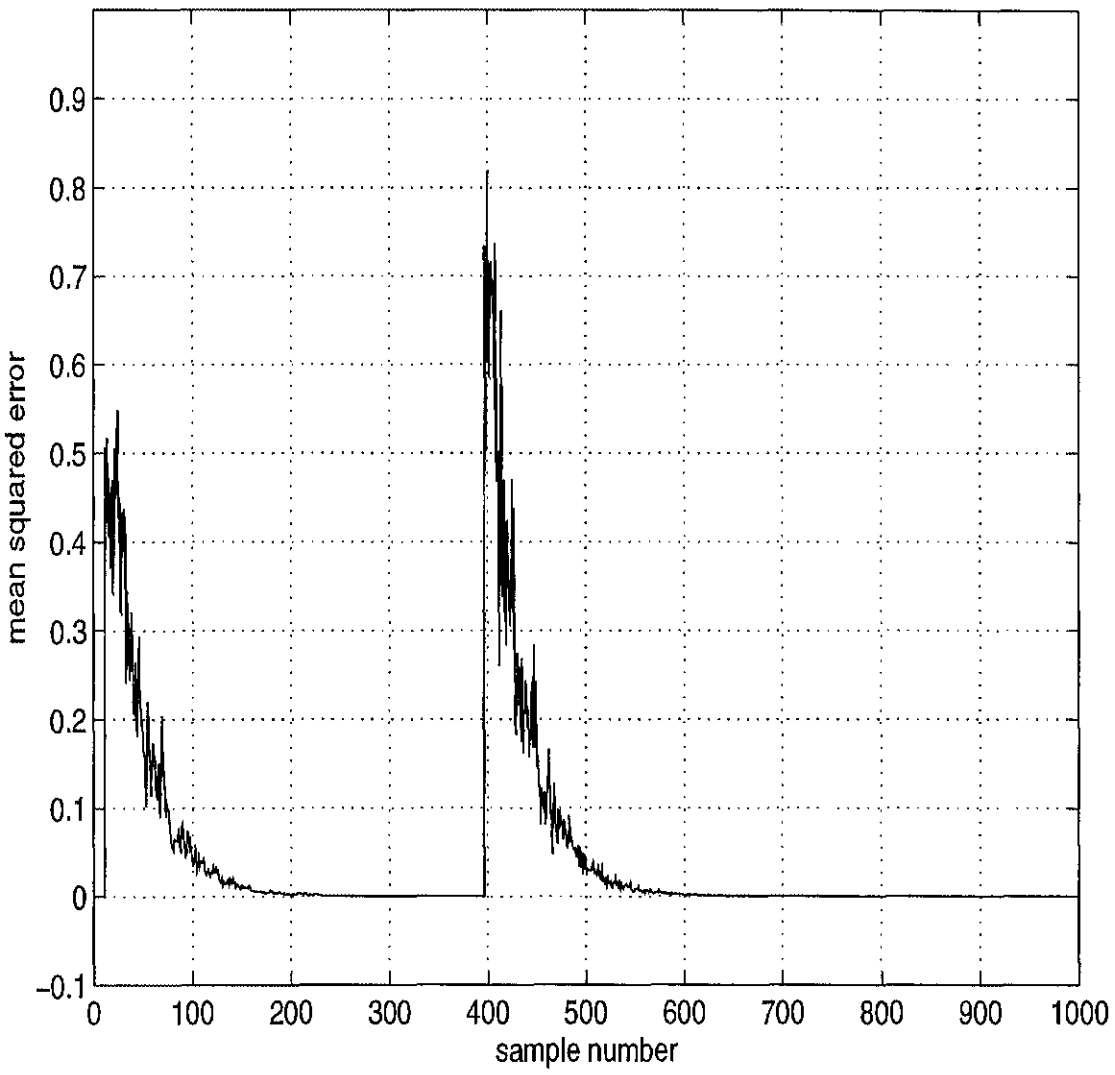

Figure 4.3 Simulation of a hardware fault and subsequent recovery

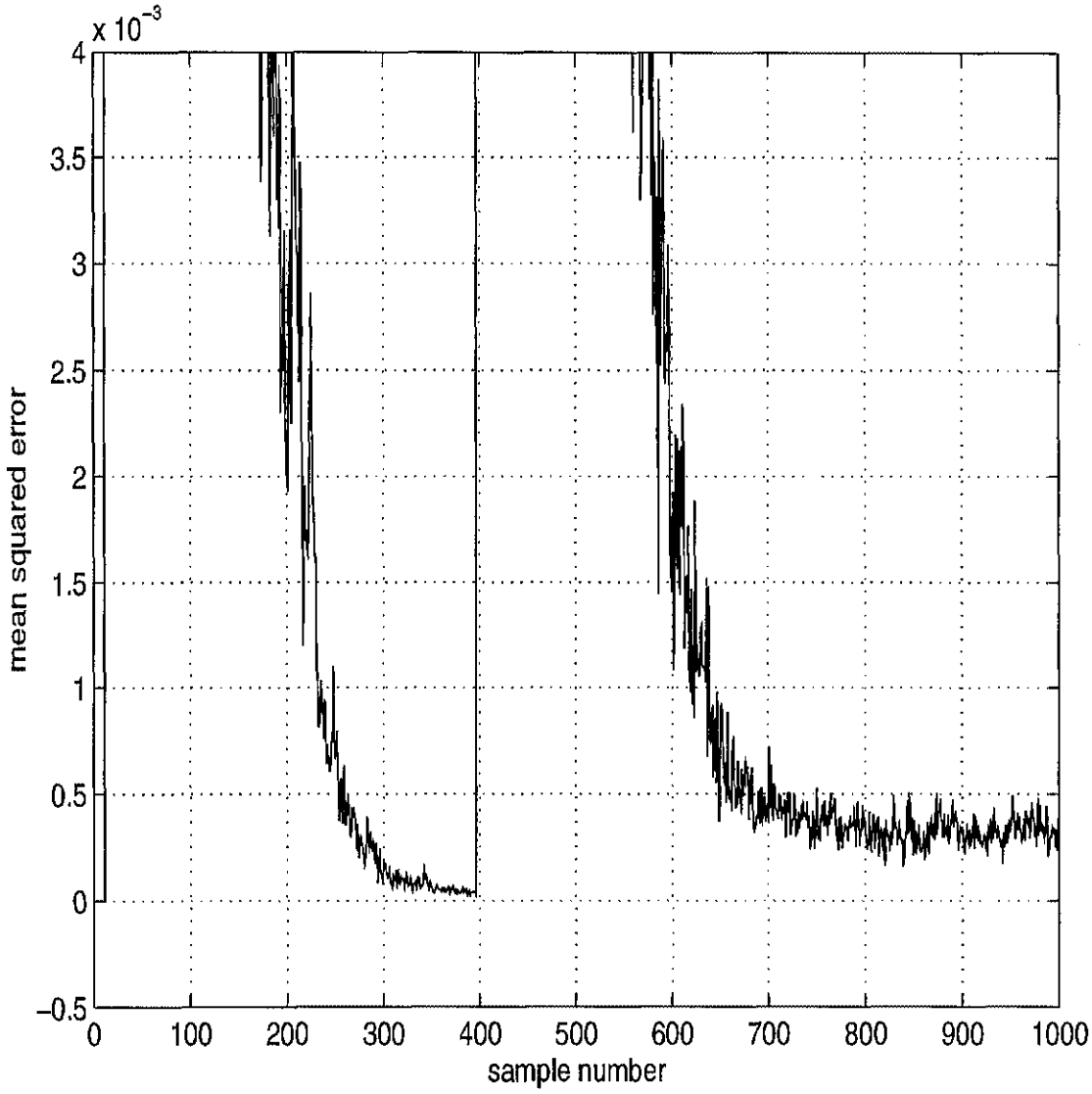

Figure 4.4 Illustration of increase in steady state error with the decrease in filter order

Repeated premultiplication of the above equation by $\mathbf{R}$ gives us

$$\mathbf{R}^k\mathbf{q} = \lambda^k\mathbf{q} \tag{4.4}$$

where q is the eigenvector associated with λ, thus proving the above stated property.

Property 2: Let $\mathbf{q}_1$, $\mathbf{q}_2$, ... , $\mathbf{q}_N$ be the eigenvectors corresponding to the distinct eigenvalues λ_1, λ_2, ... , λ_N of the N-by-N matrix of $\mathbf{R}$, respectively. If $\mathbf{Q}$ defines an N-by-N matrix

$$\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_N] \tag{4.5}$$

where $\mathbf{q}_i$ and $\mathbf{q}_j$ are orthogonal, that is,

$$\mathbf{q}_i^H\mathbf{q}_j = \begin{cases} 1, & \text{i} = \text{j} \\ 0, & \text{otherwise} \end{cases} \tag{4.6}$$

and if $\wedge$ defines an N-by-N matrix such that

$$\wedge = diag(\lambda_1, \lambda_2, ..., \lambda_N) \tag{4.7}$$

then the original matrix $\mathbf{R}$ may be diagonalized as follows

$$\mathbf{Q}^H\mathbf{R}\mathbf{Q} = \wedge. \tag{4.8}$$

this transformation is also called *unitary similarity transformation.*

Another important property of the correlation matrix $\mathbf{R}$ that will be useful in the derivation to follow is given below.

Property 3: The correlation matrix $\mathbf{R}$ of a discrete stochastic process is always non-negative definite and almost always positive definite, that is,

$$\mathbf{x}^H\mathbf{R}\mathbf{x} \geq 0 \tag{4.9}$$

where $\mathbf{x}$ is an arbitrary (non-zero) N-by-1 complex valued vector. Having defined the pproperties, it is easy to show that there is an increase in the minimum mean squared error with the decrease in the filter order.

The minimum mean squared error achievable for an LMS adaptive filter is given by

$$\mathbf{J}_{min} = \sigma_d^2 - \mathbf{p}^H \mathbf{w}_0. \tag{4.10}$$

The above equation, by making use of the normal equation (2.14) defined in chapter II, can also be written as

$$\mathbf{J}_{min} = \sigma_d^2 - \mathbf{p}^H \mathbf{R}^{-1} \mathbf{p}. \tag{4.11}$$

The unitary similarity transformation defined in equation(4.8) in property 2, can also be written as

$$\mathbf{R} = \mathbf{Q} \wedge \mathbf{Q}^{\mathbf{H}}. \tag{4.12}$$

Substituting for $\mathbf{Q}$ and $\wedge$ in the above equation gives

$$\mathbf{R} = \sum_{i=1}^{N} \lambda_i \mathbf{q}_i \mathbf{q}_i^H. \tag{4.13}$$

Applying *property 1*, to equation (4.13) gives

$$\mathbf{R}^{-1} = \sum_{i=1}^{N} \lambda_i^{-1} \mathbf{q}_i \mathbf{q}_i^H. \tag{4.14}$$

Substituting equation (4.14) into equation (4.11) for $\mathbf{R}^{-1}$ gives

$$\mathbf{J}_{min} = \sigma_d^2 - \mathbf{p}^H \sum_{i=1}^{N} \lambda_i^{-1} \mathbf{q}_i \mathbf{q}_i^H \mathbf{p}. \tag{4.15}$$

The above equation can be further written as

$$\mathbf{J}_{min} = \sigma_d^2 - \textstyle\sum_{i=0}^{N-1} \frac{\mathbf{p}^H \mathbf{q}_i \mathbf{p}^H \mathbf{q}_i^*}{\lambda_i}$$

$$= \sigma_d^2 - \sum_{i=1}^{N} \frac{| \mathbf{q}^H \mathbf{p} |^2}{\lambda_i}. \tag{4.16}$$

From *property 1*, the eigenvalues of $\mathbf{R}$ are known to be real and non-negative. Moreover for practical signals they are always positive. Hence the subtractor in equation (4.16) is always positive, which implies that $\mathbf{J}_{min}$ in equation (4.16) increases with the decrease in the order of the filter. This is illustrated in figure (4.5). In this simulation, the theoretical curve is the one obtained using equation (4.16). The other curve, the practical curve, was obtained by taking the minimum mean squared error values for various filter orders and computing the ensemble average over 50 independent trials.

4.3 Performance

A common measure of the fault tolerance of a system is reliability, which is defined as (1 - *probability of system failure*). The reliability of the N-element pipelined array shown in figure (2.4), as defined in equation (3.5) in chapter III, is given by

$$R_{pipe} = R^N. \tag{4.17}$$

R is the reliability of each individual processing module. The system shown in figure (3.2) is tolerant of single faults, and will therefore operate reliability at a given sampling rate in the presence of zero or one bad module. The overall reliability of the new system is given by

$$R_{ftpipe} = R^N + N(1-R)R^{N-1}. \tag{4.18}$$

The above equation was derived based on the permutations and combinations of elements arranged in a circular fashion. This form of analysis was required because

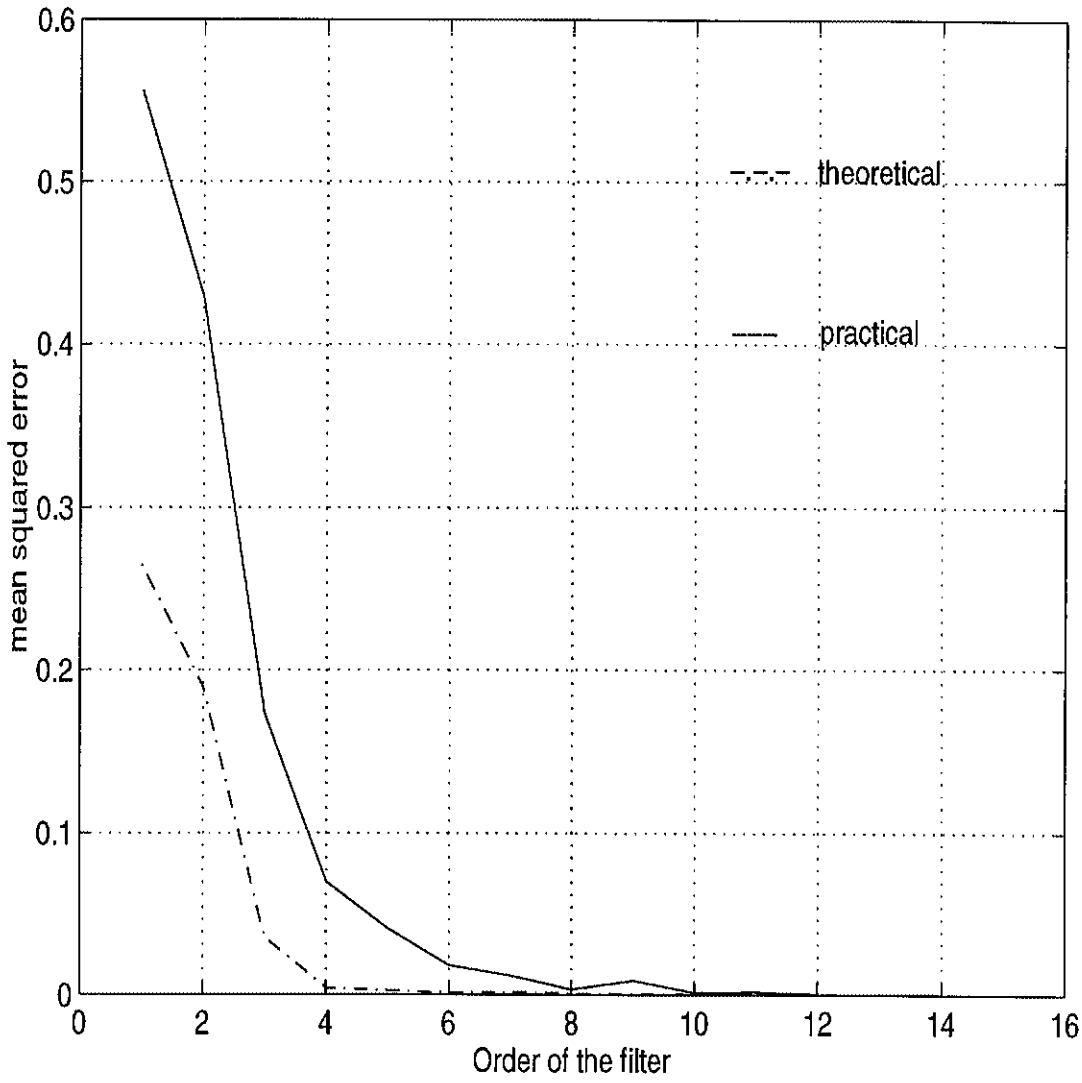

Figure 4.5 Misadjustment curve with respect to the order of the filter.

the modules in figure (3.2) are actually arranged in a circular fashion. The error signal of the last processing module is fed back into the first processing module, according to the pipelined adaptive filter structure defined in chapter II. However, it should be noted that equation 4.18 is valid only when N is greater than 3.

The improvement gained by the fault tolerant pipelined system is illustrated in figure (4.6), which shows reliability of a pipeline as a function of the pipeline length. A value of $R = 0.98$ has been assumed. It should be noted that this is actually a conservative estimate of the actual reliability, since certain type of multiple faults can in fact be tolerated, as long as no two faults occur within the same sample period and no two faults occur at consecutive processing modules. However, the extra component of reliability added by these cases is negligible for all but very large arrays, and is therefore ignored here. It is interesting to note that as the reliability of the individual subsystems is decreased, the overall reliability of the fault tolerant system approaches the overall reliability of the pipelined system. This is illustrated in figures 4.7 and 4.8. However, the proposed fault tolerant system always shows a better performance than the pipelined system. The difference between the curves is further increased when the extra component of reliability, discussed above, is taken into consideration for lengthy pipelines. From figures 4.7 and 4.8 it can be observed that the improvement in reliability of the system is very high for small arrays. This gives rise to the issue of scaling, implying that large arrays can be shortened into small arrarys by increasing the complexity of the processors to acheive the required performance.

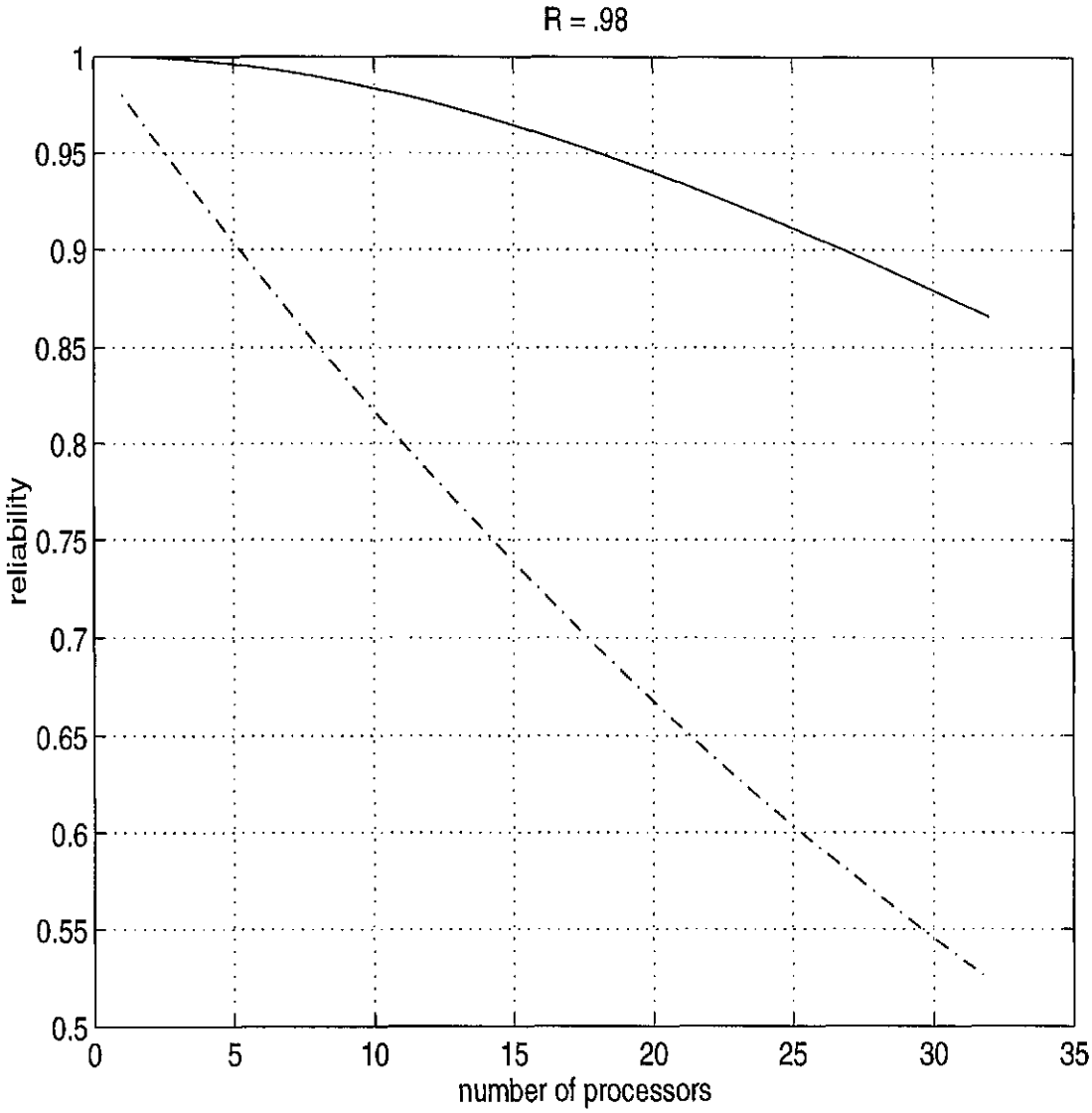

Figure 4.6 Reliability curves for pipelined systems with and without fault tolerance

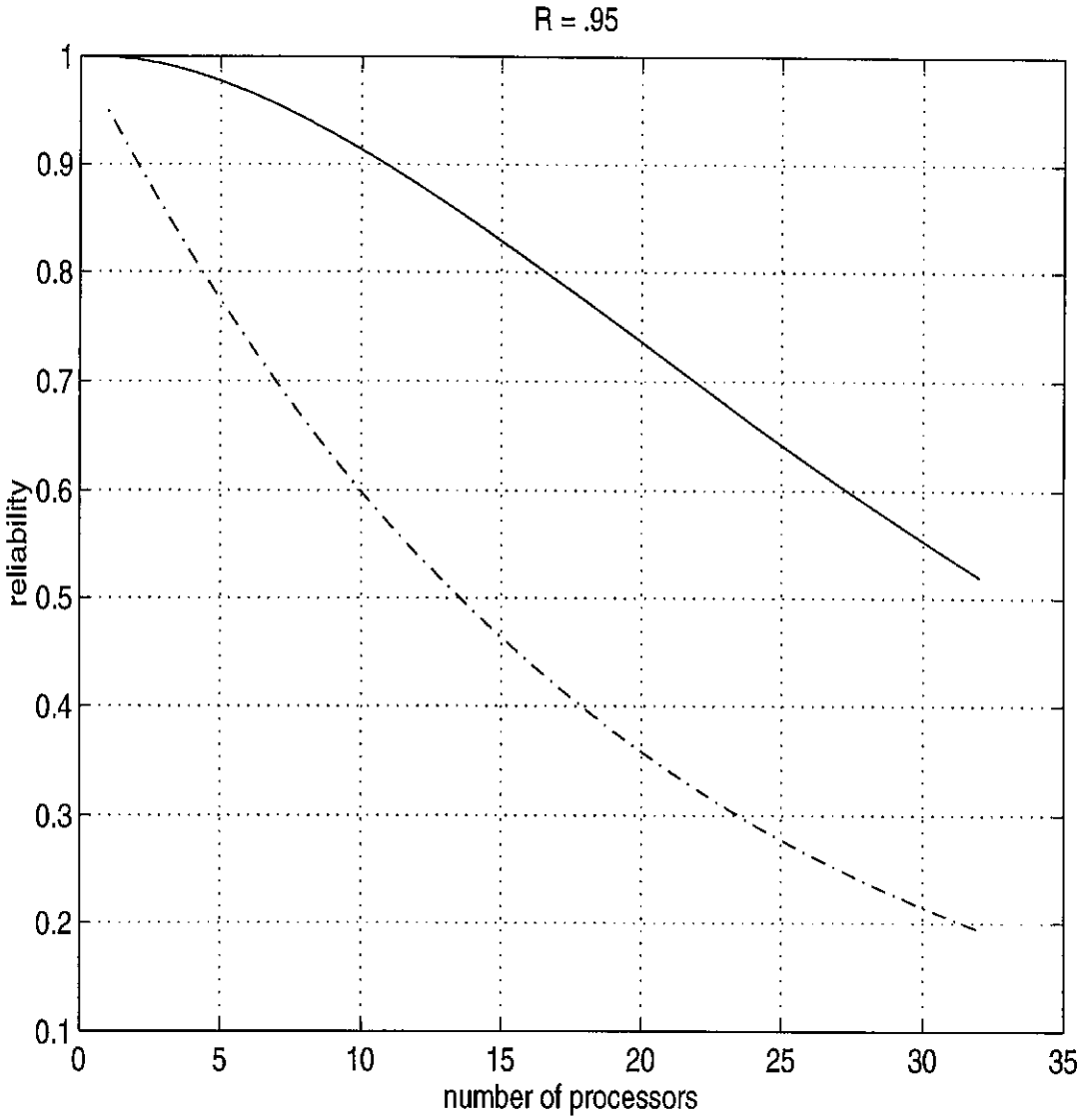

Figure 4.7 Reliability curves for pipelined systems with and without fault tolerance

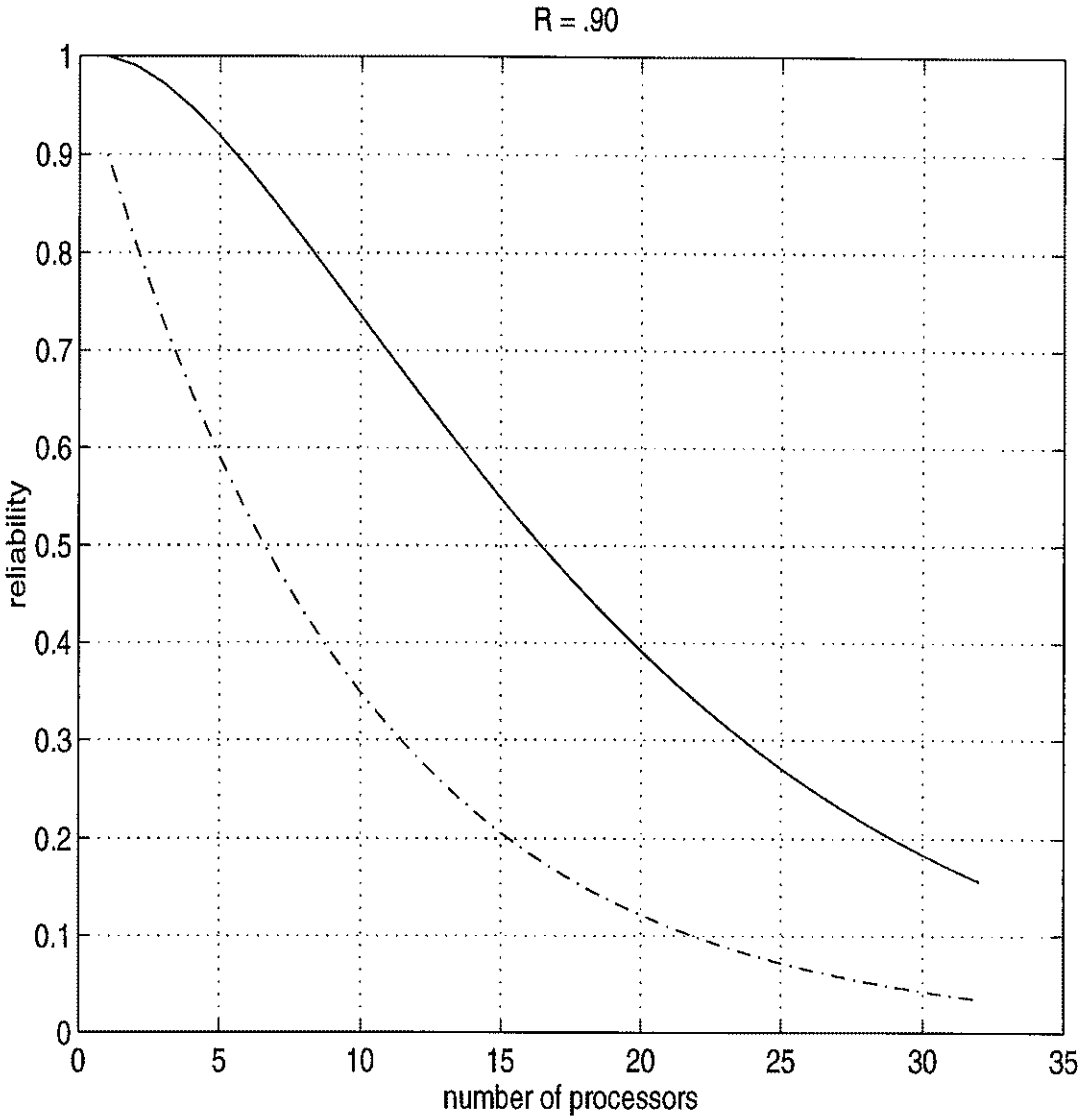

Figure 4.8 Reliability curves for pipelined systems with and without fault tolerance

CHAPTER 5

Conclusions

5.1 Summary

Pipelined systems are attractive because they support high data rates. They are also modular in their structure, which makes them an attractive candidate for VLSI implementations. The objective of this thesis has been to design and implement fault tolerant pipelined adaptive systems and also to evaluate the performance of such systems.

The fault tolerant algorithm was designed based on the assumption that there will be a self-test feature inherent to each processing module. The design assumed no specific architecture for the processing modules. However, it was assumed that each of the processing modules would possess the fault detection algorithm, implemented either in hardware or software. This allows the detection scheme to be distributed rather than being located at a single point in the pipeline. Such a distributed scheme is possible because the two systems considered, the delayed LMS and the lattice filter, have an error signal output that can be monitored for each weight.

The proposed algorithm takes advantage of the unique behavior of the pipelined adaptive filters in the event of a fault. This unique behavior was used to detect the

occurrence of faults in the system. In the diagnostic mode a predetermined self-test is performed on each processor and the results are passed to the right for evalutaion. The proposed system uses majority logic to switch out the faulty processor. In our simulation the length of the pipeline was pruned down in the event of a fault eliminating the faulty modules. The system was then allowed to reconfigure using the remaining modules.

The steady state error of the system was observed to increase when the length of the pipeline was reduced because of reconfiguration. A mathematical relationship that illustrates the increase in the steady state MSE error was derived. The theoretical and the practical results, illustrating the increase in steady state MSE with the decrease in the order of the filter, were in close agreement with each other. For over modeled systems, appreciable increase in the steady state MSE was not observed till the point at which they were close to being exactly modeled.

Reliability was another metric used to evaluate the performance of the system. An expression, to compute the reliability of the new system was derived. It was observed that there was an improvement of about 60% in the reliability of the pipelined adaptive system of length 32, and the reliability of individual subsystems was R = 0.98.

5.2 Future research

There are a number of enhancements that can be made to the work presented in this thesis. Some of them are listed below:

1. In the lattice filter, each stage of the lattice is an adaptive filter by itself. Each stage tries to minimize the FPE and BPE supplied to it by the previous stage. This implies that the forward prediction error of the pth stage is less than that

of *(p-1)*th stage. By taking advantage of this fact, a new detection technique unique to lattice filters can be developed. This technique would eliminate the need for self-testing by the processors. The testing can be achieved locally at each stage using a comparator. An attempt was made to prove the above argument mathematically, but no positive results have been obtained.

2. It would be useful to develop a testing scheme to test the processing modules, rather than using the self-testing feature of the modules. This is critical because, the speed of reconfiguration is directly proportional to the time it takes to perform the self-test on each of the processors.

3. One of the assumptions made in the design of the fault tolerant algorithm was that the switching logic was free of faults. However, the system could be made even more robust by introducing some fault tolerance into the switching logic.

4. The proposed system has a modular structure and which is highly suitable for VLSI implementation. The issues involved in such an implementation remain to investigated.

5. The fault tolerant algorithm presented in this thesis recovers from single faults and a class of multiple faults under some constraints. An important enhancement that could be made to the work presented in this thesis is to eliminate the above mention constraints. That is, make the algorithm more robust and able to correct multiple faults no matter where they occur in the pipeline.

Bibliography

[1] J. A. Abraham, et al., "Fault Tolerant Techniques for Systolic Arrays", *IEEE Computer*, pp. 65 - 75, July 1987.

[2] S. T. Alexander, "Adaptive Signal Processing, Theory and Applications", *Springer - verlag New York Inc.*, 1986.

[3] Bhasker R. Allam, Martin D. Meyer and James Leathrum, "A Fault Tolerant Adaptive Filter", *28th Asilomar Conference on Signals, Systems, and Computers*, November 1994.

[4] Leo Breiman, "Probability and Stochastic Processes", *The Scientific Press*, Palo Alto, 1986.

[5] Ernst G. Frankel, "Systems Reliability and Risk Analysis (2nd Edition)", *Kluwer Academic Publishers*, Boston, 1988.

[6] V. Grassi, L. Donatiello, and G. Iazeolla, "Performability Evaluation of Multicomponent Fault Tolerant Systems", *IEEE Transactions on Reliability*, Vol. 37, June 1988.

[7] Simon Haykin, "Adaptive Filter Theory", *Prentice-Hall*, New Jersey, 1986.

[8] C. W. Hong, Hon F. Li, and R. Jaya Kumar, "A Study of Two Approaches for Reconfiguring Fault Tolerant Systolic Arrays", *IEEE Transactions on Computers*, Vol. 38, June 1989.

[9] Micheal L. Honig, and David G. Messershmitt, "Adaptive Filters, Structures, Algorithms and Applications", *Kluwer Academic Publishers*, Boston, 1984.

[10] N. S. Jayant and Peter Noll, "Digital Coding of Waveforms, Principles and Applications to Speech and Video", *Prentice-Hall Inc.*, New Jersey, 1984.

[11] Barry W. Johnson, "Design and Analysis of Fault-Tolerant Digital Systems", *Addison-Wesley Publishing Company*, 1989.

[12] J. H. Kim, "On-line Detection of Errors in Homogeneous Multi-Processor Systems", *Proceedings, 1986 Real-Time Systems Symposium*, pp. 55-62, December 1986.

[13] Martin D. Meyer and D. P. Agrawal, "A High Sampling Rate Delayed LMS Adaptive Filter", *IEEE Transactions on Circuits and systems II*, Vol 40, No. 11, pp. 727-729, November 1993.

[14] Martin D. Meyer and D. P. Agrawal, "Modular Pipelined Implementation of a Delayed LMS Transversal Adaptive Filter", *Proceedings, 1990 IEEE Symposium on Circuits and Systems*, Vol. 3, pp. 1712-1715, May 1990.

[15] Martin D. Meyer and D. P. Agrawal, "Adaptive Lattice Filter Implementation on Pipelined Multiprocessor Architectures", *IEEE Transactions in Communications*, Vol. 69, No. 1, pp. 1234-1236, January 1990.

[16] T. K. Miller, S. T. Alexander, and L. J. Faber, "An SIMD Multi-Processor Ring Architecture for LMS Adaptive Algorithm", *IEEE Transactions on Communications*, Vol. 34, No. 1, Jan 1986.

[17] U. Minoni, G. Sansoni, N. Scarabottolo, "A Fault Tolerant Microcomputer Ring for Data Acquisition in Industrial Environments", *IEEE Transactions in Instrumentation and Measurement*, Vol. 38, No. 1, February, 1989.

[18] G. A. Clark, S. K. Mitra, and S. R. Parker, "Block Implementation of Adaptive Digital Filters",*IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol 29, pp. 744-752, June 1981.

[19] David G. Messerschmitt, "Arbitrarily High Sampling Rate Adaptive Filters", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol ASSP-35, pp. 455-470, April 1987.

[20] R. Negrini, M. G. Sami, and R. Stefanelli, "Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays", *The MIT Press*, 1989.

[21] Athanasios Papoulis, "Probability, Random Variables and Stochastic Processes (3rd Edition)", *McGraw-Hill Inc.*, 1991.

[22] Parag. K. Lala, "Fault Tolerance and Fault Testable Hardware Design", *Prentice-Hall International Inc.*, London, 1985.

[23] G. Long, F. Ling, and J. G. Proakis, "The LMS Algorithm with Delayed Coefficient Adaptation", *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 37, No. 9, September 1989.

[24] Arnold L. Rosenberg, "The Diogenes Approach to Testable Fault Tolerant Arrays of Processors", *IEEE Transactions on Computers*, Vol. C-32, October 1983.

[25] Samuel D. Sterns and Ruth A. David, "Signal Processing Algorithms", *Prentice-Hall Inc.*, New Jersey.

[26] Winfrid G. Schneewis, "Reliability Theory for Large Linear Systems with Helping Neighbours", *IEEE Transactions on Reliability*, Vol. 41, No. 3, September 1992.

[27] N. R. Shanbhag and K. K. Parhi, "A pipelined LMS Adaptive Filter Architecture", *IEEE Transactions on Signal Processing*, Vol. 41, No. 5, May 1993.

[28] Bernard Widrow, Samuel D. Sterns, "Adaptive Signal Processing", *Prentice-Hall Inc.*, New Jersey, 1985.

[29] J. H. Kim and Sudhakar M. Reddy, "On the Design of Fault Tolerant Two-Dimensional Systolic Arrays for Yield Enhancement", *IEEE Transactions on Computers*, Vol. 38, No. 4, April 1989.

[30] Nicholas Kalouptsidis and Sergios Theodoridis, "Adaptive Systems Identification and Signal Processing Algorithms", *Prentice-Hall International (UK) Limited*, 1993.

[31] Bernard Widrow, R. E. Kalman and N. DeCloris, "Adaptive Filters", *Aspects of Network and System Theory*, New York, pp. 657-687, 1971.

[32] Yuang-Ming Hsu and Earl E. Swartzlander, Jr., "FFT Arrays with Built-in Error Correction", it 28th Asilomar Conference on Signals, Systems and Computers, November 1994.

Appendix

SIMULATION CODE

I. Simulation of least squares lattice filter (curve #1 figure 2.7)

```
% Matlab simulation of least squares lattice.

alpha=0.015;                          % Step size parameter.

N=3;                                  % Order of the filter

a1tot=0;a2tot=0;                      % Initialize the weights.
clear E,clear T;

for reps=1:20                         % Begin the iterations
    reps                              % Display the iteration number.
    t=0:1:250+N;                      % length of the sequnce.

                                      % Initializing the variables.
    clear v,clear p;
    err_for = zeros(length(t),N);     % Forward prediction error.
    err_bac = zeros(length(t),N);     % Backward prediction error.
    ERR_for = ones(length(t),N) ;     % FPE residual.
    ERR_bac = ones(length(t),N);      % BPE residual.
    delta = zeros(length(t),N);       % Partial correlation coefficient.
    gama = ones(length(t),N);         % Anlge parameter.
    x = zeros(1,length(t));
    v = randn(size(t));

    for i = 1:N                       % For i <= N, x(i) = v(i).
        x(i) = v(i);
    end

                                      % Compute the input vector based
                                      % based on the given second order
                                      % equation. Have to start from N+1
                                      % because matlab doesn't allow
    for n = N+1:length(t)             % negative indexing.
        x(n) = (1.558*x(n-1)) - (0.81 * x(n-2)) + v(n);
    end

    for n = 2:1:length(t)             % LSL algorithm begins here.
                                      % Computations carried out in
```

```
        err_for(n,1) = x(n);        % first stage.
        err_bac(n,1) = x(n);
        ERR_for(n,1) = ERR_for(n-1,1) +(x(n)).^2;
        ERR_bac(n,1) = ERR_for(n-1,1) +(x(n)).^2;
        gama(n,1)    = 1;

        for m = 1:N-1               % Computatations of the rest
                                    % of the stages.

            delta(n,m+1) = delta(n-1,m+1) +
                           (err_bac(n-1,m)*err_for(n,m))/gama(n-1,m);

            err_for(n,m+1) = err_for(n,m) - (delta(n,m+1) *
                             err_bac(n-1,m))/ERR_bac(n-1,m);

            err_bac(n,m+1) = err_bac(n-1,m) -
                             (delta(n,m+1)*err_for(n,m))/ERR_for(n,m);

            ERR_for(n,m+1) = ERR_for(n,m) -
                             ((delta(n,m+1)).^2)/ERR_bac(n-1,m);

            ERR_bac(n,m+1) = ERR_bac(n-1,m) -
                             ((delta(n,m+1)).^2)/ERR_for(n,m);

            gama(n-1,m+1) = gama(n-1,m) -
                            (err_bac(n-1,m)).^2/ERR_bac(n-1,m);

        end
    end
                                % Computing the reflection
                                % coefficients.

    kf1 =  delta(N+1:length(t),2)./ERR_for(N+1:length(t),1);
    kb1 =  delta(N+1:length(t),2)./ERR_bac(N:length(t)-1,1);
    kb2 =  delta(N+1:length(t),3)./ERR_bac(N:length(t)-1,2);

                                % Computing the weight using
                                % reflection coefficients.

    a1tot = a1tot + kb1 - kf1.*kb2;
    a2tot = a2tot + kb2;
end
```

```
T = t(1:(length(t) -N));
a1avg = a1tot/reps;                    % Average the value of weights
a2avg = a2tot/reps;                    % over the number of iterations.

plot(T,a1avg,T,a2avg);grid             % Plot the graph.
```

II. Simulation of the lattice filter (curve #2 of figure 2.7)

```
% Matlab simulation for lattice filter for N = 2.

N=2;                                % Order of the filter.

a1tot=0;a2tot=0;                    % Initializing the weights to zero.
clear E,clear T;
randn('seed', 1);

for reps=1:20
    reps
    t=0:1:250+N;

                                    % Initializing the variables.
    clear v, clear p;
    err_for = zeros(length(t),N);   % Forward error vector.
    err_bac = zeros(length(t),N);   % backward error vector.
    K = zeros(length(t),N);
    D = ones(length(t),N);
    randn(size('normal'));
    x = zeros(1,length(t));         % Initialize input vector.
    v = zeros(1,length(t));         % Initialize random vector.
    ref_coeff = zeros(length(t),N); % Initialze reflection
                                    % coefficients.
    v = randn(size(t));

                                    % Compute the input vector based
                                    % on the second order equation.
    for n = N+1:length(t)
        x(n) = (1.558*x(n-1)) - (0.81 * x(n-2)) + v(n);
    end

    for n = N+1:1:length(t)-2       % Lattice algorithm.

                                    % Computations carried out in
                                    % the first stage.

        K(n+1,1) = K(n,1) + x(n+1)*x(n) ;
        D(n+1,1) = D(n,1) + x(n+1)^2 + x(n)^2;
        ref_coeff(n+1,1) = (2*K(n+1,1))/(D(n+1,1));
```

```
        err_for(n+1,1) = x(n+1) - ref_coeff(n,1)*x(n);
        err_bac(n+1,1) = x(n) - ref_coeff(n,1)*x(n+1);

        for p = 2:N                     % Computations of the rest of
                                        % the stages.

            K(n+1,p) = K(n,p) + err_for(n+1,p-1)*err_bac(n,p-1);
            D(n+1,p) = D(n,p) + (err_for(n+1,p-1))^2 +
                                  (err_bac(n,p-1))^2;

            ref_coeff(n+1,p) = (2*K(n+1,p))/D(n+1,p);

            err_for(n+1,p) = err_for(n+1,p-1) -
                             ref_coeff(n,p) * err_bac(n,p-1);
            err_bac(n+1,p) = err_bac(n,p-1) -
                             ref_coeff(n,p) * err_for(n+1,p-1);

        end
    end
                                    % Computing the weights from
                                    % reflection coefficients.

    a1tot = a1tot + ref_coeff(N+1:length(t),1).*
                    (1 - ref_coeff(N+1:length(t),2));

    a2tot = a2tot + ref_coeff(N+1:length(t),2);

end

T = 0:1:(length(t) -N-1);

a1avg = a1tot/reps;                 % Average the value of weights
a2avg = a2tot/reps;                 % over a number of iterations.

plot(T, a1avg, T, a2avg);           % Plot the graph.
```

III. Simulation of LMS filter (Order = 2) (curve #3 figure 2.7)

```
% Matlab simulation program for LMS algorithm for N = 2.

alpha=0.015;              % Step size parameter.

                          % Filter coefficients of the
                          % system to be modeled.

b=[.0528 .2639 .5279 .5279 .2639 .0528];
a=[1 0 .6334 0 .0557 0];
bb = [.3842 .8704 .3842];
aa = [1 0 0];

N=2;                      % Filter order.
etotal=0;                 % Initializing the error and
eaverage=0;               % average error vectors.
w1 = 0; w2 = 0;           % Initializing the weights.
clear E,clear T;

for reps=1:10             % Begin the iterations.
    reps                  % Display iteration number.
    t=0:1:250+N;          % Length of the sequence.

                          % Initialize the variables.
    clear e;
    clear d;
    clear x;
    clear y;
    clear noise;
    W = zeros(length(t),N);         % Initialize weight vector.
    X = zeros(1,N);                 % Initialize input vector.
    x = zeros(1,length(t));         % Initialize input sequnce vector.
    v = randn(size(t));             % Initialize the random vector.
    noise = .0015*randn(size(t));   % Generate the noise signal.

    for i = 1:N                     % For i <= N, x(i) = v(i).
        x(i) = v(i);
    end
```

```
                                % Compute the input vector based
                                % based on the given second order
                                % equation. Starts from N+1 because
                                % matlab doesn't allow negative
    for n = N+1:length(t)       % indexing.
        x(n) = (1.558*x(n-1)) - (0.81 * x(n-2)) + v(n);
    end

    d = (filter(b,a,x) + noise);   % Generate the desired signal.

    for n = N+1:length(t)          % LMS algorithm.

        X = x(n:-1:(n-N+1));
        y(n) = X*W(n,:)';
        e(n) = d(n) - y(n);
        W((n+1),:) = W(n,:) +(alpha*(e(n)*X));
    end

    w1 = w1 + W(:,1)';             % Updating an entry in
    w2 = w2 + W(:,2)';             % the weight vector.
    etotal = etotal + e.*e;        % Computing the mean squared error.
end

eaverage = etotal/reps;            % Average the MSE.
w1 = w1/reps;                      % Average the weight vectors.
w2 = w2/reps;

T = t(1:length(t)-N);
E = eaverage(N+1:length(t));
                                % Plot w1 and w2 w. r. t to T.

plot(T,w1(1:length(T)), T,w2(1:length(T)));
xlabel(' Sample Number')
ylabel(' Filter Value (w1 = 1.51, w2 = -0.81)');
```

IV. Simulation code for the LMS filter (figure 4.1)

```
% Matlab simulation program for LMS algorithm.

alpha=0.015;                          % Step size parameter.

                                      % Filter coefficients of the
                                      % system to be modeled.

b=[.0528 .2639 .5279 .5279 .2639 .0528];
a=[1 0 .6334 0 .0557 0];
bb = [.3842 .8704 .3842];
aa = [1 0 0];

N=16;                                 % Order of the filter
etotal=0;                             % Initializing the error and
eaverage=0;                           % average error vectors.
clear E,clear T;

for reps=1:20                         % Begin the iterations.
    reps                              % Display the iteration number.
    t=0:1:500+N;                      % length of the sequnce.

                                      % Initialize the variables.
    clear e;
    clear d;
    clear x;
    clear z;
    clear y;
    clear noise;
    W = zeros(length(t),N);           % Initializing weight vector.
    X = zeros(1,N);                   % Initializing input vector.
    randn(size('normal'));
    z=randn(size(t));
    noise = .0015*randn(size(t));     % Generating the noise vector.
    x = filter(bb,aa,z);

    d = (filter(b,a,x) + noise);      % Generating the desired signal.

    for n = N+1:length(t)             % LMS algorithm.
```

```
        X = x(n:-1:(n-N+1));
        y(n) = X*W(n,:)';
        e(n) = d(n) - y(n);
        W((n+1),:) = W(n,:) +(alpha*(e(n)*X));
    end

    etotal = etotal + e.*e;          % Compute the mean squared error.
end

eaverage = etotal/reps;              % Average the mean squared error.
T = t(1:length(t)-N);
E = eaverage(N+1:length(t));
plot(T,E,'r-');grid                  % Plot the graph
                                     % 'error Vs sample number'

xlabel(' Sample number ');
ylabel(' mean squared error ');
```

V. Simulation code for the delayed LMS algorithm (figure 4.2)

```
% Matlab simulation program for delayed LMS algortihm.

alpha=0.010;                    % Stepsize parameter.
                                % Filter coefficients for the
                                % filter to be modeled.

b=[.0528 .2639 .5279 .5279 .2639 .0528];
a=[1 0 .6334 0 .0557 0];
bb = [.3842 .8704 .3842];
aa = [1 0 0];

order = 16;                     % Order of the filter
N = order;
D = order;

etotal = 0;                     % Initializing the error and
eaverage = 0;                   % average error vectors.
clear E,clear T;

for reps=1:20                   % Begin the iterations.
    reps                        % Display the iteration number.
    t=0:1:500+D;                % Length of the sequence.

                                % Initialize the variables

    e = zeros(1,length(t));
    d = zeros(1,length(t));
    x = zeros(1,length(t));
    y = zeros(1,length(t));

    W = zeros(1,N);                     % Initializing weight vector.
    X = zeros(length(t),N);             % Initializing input vector.
    randn(size('normal'));
    z=randn(size(t));
    noise = .0015*randn(size(t));       % Generating the noise vector.
    x = filter(bb,aa,z);

    d = (filter(b,a,x) + noise);        % Generating the desired signal.
```

```
                                        % Begin processing input.
    for n = (D+1) :length(t)

        X(n,:) = x(n:-1:(n-N+1));
        y(n) = X(n,:)*W';
        e(n) = d(n) - y(n);
        W = W + ( alpha*e(n-D)*X(n-D,:));   % DLMS updates.

    end

    etotal = etotal + e.*e;             % Compute the mean squared error.
end

eaverage = etotal/reps;                 % Average the mean squared error.
T = 0:1:(length(t) -D-1);
E = eaverage(D+1:length(e));
plot(T,E);grid                          % Plot the graph for
                                        % MSE Vs # of samples.
%semilogy(T,E,'r-');grid
xlabel(' sample number ');
ylabel(' mean squared error ');
```

VI. Simulation of the occurrence of a fault in a pipeline system (figures 4.3 and 4.4)

```
% Matlab simulation illustrating the occurrance of a fault in a
% pipelined system.

alpha=0.015;                          % Step size parameter.
                                      % Coefficients of filter
                                      % to be modeled.

b=[.0528 .2639 .5279 .5279 .2639 .0528];
a=[1 0 .6334 0 .0557 0];

N=16;                                 % Order of the filter.
D=4;                                  % Delay

etotal=0;                             % Error vector and
eaverage=0;                           % average error vector.
clear E,clear T;

for reps=1:50                         % Begin the iterations.
    reps
    t=0:1:1000+D;
                                      % Initialize the variables
    e = zeros(1,length(t));           % Initialize Error sequence.
    d = zeros(1,length(t));           % Initialize Desired signal
    x = zeros(1,length(t));           % Initialize input vector.
    y = zeros(1,length(t));
    W = zeros(1,N);                   % Initializing weight vector.
    X = zeros(length(t),N);           % Input vector.
    randn(size('normal'));
    x = randn(size(t));               % Generating input vector.
    d = filter(b,a,x);                % Generating desired signal.

    for n = (N+1) :400                % DLMS algorithm.

        X(n,:) = x(n:-1:(n-N+1));
        y(n) = X(n,:)*W';
        e(n) = d(n) - y(n);
        W = W + ( alpha*e(n-D)*X(n-D,:));
```

```
    end

    while (e(n)*e(n))<1              % Allow the fault to occur
                                     % at sample 400.

        n = n+1;
        X(n,:) = x(n:-1:(n-N+1));
        y(n) = X(n,:)*W';
        e(n) = d(n) - y(n);
        W = W + (alpha*(e(n-D)*X(n-D,:)));
        W(1:1:4) = rand(size(1:1:4));

    end

    D=D-1;
    for n=(n+1):length(t)            % Allow the filter to
                                     % reconverge using 12 weights.

        X(n,:) = x(n:-1:(n-N+1));
        y(n) = X(n,:)*W';
        e(n) = d(n) - y(n);
        W = W + (alpha*(e(n-D)*X(n-D,:)));

        W((N-3):1:N) = [0 0 0 0];    % Last four weight allowed to
                                     % take zero values. Since they
                                     % are out of the system.

    end

    D=D+1;
    etotal = etotal + e.*e;          % Computing mean squared error.
end

eaverage = etotal/reps;              % Ensemble of mean squared error.

T = 0:1:(length(t) -D-1);
E = eaverage(D+1:length(e));

plot(T,E,'r-');grid                  % Plot MSE Vs sample number.
%axis([0 1000 -0.1 1.0]);
ylabel('mean squared error')
xlabel('sample number')
```

VII. Simulation of the theoretical curve (figure 4.5)

```
% Matlab simulation for theoretical proof of increase in minimum MSE
% as the filter order decreases.

alpha=0.015 ;              % Step size parameter.

                           % Filter coefficients of the
                           % system to be modeled.

b = [.0528 .2639 .5279 .5279 .2639 .0528];
a = [1 0 .6334 0 .0557 0];

len = 1000;                % Sequence length.
K = 16;                    % Order of the filter.
D = 16;
clear Z;
J = zeros(1,K);            % Minimum mean squared error vector.

for N = K:-1:1             % Begin computaions for an order K.
    N
    Var = 0;
    R1 = zeros(N,N);
    randn('seed',1);

    for reps=1:10
        reps
        t=0:1:len+D;
        var = 0;
        d = zeros(1,length(t));
        x = zeros(1,length(t));
        y = zeros(1,length(t));
        R = zeros(N,N);            % Auto-correlation vector.
        P = zeros(N,1);            % Cross-correlation vector.
        Q = zeros(N,N);            % Matrix of eigenvectors.
        E = zeros(N,N);            % Diagonal matrix. Diagonal
                                   % elements being the eigenvalues.
        L = zeros(1,N);            % Vector of eigenvalues.
        X = zeros(length(t),N);
        randn(size('normal'));
```

```
        x = randn(size(t));         % Genrating input signal.
        d = filter(b,a,x);          % Generating desired signal.

        dsum = 0;                   % Sum of the desired signal
        dsqsum = 0;                 % squared sum of desired signal.

        for n = (N+1):length(t)     % begin computations.

            X(n,:) = x(n:-1:(n-N+1));
            dsum = dsum + d(n);             % Sum of desired sig.
            dsqsum = dsqsum + d(n)*d(n); % Squared sum of d(n).
            R = R + X(n,:)'*X(n,:);         % Correlation matrix.
            P = P + d(n)*X(n,:)';           % Cross-corr. matrix.

        end

        R = R/len;          % Averaging the correlation matrix.
        P = P/len;          % Averaging the cross-corr. matrix.

                            % Computing variance for a sequence.
        var = dsqsum/len - (dsum/len)^2

        R1 = R1 + R;        % Ensembling correlation matrix.
        Var = Var + var;    % Esembling variance.
    end

    R1 = R1/reps;           % Computing the ensemble of
    Var = Var/reps;         % correlation matrix and variance.
    [Q,E] = eig(R1);        % Obtain the eigenvalues and
                            % eigenvectors of correlation matrix.
    L = diag(E);            % Extract the eigenvalues.

    sum = 0;
    for i = 1:1:N
        sum = sum + ((Q(:,i)'*P)^2)/L(i);
    end

    J(N) = Var - sum;
end

J = J.*J;                   % Jmin squared error.
Z = 1:K;
plot(Z,J,'r-.');grid        % plot the graph. }
```

VIII. Simulation of the practical curve (figure 4.5)

```
% Matlab simulation to practically show that the minimum MSE
% increases as the filter order is decreased.

alpha=0.015;                            % Step size parameter.

                                        % Coefficients of the
                                        % filter to be modeled.

b=[.0528 .2639 .5279 .5279 .2639 .0528];
a=[1 0 .6334 0 .0557 0];

len = 1000;                             % Input sequence length.
ORDER = 16;                             % Order of the filter.
K = ORDER;
D = ORDER;
clear Z;

ERR = zeros(1,K);                       % Error vector holding the
                                        % value of each minimum MSE
                                        % for a particular order.

for N = K:-1:1                          % Find the Steady state MSE
    N                                   % for order N, N-1, ... 1.
    etotal = 0;

    for reps=1:10
        reps
        t=0:1:len+D;

                                        % Initialize variables
        e = zeros(1,length(t));         % Error sequnce for each n.
        d = zeros(1,length(t));         % Desired signal.
        x = zeros(1,length(t));         % Input signal.
        y = zeros(1,length(t));         % Output signal.
        W = zeros(1,N);                 % Weight vector.
        err = 0;                        % Input vector.
        X = zeros(length(t),N);
        randn(size('normal'));
```

```
        x = randn(size(t));
        d = filter(b,a,x);              % Desired signal.

        for n = (N+1):len               % delayed LMS algorithm.

            X(n,:) = x(n:-1:(n-N+1));
            y(n) = X(n,:)*W';
            e(n) = d(n) - y(n);
            W = W + ( alpha*e(n-N)*X(n-N,:));

        end

        V = 0;
        for I = n-10:1:n                % Computing the minimum MSE.
            V = V + e(n);
        end

        err = V/10;
        etotal = etotal + err.*err;     % Ensembling minimum MSE.
    end
    ERR(N) = etotal/reps;               % Computing the ensemble
                                        % of minimum MSE.
end

Z = 1:K;
plot(Z,ERR,'r-'); grid
xlabel('Order of the filter')
ylabel('mean squared error')
```

IX. Simulation code for the reliability curves (figure 4.6)

```
% Matlab simulation illustrating the reliabilities of
% pipelined and fault tolerant pipelined adaptive systems.

N = 1:1:32;                 % Order of the system.
R = .95;                    % Reliability of an individual subsystem.

r1 = R.^N;                  % Computing the reliability
                            % of a series system.

                            % Computing the reliability of a
                            % fault tolerant system defined
                            % in equation 4.18.

r2 = r1 + ((N * (1 - R)).*(R.^(N-1)));

plot(N,r1,'-.');            % plot reliability of a series system.
hold
plot(N,r2);                 % plot reliability of fault tolerant system.
grid;
hold off
xlabel('number of processors');
ylabel('reliability');
```