NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

ADAPTIVE FILTERING WITH SYSTOLIC ARRAYS

by

Kristopher Kshonze, B.Sc.

A thesis
submitted to the School of
Graduate Studies and Research
in partial fullfillment of the requirements
for the degree of
Master of Applied Science
in Electrical Engineering

Ottawa-Carleton Institute for Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
University of Ottawa
December, 1987

ISBN 0-315-46792-4

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

ABSTRACT

The QR decomposition realizable in systolic array form is studied and compared to the LMS algorithm. The systolic array that performs the QR decomposition is based on the Givens rotations and the operation of this array has been linked to the conventional RLS algorithm. The differences that have been discovered between the algorithms are explained in detail.

Several hardware structures for the systolic array are presented. With some modifications introduced to the array, the Sequential Decorrelation algorithm (SD) was shown to be equal at convergence to that of the QR decomposition. Finite precision simulations for the SD and the LMS were performed.

In conclusion a systolic architecture for the LMS algorithm was also developed to make the comparison with the QR algorithm more complete.

ACKNOWLEDGEMENTS

The author wishes to express his sincere gratitude to his academic supervisor, Dr. Willem Steenaart for his guidance and encouragement in this research work.

Also the author wishes to acknowledge the constructive advice of Mr. Emil Savov and Dr. Tyseer Aboulnasr.

Finally, the author wishes to thank his wife Catherine for her constant encouragement and patience during his studies.

CONTENTS

LIST OF FIGURES

Chapter 1

Adaptive Noise Cancelling

1.1 General Introduction

The beginning of filtering lies in Wiener's pioneering work that can be found in his book 'Extrapolation, Interpolation and Smoothing of Stationary Time Series' [Wiener 49]. This theory was developed for continuous time series. Shortly after Wiener's work, Levinson considered the discrete time version of Wiener's filtering problem which can be found in the same book. However, Wiener's and as well Levinson's results depend upon knowledge of the statistics for the time series. In the case of Levinson's algorithm the covariance matrix and the cross correlation vector must be known a priori. In practice the statistics of the time series may be unknown or even nonstationary in this case the ensemble average is replaced with time averaged sum involving the actual received data and the covariance matrix is replaced by the sample data covariance matrix. This approach is referred

to as adaptive filtering and will be the main subject of this thesis.

In 1960 Widrow and Hoff [Widrow 60] introduced the Least Mean Square **LMS** algorithm which since then has been the most popular and widely used in many signal processing applications. Modern communication and radar systems need to operate at high data rates in real time, and as a result signal processors with high throughput rate are extremely important. By throughput rate, we mean the rate at which a signal processor can accept input data for processing. A high throughput rate signal processor can be realized by high speed hardware and by efficient design of the signal processing algorithm. One way to achieve high throughput rate is by designing fast as well as time recursive algorithms. The first such algorithm was discovered by Morf, Kailath and Liung [Morf at al 76] and is referred to as the fast Kalman algorithm. (The original Kalman algorithm was discovered in [Kalman 1960]) The fast Kalman has the order of N computations per time update where N is the filter length, however, some researchers have reported that the fast Kalman can be numerically unstable [Haykin 86]. Another fast algorithm for adaptive filtering is the Least Squares Lattice Algorithm. The LSLA has the same numerical complexity as the fast Kalman, however, it has better numerical stability. The interesting feature of the LSLA is that it is both order and time recursive. The LSLA gives the filtered output directly without computing the corresponding weights, it preforms the filtering by calculating the partial coefficients associated with the autocorrelation matrix of the input. Another method for obtaining high throughput rate is by developing algorithms that can take advantage of concurrent processing. This is the idea behind systolic

arrays as will be discussed in Chapter 2. Briefly however, the constraints of local communication, simplicity of processing cells and simple data paths allow systolic arrays to be realized in hardware with Very Large Scale Integration technology. Not only that the LSLA is a fast algorithm but due to its modular structure it can be mapped into a systolic array. By taking advantage of parallel processing the LSLA can achieve a throughput rate independent of its order. Another excellent candidate for fast filtering using the concurrent processing is the **QR** decomposition using Givens plane rotations for solving the least squares problem [Gentleman 81]. This approach is particularly well designed for the systolic architecture and is the main topic of this thesis. The QR decomposition using Givens plane rotations differes from LSLA in the sense that it is not order recursive.

This thesis is concerned with the comparison of the LMS algorithm to the **QR** decomposition realized in systolic array form. Also the **QR** decomposition is evaluated in terms of hardware complexity, misadjustment and speed of convergence. Finally a systolic structure is proposed for realization of the LMS. In spite of the development of the new algorithms the LMS still remains the most widely used due to simplicity of its implementation, efficiency and numerical stability. On the other hand the **QR** decomposition using systolic arrays is a new method developed previously only in theory with essentially no actual results (simulations) and with no remarks on its efficiency. Thus the comparison is between a well proven method and a new candidate for adaptive filtering in terms of speed, quality (misadjustment) and complexity. The application used for the comparison is Adaptive Noise Cancelling (**A.N.C.**) which is a branch of adaptive filtering.

The aim of this work, however, is to refer to the general case of adaptive filtering through the use of noise cancelling. This assumption is legitimate since the principles involved in noise cancelling are essentially the same as in echo cancelling or channel equalization thus the conclusions drown from the comparison are valid for any other application of adaptive filtering. The terms noise cancelling and adaptive filtering are synonymous throughout this thesis. In the remaining part of this chapter the principles of the LMS and the Least Squares method are developed and explained. It is particularly important for the understanding of the systolic array (**QR** algorithm) described later as we will make the connection between the systolic array (**QR** algorithm) and the Recursive Least Squares (RLS) algorithm. The open loop method is described as it is the basis for the systolic array operation contrary to the LMS which operates in a closed loop system. In the second chapter the operation of the systolic array is examined in detail and original simulations are presented. Also the need for rescaling the error is shown along with a updated expression for the self noise (misadjustment) in the systolic array. The third chapter deals with the practical realization of the array giving a possible hardware implementation. Through modifications to the original array two other possible structures [McWhirter 86] ,that require less hardware, are presented. One of these structures is the Sequential Decorrelation algorithm (SD) [McWhirter 86], which was shown to be equal in terms of self noise (misadjustment) to the original QR decomposition with systolic arrays. Simulations of the SD array and the LMS in fixed point arithmetic are presented. The fourth chapter deals with possible implementation of the LMS algorithm itself in systolic structure

based on distributed arithmetic convolution [Walicki S5]. In chapter five the final conclusion on the comparison between the algorithms under their systolic array realization is presented.

The original features of the Thesis are;

- Application of the QR systolic array to noise cancelling and comparison to the LMS algorithm in terms of;
  1. Self Noise (Misadjustment)
  2. Speed
  3. Finite Precision
- The need to rescale the a posteriori error in the QR decomposition with systolic array for the case when $\lambda < 1$
- Systolic update structure for the LMS algorithm

1.2 A.N.C. based on Wiener Theory

Before we start the discussion we must note that Wiener Theory provides a unique solution to the filtering problem and not a method of arriving to this solution. Or in other words the solution is unique regardless of the method used. However, in this section for simplicity reasons we will arrive at Wiener solution by using a specific method, namely the closed loop approach.

A closed loop adaptive filtering system is shown in Figure 1.1. It consists of a transversal filter with N variable tap weights, an adaptive al-

gorithm which adjusts these weights in some consistent manner and two inputs which must be correlated. The actual (desired) response d is one of the inputs to the system and is composed of a signal s and an undesired interference n_0 thus $d = s + n_0$. The second input consists of undesired interference n_1 which is correlated to the interference in the desired response but is uncorrelated with the signal. When the undesired interference is noise the system is a noise canceller. Although n_0 and n_1 are correlated they are not identical, since they are being sensed at two different locations.

For example consider a mobile telephone located in a vehicle where the engine noise is causing the interference, the desired response (speech and noise) comes from inside the vehicle but the reference signal in form of engine noise must come from outside the vehicle since we do not want to capture any speech in the reference input. Clearly the noise n_0 is somewhat different from n_1 as it travelled through different media, still some correlation must exist because of the common origin. So our objective is, removing the interference imposed on the signal by using the statistical information located in the reference input. (The following analysis is based on [Widrow 75])

The input vector of length N to the tapped delay line is the delayed version of the reference noise n_1 and can be defined at time m as:

$$\mathbf{X}^T(m) = (n_1(m), n_1(m-1), \ldots, n_1(m-N)) \qquad (1.1)$$

and the tap weight vector can be described by:

$$\mathbf{W}^T(m) = (w_1(m), w_2(m), \ldots, w_N(m)) \qquad (1.2)$$

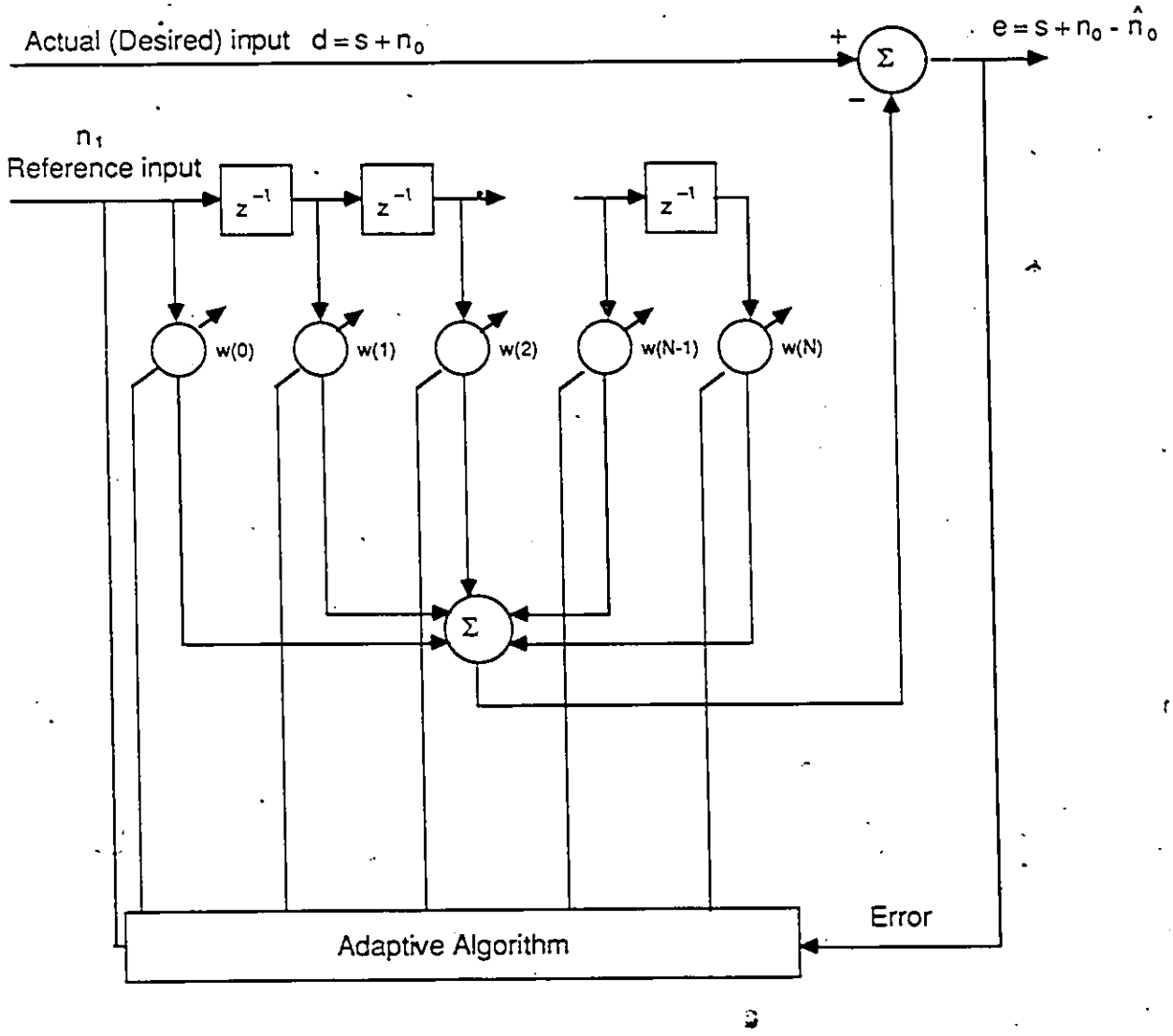


Figure 1.1: Closed Loop Adaptive Filter

The output of the filter is the inner product of these two vectors and is referred to as the linear combiner, in general at any time;

$$\hat{n}_0 = \mathbf{X}^T\mathbf{W} = \mathbf{W}^T\mathbf{X} \quad . \tag{1.3}$$

where $\hat{n}_0$ is the estimate of the noise n_0 in the actual (desired) input. Using the definition $d = s + n_0$ the output of the system becomes:

$$e = s + n_0 - \hat{n}_0 \tag{1.4}$$

Squaring this term and taking the expected value with the assumption that s, n_0, and $\hat{n}_0$ are zero mean leads us to the expression:

$$E\{e^2\} = E\{s^2\} + E\{(n_0 - \hat{n}_0)^2\} \tag{1.5}$$

The signal power will be unaffected if we try to minimize the mean square error signal, therefore the minimum mean square error signal occurs when $E\{(n_0 - \hat{n}_0)^2\}$ is minimum.

$$minE\{e^2\} = E\{s^2\} + minE\{(n_0 - \hat{n}_0)^2\} \tag{1.6}$$

When the expression $E\{(n_0 - \hat{n}_0)^2\}$ is zero, a case which in practice does not occur exactly, the error signal e is at its minimum and is equivalent to the signal itself $e = s$. This is perfect noise cancellation.

Going back to (1.4) we can express the error e in terms of the weight vector;

$$e = d - \mathbf{X}^T\mathbf{W} \tag{1.7}$$

squaring this expression and taking the expected value we obtain;

$$E\{e^2\} = E\{d^2\} - 2E\{d\mathbf{X}^T\}\mathbf{W} + \mathbf{W}^T E\{\mathbf{X}\mathbf{X}^T\}\mathbf{W} \tag{1.8}$$

now we can define $\mathbf{R}$ as the $N \times N$ autocorrelation matrix of the tap input vector $\mathbf{X}$ and $\mathbf{P}$ as the $1 \times N$ crosscorrelation vector between the tap input vector and the desired response d, such that:

$$\mathbf{R} = E\{\mathbf{X}\mathbf{X}^T\} \tag{1.9}$$

$$\mathbf{P} = E\{d\mathbf{X}^T\} \tag{1.10}$$

With these new definitions we can rewrite (1.8) as:

$$E\{e^2\} = E\{d^2\} - 2\mathbf{P^T W} + \mathbf{W}^T\mathbf{R}\mathbf{W} \tag{1.11}$$

From this expression we can see that the mean square error is a quadratic function of the weights with a bowl shaped surface. The filter is to adjust its weights so that the bottom of the bowl is reached where the mean square error is at its minimum value. Noting that the gradient at the bottom is zero, we can derive an optimum solution for the weights denoted by $\mathbf{W}_{opt}$ first by finding the gradient of (1.11) and second by setting this gradient to zero.

$$\nabla = \begin{bmatrix} \frac{\partial E\{e^2\}}{\partial w_1} \\ \vdots \\ \frac{\partial E\{e^2\}}{\partial w_n} \end{bmatrix} = -2\mathbf{P} + 2\mathbf{R}\mathbf{W} \tag{1.12}$$

$$\nabla = 0 = -2\mathbf{P} + 2\mathbf{R}\mathbf{W}_{opt} \tag{1.13}$$

$$\mathbf{W}_{opt} = \mathbf{R}^{-1}\mathbf{P} \tag{1.14}$$

Expression 1.14 is the matrix form of the Wiener Hopf equation [Widrow 75] also referred to as the normal equation. The reason for this name (normal) lies in the principle of orthogonality, as the output error must be

normal to the output of the filter (linear combiner) when $\mathbf{W}_{opt}$ is reached. There are numerous other algorithms based on different methods that can be used in adaptive filtering. For each case $\mathbf{W}_{opt}$ is the solution to which all algorithms should converge. In the next section of this chapter we will describe two families of these adaptive algorithms.

1.3 Steepest Descent Method and the LMS

One method of finding the optimum weight vector $\mathbf{W}_{opt}$ is through the method of steepest descent which is a well known optimization method [Murray 72]. In this method we first assume an initial value for $\mathbf{W}_{opt}$ (zero usually), second we compute the gradient vector of the mean square error with respect to the old tap weight vector and third we compute the new tap weight vector by making a change in the previous weight vector in a direction opposite to the gradient. After that we go back to step two until $\mathbf{W}_{opt}$ is reached. This procedure can be expressed by a simple recursive relation;

$$\mathbf{W}(n+1) = \mathbf{W}(n) + \frac{1}{2}\mu(-\nabla) \tag{1.15}$$

where μ is the step size parameter or the gain that controls the size of incremental correction applied to the tap weight vector. If the autocorrelation $\mathbf{R}$ and the crosscorrelation $\mathbf{P}$ were known at each step then the exact gradient could be obtained and the algorithm would converge to $\mathbf{W}_{opt}$ after N iterations. However, under real conditions, the exact gradient can not be found and it must be estimated directly from the data. We can rewrite

(1.12) for the gradient estimate as;

$$\widehat{\nabla} = \begin{bmatrix} \frac{\partial e^2}{\partial w_1} \\ \vdots \\ \frac{\partial e^2}{\partial w_n} \end{bmatrix} = -2\mathbf{P}_{est} + 2\mathbf{R}_{est}\mathbf{W} \tag{1.16}$$

where the instantaneous estimates $\mathbf{P}_{est}$ and $\mathbf{R}_{est}$ replace the exact values in (1.10) and (1.9) thus we can rewrite;

$$\mathbf{R}_{est} = \mathbf{X}^T(n)\mathbf{X}(n) \tag{1.17}$$

$$\mathbf{P}_{est} = \mathbf{X}(n)d(n) \tag{1.18}$$

now replacing the gradient in (1.15) by its estimate will yield a new expression known as the LMS algorithm;

$$\begin{aligned} \mathbf{W}(n+1) &= \mathbf{W}(n) + \mu\mathbf{X}(n)e(n) \\ e(n) &= d(n) - \mathbf{W}^T(n)\mathbf{X}(n) \end{aligned} \tag{1.19}$$

In order to measure and compare the quality of adaptive algorithms we must define criteria of performance known as misadjustment. The misadjustment $\mathcal{M}$ is defined as the dimensionless ratio of the average excess mean squared error (self noise of the filter) to minimum mean squared error (power of the signal s). The misadjustment is a measure of the cost of adaptivity. For example a misadjustment of 10 percent means that the adaptive algorithm produces mean square error 10 percent greater then the optimum mean square error.

The estimates for $\mathbf{R}$ and $\mathbf{P}$ are very noisy and it may seem as if the algorithm will have poor performance, however, the LMS is an iterative

algorithm and it will tend to average those estimates during the adaptation process. A typical learning curve for the LMS algorithm is shown in Figure 1.2 for 16 taps and the step size set to 0.005. The great advantage of the LMS algorithm lays in its simple implementation as the entire filtering process can be described by (1.19). The disadvantages of the LMS on the other hand, are:

- Dependance on the step size: large step size yields fast convergence with large misadjustment as only a small amount of data was considered for the solution, while small step size will give slow convergence with small misadjustment since a large amount of data was considered. We can view the step size as the memory length of the LMS algorithm: large step size corresponds to a short memory and small step size results in a long memory.

- The eigenvalue spread of the correlation matrix $\mathbf{R}$. When the eigenvalues are widely spread the LMS algorithm will require a large number of iterations to converge [Haykin 86].

- The misadjustment $\mathcal{M}$ and is given by:

$$\mathcal{M} \approx \mu N(Signal\ Power) \tag{1.20}$$

  which is seen proportional to the number of taps [Widrow 85]. However, since μ is also dependent on N the misadjustment can be equivalent for different order filters.

Other simulations of the closed loop filtering system with the LMS algorithm will be shown in the following chapters.

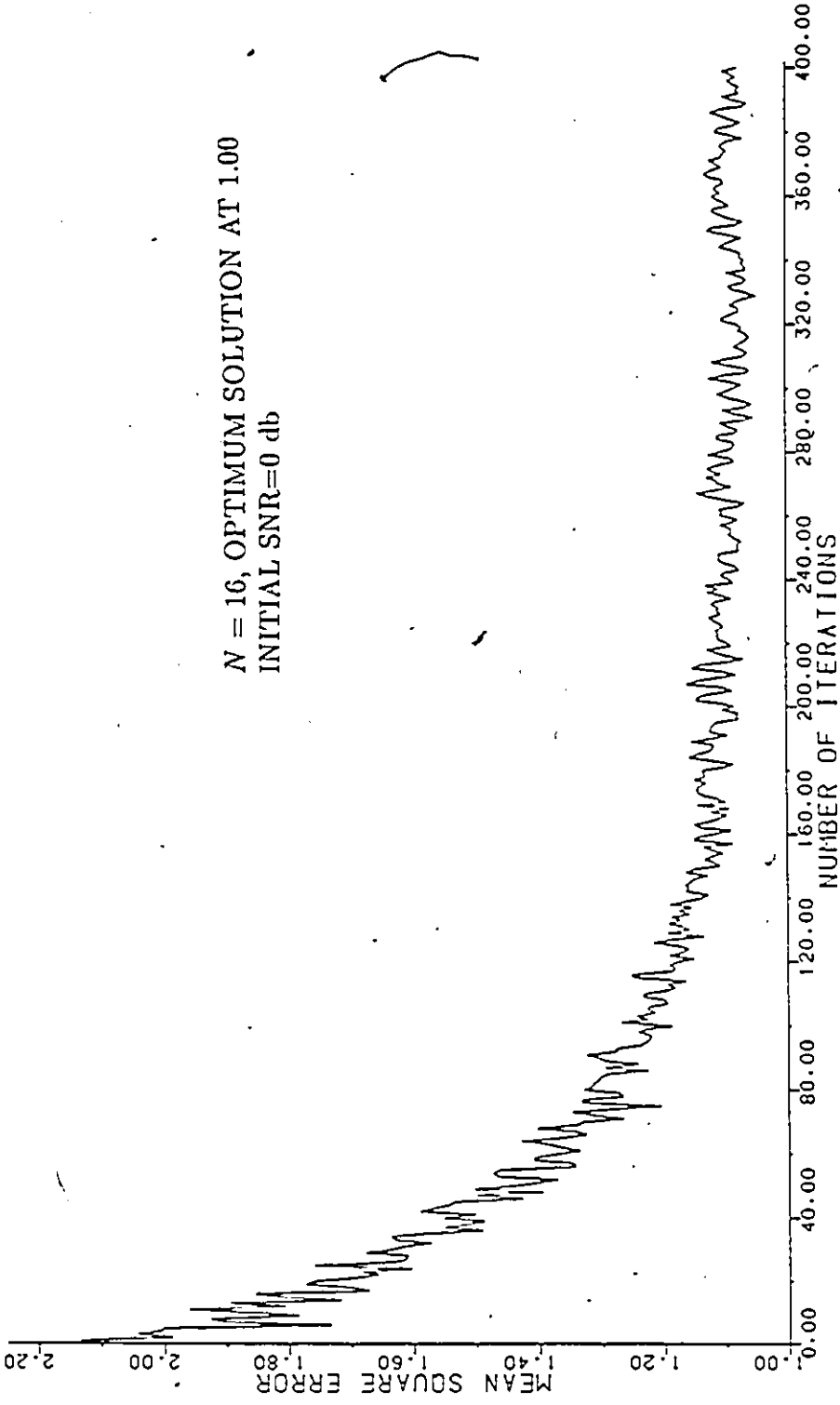


Figure 1.2: LMS Learning Curve

1.4 Least-Squares and the RLS

In section 1.2 we considered the Wiener filter which was derived from ensemble averages and resulted in one optimum solution. With the least-squares method on the other hand we try to give a solution for every new data input. The least squares method will try to minimize a deterministic error criterion with an exact solution at every point in time whereas the Wiener method has an statistical error criterion which is minimized only at convergence. In the method of least squares we choose the tap weights to minimize the error criterion that consists of the *sum of error squares*;

$$\mathcal{E}\{(n)\} = \sum_{i=1}^{n} \mid e(i) \mid^2 \tag{1.21}$$

as opposed to the Wiener solution where the criterion was the *mean square error*.

$$\mathcal{J}\{\mathbf{W}\} = E\{e(n)^2\} \tag{1.22}$$

The index of performance defined by (1.21) should include a weighting or forgetting factor and the modified index of performance becomes:

$$\mathcal{E}(n) = \sum_{i=1}^{n} \lambda^{n-i} \mid e(i) \mid^2 \tag{1.23}$$
$$0 < \lambda \leq 1$$

Expression (1.23) defines the exponentially weighted least squares method. The importance of λ will become evident later. The optimum value of $\mathbf{W}$ for which $\mathcal{E}(n)$ attains its minimum value is defined by the normal equation similar to (1.14) but with different definitions for the autocorrelation matrix ($\mathcal{R}(n)$) and the crosscorrelation vector ($\mathcal{P}(n)$). Assuming that the

prewindowing method is used where the data prior to $i = 1$ is assumed to be zero and no assumption is made about the data after $i = n$ (n is variable length of the observable data). The data matrix $\bar{\mathbf{X}}(n)$ takes the form of;

$$\bar{\mathbf{X}}(n) = \begin{bmatrix} x(1) & x(2) & \cdots & x(N) & \cdots & x(n) \\ \vdots & x(1) & \cdots & x(N-1) & \cdots & x(n-1) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x(1) & \cdots & x(n-N) \end{bmatrix} \tag{1.24}$$

The columns of the data matrix are the set of vectors $\mathbf{X}(n)$ and the desired response vector $\bar{d}$ consists of set of observations d, now (1.14) becomes;

$$\bar{\mathbf{X}}(n)\bar{\mathbf{X}}^T(n)\mathbf{W} = \bar{\mathbf{X}}(n)\bar{d} \tag{1.25}$$

The autocorrelation can be written as;

$$\mathcal{R}(n) = \sum_{i=1}^{n} \lambda^{n-i}\mathbf{X}^T(i)\mathbf{X}(i) \tag{1.26}$$

or in a recursive form,

$$\mathcal{R}(n) = \lambda\mathcal{R}(n-1) + \mathbf{X}^T(n)\mathbf{X}(n) \tag{1.27}$$

similarly,

$$\mathcal{P}(n) = \sum_{i=1}^{n} \lambda^{n-i}\mathbf{X}(i)d(i) \tag{1.28}$$

$$\mathcal{P}(n) = \lambda\mathcal{P}(n-1) + \mathbf{X}(n)d(n) \tag{1.29}$$

Now the function of the exponential weighting factor becomes evident since $\mathcal{R}(n)$ and $\mathcal{P}(n)$ in 1.27 and 1.29 are numerically growing functions and for large n the solution $\mathbf{W}$ will not be affected by changes in the environment

since the current input $\mathbf{X}^T(n)\mathbf{X}(n)$ or $\mathbf{X}(n)d(n)$ will be very small compared to the large accumulated values of $\mathcal{R}(n-1)$ and $\mathcal{P}(n-1)$. So the function of the weighting factor is to limit number growth and by that preventing overflow as well as giving the weight solution a dynamic response to changes in the environment. The weighting factor λ can also be viewed as a forgetting factor where the data in the distant past are forgotten, the length of the memory ML can be approximated by the expression;

$$ML = \frac{1}{1-\lambda} \tag{1.30}$$

for the case when $\lambda = 1$ the memory is infinite and we consider all data from time 1 up to n. The curve of the forgetting factor is shown in Figure 1.3. When the index $n-i$ is zero it corresponds to the most recent observation $(n=i)$ which is scaled by 1 and all other past observation from that point will be scaled exponentially, thus the fiftieth data point from the current data will be scaled by λ^{49}. Using the previous definition and the matrix inversion lemma it is possible to derive the exponentially weighted recursive least squares algorithm RLS which can be summarized as follows. (Based on [Haykin 86])

The gain vector $k(n)$ has the same function as the step size μ in the LMS algorithm and is expressed as;

$$k(n) = \frac{\lambda^{-1}\mathcal{R}^{-1}(n-1)\mathbf{X}(n)}{1+\lambda^{-1}\mathbf{X}^T(n)\mathcal{R}^{-1}(n-1)\mathbf{X}(n)} \tag{1.31}$$

or equivalently,

$$k(n) = \mathcal{R}^{-1}(n)\mathbf{X}(n) \tag{1.32}$$

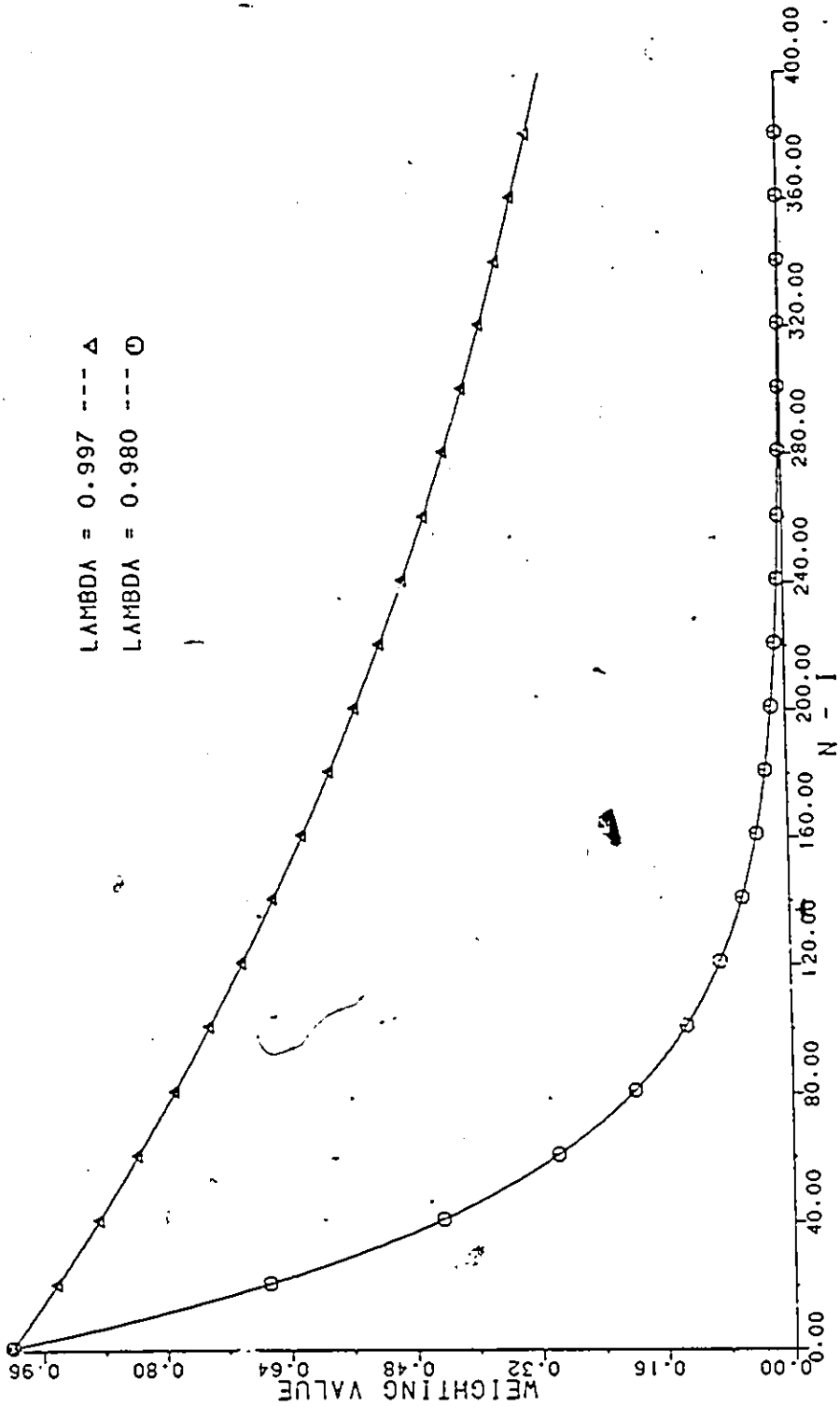


Figure 1.3: Exponential Weighting (Forgetting) λ^{n-i}

the inverse autocorrelation matrix is written as;

$$\mathcal{R}^{-1}(n) = \lambda^{-1}\mathcal{R}^{-1}(n-1) - \lambda^{-1}k(n)\mathbf{X}^T(n)\mathcal{R}^{-1}(n-1) \tag{1.33}$$

and the time update for the weight vector is given by;

$$\mathbf{W}(n) = \mathbf{W}(n-1) + k(n)\alpha(n) \tag{1.34}$$

where

$$\alpha(n) = d(n) - \mathbf{W}^T(n-1)\mathbf{X}(n) \tag{1.35}$$

The variable $\alpha(n)$ represents the output of the filter at time n with tap weights computed at $(n-1)$. We refer to $\alpha(n)$ as the *a priori* estimation error. This error is different from the general error considered in previous sections where the tap weights were computed at time n, as shown below;

$$e(n) = d(n) - \mathbf{W}^T(n)\mathbf{X}(n) \tag{1.36}$$

We refer to the error in 1.36 as the *a posteriori* estimation error. It is reasonable to assume that after convergence both errors will be equal since $\mathbf{W}(n-1) \approx \mathbf{W}(n)$. Now the RLS algorithm has been fully described with gain vector 1.31, filtering operation 1.35, update of the tap weights 1.34 and update of the inverse correlation matrix 1.33.

It is advantageous to examine the a priori error $\alpha(n)$ rather then the a posteriori error $e(n)$ since $\alpha(n)$ has a large value at $n = 1$ and then decays with time which makes the shape of the learning curve (mean square error) similar to the LMS algorithm. Whereas the a posteriori error attains small value at $n = 1$ and then increases with time. When $\lambda = 1$ (infinite memory)

the convergence of the a priori error is given approximately by [Haykin 86];

$$E\{\alpha(n)^2\} \approx \sigma^2 + \frac{N\sigma^2}{n} \tag{1.37}$$

where N is the number of taps and σ^2 is the variance of the minimum mean square error or the signal power of the desired signal. From this expression we can see that the RLS algorithm converges approximately in $2N$ iterations and when n is large the algorithm produces zero excess mean squared error or in other words zero misadjustment. The RLS algorithm is also insensitive to eigenvalue spread unlike the LMS. For the case when $\lambda < 1$ expression 1.37 is no longer true since the tap weights are not at their optimum setting because of the limited memory. That causes noise to appear in the solution which will result in misadjustment or excess mean square error J_{ex} given by 1.38 [Ling 84],

$$E\{J_{ex}(\infty)\} \approx \frac{1-\lambda}{1+\lambda} N\sigma^2 \tag{1.38}$$

expression 1.38 is restricted for values of λ very close to 1.

It is possible to relate the a priori error to the a posteriori error by introducing another variable $\gamma(n)$ and referred to as the *conversion factor*,

$$\gamma(n) = 1 - k^T(n)\mathbf{X}(n) \tag{1.39}$$

where $k(n)$ is the gain vector defined by (1.31), $\gamma(n)$ can also be written as;

$$\gamma(n) = \frac{1}{1 + \lambda^{-1}\mathbf{X}(n)\mathcal{R}^{-1}\mathbf{X}(n)} \tag{1.40}$$

or,

$$\gamma(n) = \lambda \frac{det[\mathcal{R}(n-1)]}{det[\mathcal{R}(n)]} \tag{1.41}$$

The amazing feature of the conversion factor is shown below where the a priori error multiplied by the conversion factor yields the current error.

$$e(n) = \gamma(n)\alpha(n) \tag{1.42}$$

Thus through the use of the conversion factor we can obtain the current estimate even before updating the tap weights. It should be noted that the a posteriori error derived in this manner is the scaled version of the a posteriori error defined in (1.35) for the case when $\lambda < 1$. This can be verified by examining (1.41) for large n. In that case $det[\mathcal{R}(n-1)] \approx det[\mathcal{R}(n)]$ and $\gamma(n) \approx \lambda$, then the a posteriori error becomes scaled through λ,

$$e(n) = \lambda d(n) - \lambda \mathbf{W}^T(n)\mathbf{X}(n) \tag{1.43}$$

So to obtain the same value as in (1.35) we must rescale with λ. Now that the RLS and the LMS algorithms have been fully described we can turn to a different structure of adaptive filter realization.

1.5 Open Loop Filtering

So far we have considered only the closed loop system (Figure 1.1) where the error in both algorithms LMS and RLS updates the tap weights. For the LMS algorithm we mentioned that the loop performance and stability are dependent upon the internal gain and the power level of the reference input as well as on the eigenvalue spread of the autocorrelation matrix which governs the speed of convergence. Although the RLS algorithm is not affected

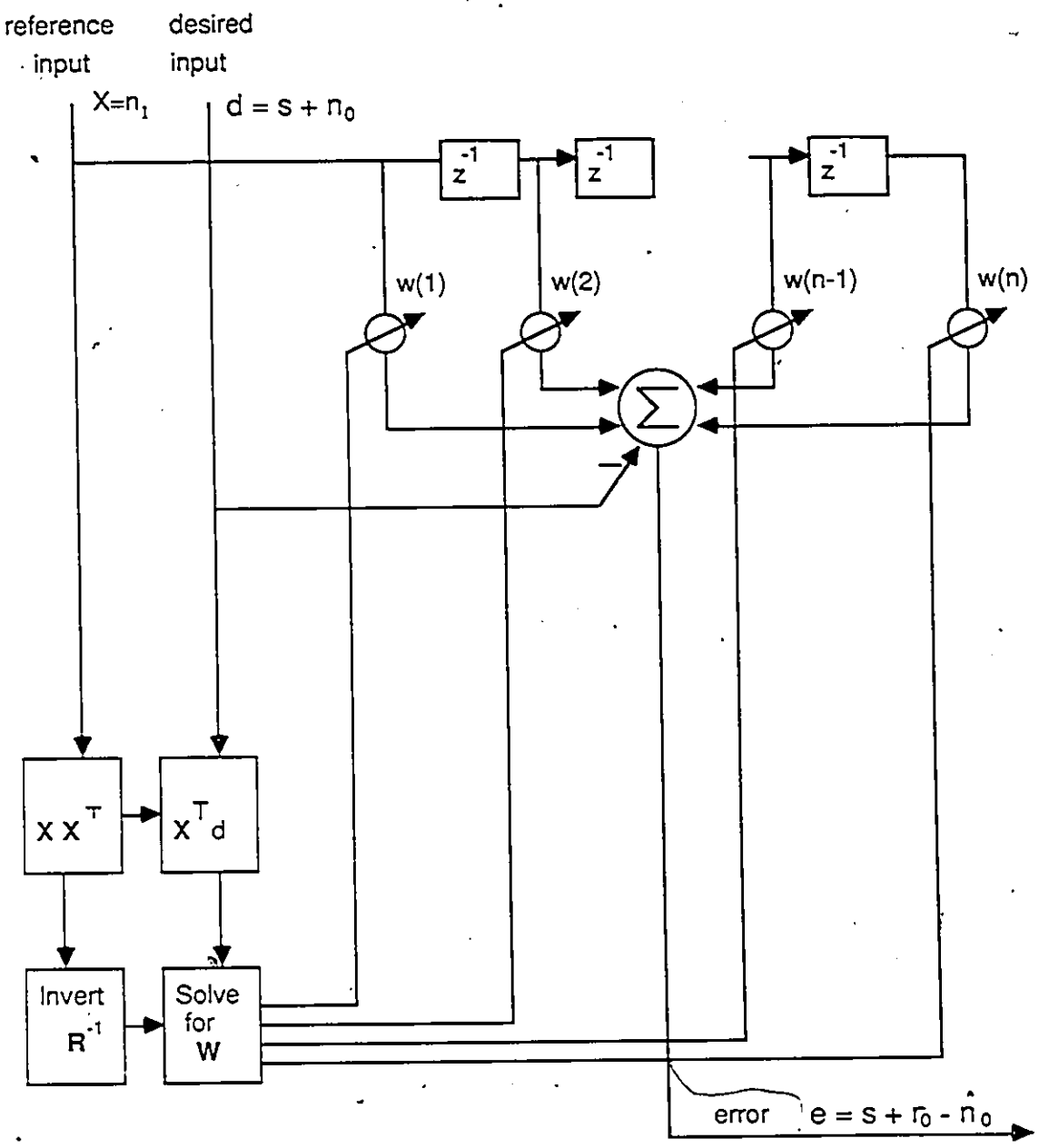


Figure 1.4: Open Loop System

by the same problems, the least squares problem can be solved in a different way using the direct solution approach with an open loop system as shown in Figure 1.4. The direct approach, also known as the sample matrix inversion, is based on the direct solution of the normal equation where the autocorrelation matrix is inverted and multiplied by the crosscorrelation to produce tap weight values. Since $\mathcal{R}$ and $\mathcal{P}$ are exact with all the data being considered (accumulated samples and N new samples), the tap weight values are the exact solution to the normal equation at every step and there is no need to update them with an iterative system (closed loop). The advantage of this method is that it requires small amount of data in order to give a good description of the external environment and that the speed of convergence is fast. Also since the cost of computation is continually decreasing relative to the cost of aperture in space or time, it is desirable to avoid statistical iterations, and rather to provide at each time the best least squares solution possible with the available data [Speiser 86]. The disadvantage of this method on the other hand lies in the fact that the matrix inversion procedure requires high precision arithmetic since the squaring procedure in estimating the autocorrelation matrix can cause the matrix to be poor conditioned (unequal eigenvalues). If high precision arithmetic is not used, solving the poor conditioned equations can result in numerical instability and poor system performance. As well, the open loop system described in Figure 1.4 requires complicated architecture, particularly the matrix inversion block is hard to realize. One part of the system must form and store the autocorrelation matrix, another must compute the weight vector and yet another must apply the weight vector to the data. The high

requirement for storage during the computation of the weight vector and the need for large number of high speed data communication buses along with a sophisticated control unit necessary for the delivery of the instructions to the various parts make this architecture very complicated, hard to design and not suitable for large scale integration.

Conclusion of the Chapter

In this chapter we have examined two approaches to adaptive filtering. The first, the closed loop approach, was implemented with two methods the gradient method (LMS algorithm) and the least squares method (RLS algorithm). The second approach (open loop) was implemented with the matrix inversion method (direct solution) which was shown to be not very efficient. However, the open loop approach itself is superior to the closed loop approach in terms of speed and the dependance on signal characteristics, thus it would be only natural to try to find an efficient method of implementing the open loop approach. This will be done in the next chapter where the QR decomposition will be used for open loop filtering and will be proven to be a better method then matrix inversion.

Chapter 2

A.N.C. with Systolic Arrays

2.1 Systolic Arrays Overview

'The increasing demands of high speed real time Digital Signal Processing (DSP) coupled with low cost, high density VLSI based architectures are a viable approach to achieve a tremendous computation capability in a cost effective way' [H.T. Kung 84]. In order to achieve equivalent performance of systolic arrays using conventional architectures and methodologies, complicated systems (or programming) have to be designed with penalty in cost, size and power consumption. Parallel processing and custom design are major factors to building high performance signal processors with reasonable hardware complexity. In designing special purpose processors we must take into account the limitations and difficulties involved in VLSI architecture. For large scale computational system requiring high throughput, locality of data flow and the controller becomes an important factor in VLSI

design. This locality property simplifies the hardware design of the chip. Another important consideration is modularity of the processors. Modularity will reduce the design difficulty and the architecture could be easily expended to fit larger computational processing size if necessary. Finally faster throughput rate can be achieved by pipelinability of the architecture.

H.T Kung and C.E Leiserson [H.T Kung 78] introduced a new class of array processors called systolic arrays that allow implementing algorithms previously thought to be inefficient and also a way to implement currently used signal processing and matrix algorithms in a regular and modular manner. Systolic arrays are a network of several interconnected (ideally) identical cells laid out to the guidelines of VLSI design; locality, modularity and pipelinability. In systolic arrays each cell performs some basic operation (add, multiply) and transfers the intermediate results rhythmically to the neighboring cell. Each cell communicates with only its nearest neighbors, except for boundary cells which also communicate with the external environment, thus local communication is required only. All the cells are under the control of a master clock. The data and the intermediate results flow through the array under the 'beat' of the master clock and that is the reason for the name systolic. The great advantage of this array is that data can be pumped regularly through the system, thus providing regular synchronous fast throughput rate. Systolic arrays are being extensively studied as high speed processors of signals and images.

The disadvantage of the systolic arrays lies in the global control of the data movement in different cells. In order to allow proper timing and synchronization in systolic arrays, extra delays are needed to assure that all

the cells have the same processing time. This slows down the computation and therefore decreases the throughput rate. Furthermore, for large number of processing cells the synchronization can become difficult. This problem can be solved by asynchronous data driven cells that allows local control and self timed cells in which each cell operation is triggered by the availability of the data. This idea was developed and applied to systolic arrays by S.Y Kung [S.Y. Kung 82] and is referred to as a wavefront array. The major difference between the two architectures is that the wavefront array transfers the data to the next cell asynchronously depending only on the availability of data, whereas the systolic arrays require global timing and control of data flow.

2.2 Least Squares Method with Systolic Arrays

In section 1.4 we gave an example of an open loop system which solves the least squares problem by using the matrix inversion method, in this section an alternative way of solving essentially the same problem is shown. The least squares problem can be solved at every clock period with a triangular systolic array which performs orthogonal triangularization using Givens rotations. The array which has been proposed first by Kung and Gentleman [Gentleman 81] is shown in figure 2.1. It uses the **QR** decomposition, which first brings the data to an upper triangular matrix form and second solves the normal equation (1.25) by back substitution. This method is superior

to the matrix inversion method (described in section 1.5) as the formation of covariance matrix (autocorrelation) is not required. Any numerical algorithm which avoids forming the covariance matrix explicitly and operates directly on the data is likely to be much better conditioned' [Christopher R. Ward et all 1986]. Also the rounding error caused by the finite word length effect will not grow. Since we do not need to perform the squaring operation in the left hand side of (2.1). The normal equation is given by:

$$\bar{\mathbf{X}}\bar{\mathbf{X}}^T\mathbf{W} = \bar{\mathbf{X}}^T\bar{d} \tag{2.1}$$

using the decomposition:

$$\bar{\mathbf{X}} = \mathbf{Q}\hat{\mathbf{R}} \tag{2.2}$$

where $\mathbf{Q}$ is a orthogonal matrix such that

$$\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$$

$$\mathbf{Q}^T = \mathbf{Q}^{-1} \tag{2.3}$$

and $\hat{\mathbf{R}}$ is upper triangular matrix, we can rewrite the normal equation as:

$$\bar{\mathbf{X}}^T\bar{\mathbf{X}}\mathbf{W} = \hat{\mathbf{R}}^T\mathbf{Q}^T\mathbf{Q}\hat{\mathbf{R}}\mathbf{W} = \hat{\mathbf{R}}^T\hat{\mathbf{R}}\mathbf{W} = \hat{\mathbf{R}}^T\mathbf{Q}^T\bar{d} \tag{2.4}$$

or

$$\hat{\mathbf{R}}\mathbf{W} = \mathbf{Q}^T\bar{d} \tag{2.5}$$

Ill conditioning occurs if the matrix $\hat{\mathbf{R}}$ has a very small determinant in which case the solution deviation from the true solution (weight vector W) can be quite large and still satisfy (2.1) accurately. The numerical quality of system of linear equations is measured by the condition number Cn of the coefficient matrix ($\hat{\mathbf{R}}$ in our case), which is defined as the

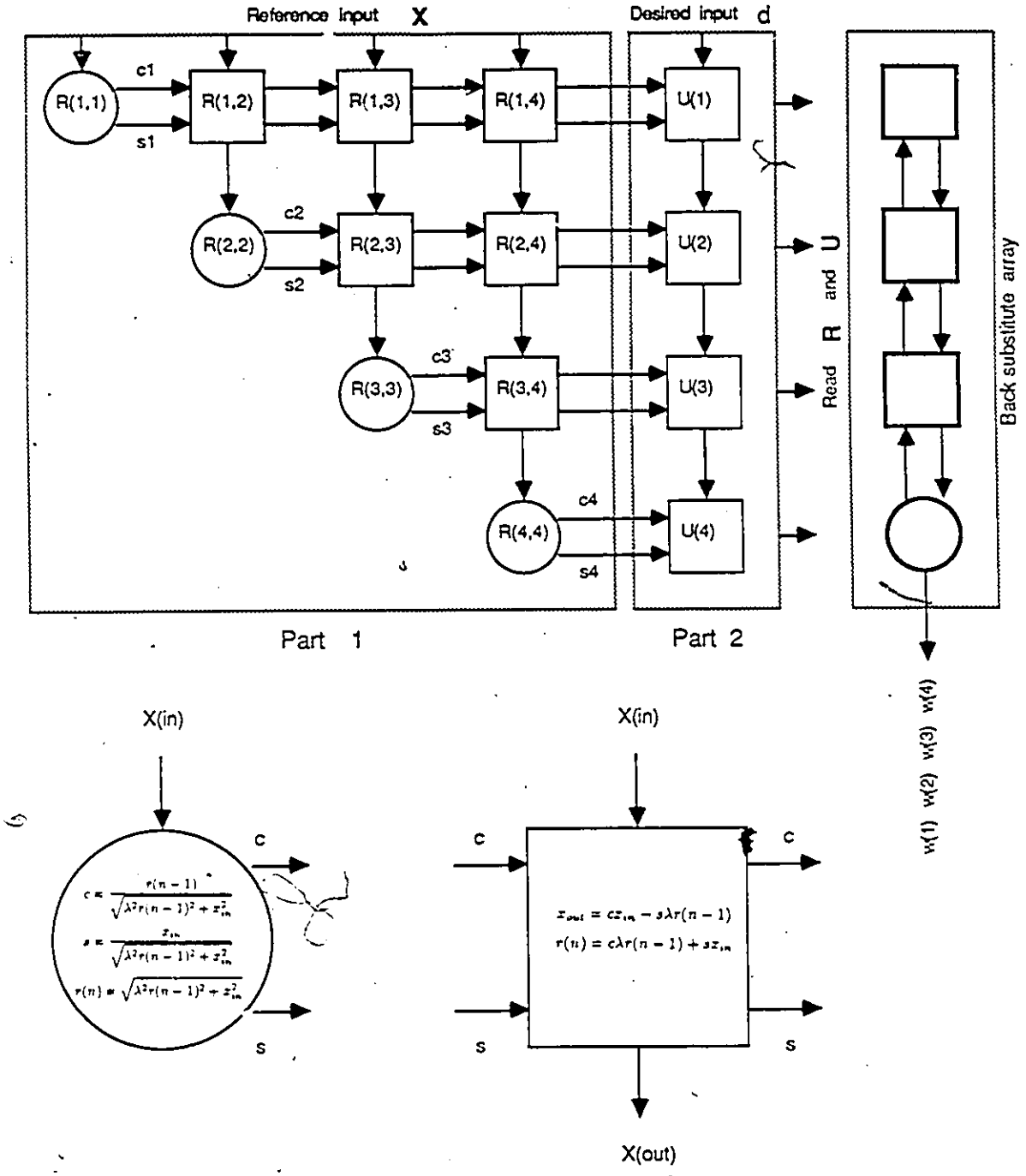


Figure 2.1: Triangular systolic array with back substitution

ratio of the largest and the smallest eigenvalues of a matrix, $Cn = \beta_l/\beta_s$. In the case of the direct solution the condition number is proportional to $Cn[\bar{\mathbf{X}}^T(n)\bar{\mathbf{X}}(n)] = Cn^2[\bar{\mathbf{X}}(n)]$ where in the case of **QR** decomposition the condition number becomes $Cn[\mathbf{R}(n)] = Cn[\mathbf{Q}(n)\bar{\mathbf{X}}(n)] = Cn[\bar{\mathbf{X}}(n)]$. Clearly the **QR** decomposition is better conditioned since its condition number corresponds to the condition number of the data matrix and not as in the direct solution (section 1.5) where the condition number is much greater then the corresponding Cn of the data matrix.

Due to the decomposition process as was done in 2.4 the upper triangular matrix $\hat{\mathbf{R}}$ and $\mathbf{Q}^T\bar{d}$ no longer represent the autocorrelation matrix $\mathcal{R}$ and the crosscorrelation vector $\mathcal{P}$. The solution $\mathbf{W}_{opt}$, however, is still the same as in (2.1). The matrix $\hat{\mathbf{R}}$ can be seen as the product of Cholesky factorization process, or in other words as the Cholesky factor.

The **QR** decomposition or the triangularization process can be carried out by several methods for example by the Gram Schmidt orthogonalization or the Householder transformation [Strang 80]. The Givens rotation method [Givens 58] is more practical since it can process the data $\mathbf{X}$ one row at a time and can be used in a continuous manner as new data is generated and entered in the array, whereas with the other methods all the data must be available prior to the triangularization process.

The Givens transformations, used on two row vectors at a time, eliminate the leading element of the lower vector, as shown in an example below;

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} r_i & r_{i+1} & \dots & r_k \\ x_i & x_{i+1} & \dots & x_k \end{bmatrix} = \begin{bmatrix} \bar{r}_i & \bar{r}_{i+1} & \dots & \bar{r}_k \\ 0 & \bar{x}_{i+1} & \dots & \bar{x}_k \end{bmatrix} \quad (2.6)$$

For this operation to be true, the rotating elements c and s, which can be

seen as cosine and sine, must satisfy,

$$-sr_i + cx_i = 0 \tag{2.7}$$

$$s^2 + c^2 = 1 \tag{2.8}$$

from 2.7 and 2.8 it follows that,

$$s = \frac{x_i}{\sqrt{x_i^2 + r_i^2}} \tag{2.9}$$

$$c = \frac{r_i}{\sqrt{x_i^2 + r_i^2}} \tag{2.10}$$

The elements x_i are the data as it enters the systolic array (Figure 2.1) and r_i are the elements of the upper triangular matrix $\hat{\mathbf{R}}$. In each row the leading element of $\mathbf{X}$ is eliminated and the remaining data vector is rotated through s and c, and passed down to the next row where the next leading element is eliminated. This is a iterative operation where the result of one row rotation is passed down to be rotated again by the next leading element of the input vector, as illustrated below:

$$\begin{bmatrix} Generate \\ Givens \\ Rotations \\ for \\ each \\ row \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \dots & x_k \\ r_{1,1} & r_{1,2} & \dots & r_{1,k} \\ 0 & \hat{x}_2 & \dots & \hat{x}_k \\ 0 & r_{2,2} & \dots & r_{2,k} \\ 0 & 0 & \dots & \hat{\hat{x}}_k \\ 0 & 0 & \dots & r_{k,k} \end{bmatrix} \Rightarrow \begin{bmatrix} \bar{r}_{1,1} & \bar{r}_{1,2} & \dots & \bar{r}_{1,k} \\ 0 & \bar{r}_{2,2} & \dots & \bar{r}_{2,k} \\ 0 & 0 & \dots & \bar{r}_{k,k} \end{bmatrix} \tag{2.11}$$

The sequence of rotations required to eliminate N data elements is given by the matrix $\hat{\mathbf{Q}}(n)$ and is the product of the individual rotations preformed

on each row: ($\hat{\mathbf{Q}}_k$ eliminates one element of the data vector $\mathbf{X(n)}$ and $\hat{\mathbf{Q}}(n)$ is the matrix that eliminated the vector $\mathbf{X(n)}$)

$$\hat{\mathbf{Q}}(n) = \hat{\mathbf{Q}}_1\hat{\mathbf{Q}}_2 \ldots \hat{\mathbf{Q}}_l \tag{2.12}$$

where k^{th} rotation is;

$$\hat{\mathbf{Q}}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & \ddots & 0 & 0 & \ldots & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 0 & c_k & \ddots & s_k \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & -s_k & \ldots & c_k \end{bmatrix} \tag{2.13}$$

If the 'accumulating' matrix $\hat{\mathbf{R}}$ is initialized to zero, the process described above can be used in a continuous mode to triangularize all the data as it enters the array and at any time the matrix $\hat{\mathbf{R}}$ represents the exact $\mathbf{QR}$ decomposition of all the data passed up to that time.

The triangular array shown in Figure 2.1 is composed of two parts, one which has two different PE structures and functions (Part 1) and a second that contains only one type (Part 2). All the cells communicate via nearest neighbor connections and the array meets the requirements for VLSI implementation, which are local communications and regularity. The first part with reference signal input accumulates the elements of the factored autocorrelation matrix $\hat{\mathbf{R}}$. The boundary cells (round) generate the parameters s and c and eliminate the entering data. The rotating elements s_k and c_k which were computed in the boundary cell on one clock

cycle will propagate horizontally to the next cell on the next clock cycle and so on, the propagation continues on successive clock cycles (without changing s_k and c_k). Because of the delay of one clock cycle per cell in propagation of the rotating parameters, it is necessary to impose a time skew on the input data to allow the data to be rotated by the corresponding s_k and c_k. The data flow and the cell computations are controlled by a global clock. The internal cells (square) rotate the remaining elements of the input vector $\mathbf{X}$ with the same parameters that forced the leading element of the vector to zero. The rotated vector $\hat{\mathbf{X}}$ which now has $N-1$ elements is passed down to the next row where another element of the vector is eliminated and the remaining elements are rotated through new s and c. This procedure continues until all the vector $\mathbf{X}$ has been eliminated and 'accumulated' on the matrix $\hat{\mathbf{R}}$. This structure is 100% efficient since for any rotated vector that moves down, a new data vector enters the now empty PE and interact with the previously stored elements of $\hat{\mathbf{R}}$ and perform the same computation as described before. The second part of the triangular array with the desired input consists only of internal cells which 'accumulate' the crosscorrelation factor (Shown as vector $\mathbf{U}$). As the input d which has been delayed N clock cycles in order to allow the right timing, moves down, it is being rotated by the same parameters s and c that eliminated the corresponding vector $\mathbf{X}$ and these rotations are accumulated on the vector $\mathbf{U}$. The operation of the boundary cells can be described in three steps:

1. Compute c

$$c = \frac{r(n-1)}{\sqrt{\lambda^2 r(n-1)^2 + {x_{in}}^2}} \tag{2.14}$$

2. Compute s

$$s = \frac{x_{in}}{\sqrt{\lambda^2 r(n-1)^2 + {x_{in}}^2}} \tag{2.15}$$

3. Update $r(n)$

$$r(n) = \sqrt{\lambda^2 r(n-1)^2 + {x_{in}}^2} \tag{2.16}$$

The operation of the internal cell (for Part 1 and Part 2, thus $r(n)$ replaces $u(n)$ for Part 2) can be summarized with two steps:

1. Rotate the incoming data (x or d) by s and c

$$x_{out} = cx_{in} - s\lambda r(n-1) \tag{2.17}$$

2. Update r(n) or u(n)

$$r(n) = c\lambda r(n-1) + sx_{in} \tag{2.18}$$

After $2N$ clock cycles the reference input vector $\mathbf{X}$ and the desired input d have been rotated through the same parameters and their accumulated factors in form of $\hat{\mathbf{R}}$ and $\mathbf{U}$ can be seen as the solution to normal equation. (For example when $N = 3$)

$$\begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ 0 & r_{2,2} & r_{2,3} \\ 0 & 0 & r_{3,3} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \tag{2.19}$$

Since the matrix in (2.19) is in a upper triangular form it does not need to be inverted in order to solve the normal equation as in section 1.4. Instead

the above system can be solved by performing back substitution with a systolic array proposed by Kung [Gentleman 81] and illustrated in figure 2.1. The weight values then can be assigned to a filter which will preform the actual filtering process.

2.3 Direct Filtering with the Systolic Array

In previous section we have seen that the triangular array in conjunction with the back substitute array and a filter can be used in adaptive noise cancelling. In many applications, however, including A.N.C. we are not interested in the weight solution itself but in the error which is the filtered signal. McWhirter [McWhirter 83] proposed an array that produces the error directly without the need for the back substitution array and the filter, this development simplifies significantly the circuitry needed for the filtering process. The array proposed by McWhirter is similar to Kung and Gentlemen's array as can be seen in Figure 2.2 with the exception of the bottom cell which is a simple multiplier and some delay elements denoted by dots. Considering also the weighting factor λ that now takes the form of the matrix Λ

$$\Lambda = \begin{bmatrix} \lambda^{n-1} & 0 & 0 & 0 \\ 0 & \lambda^{n-2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.20}$$

it can be shown how the error is derived directly from the array [Based on McWhirter 83]. This derivation is carried out in appendix A. From the

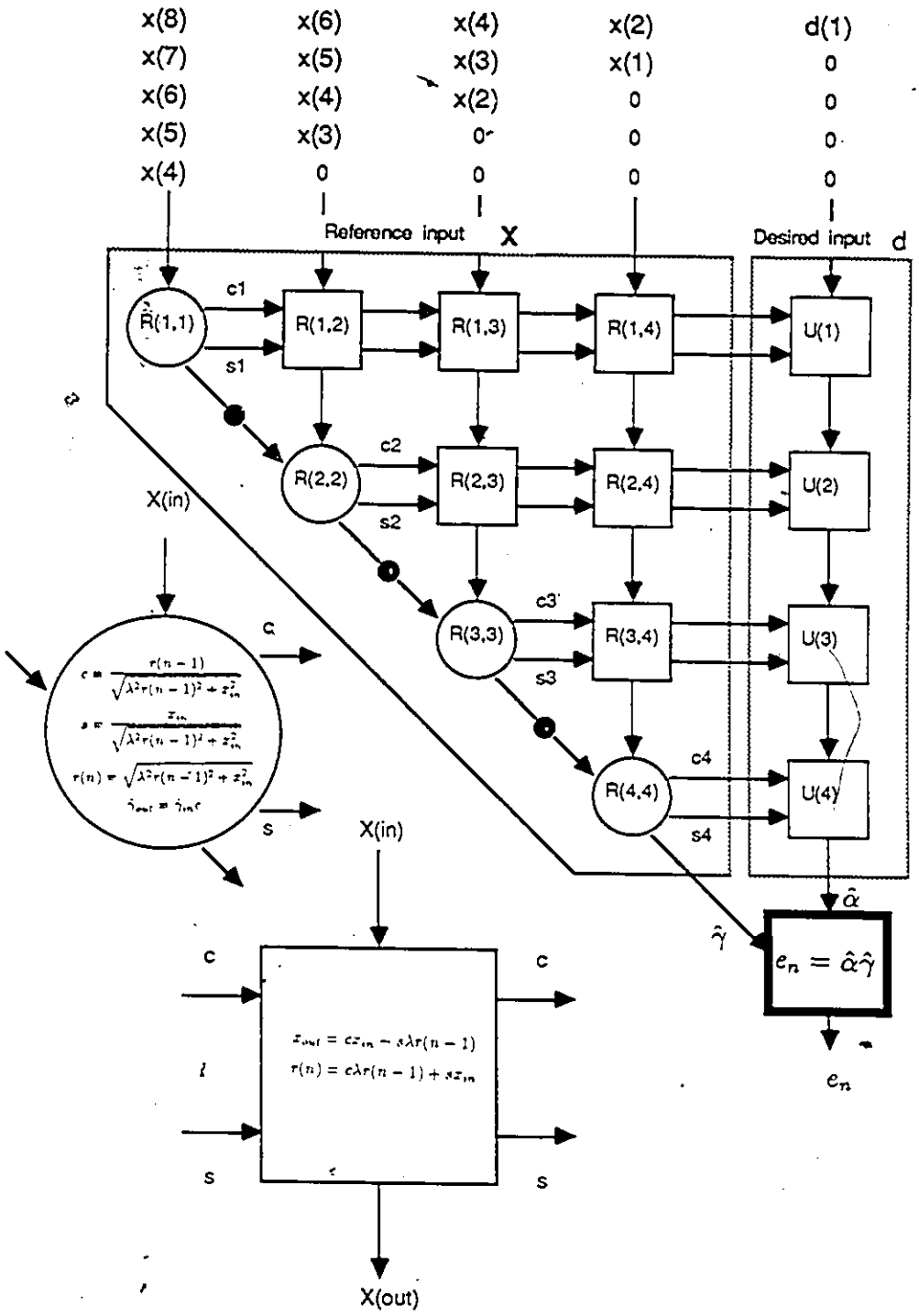


Figure 2.2: Direct Filtering with Systolic Array

proof in appendix A it follows that the last element or the current error $e(n)$ on the error vector $\mathbf{E}(n)$ is given by;

$$e(n) = \hat{\alpha}(n)\hat{\gamma}(n) \tag{2.21}$$

$\hat{\gamma}(n)$ is the product of the rotating cosine components, $\hat{\gamma}(n) = c_1 c_2 \dots c_N$ and $\hat{\alpha}(n)$ is the rotated desired input that emerges at the bottom of the array, or in other words $\hat{\alpha}(n)$ is the last element on $\mathbf{V}(n)$ which is the component that emerges from the right hand side of the array. As mentioned in appendix A some of the parameters that led to this result do not exist in the array, never the less, the final result 2.21 can be easily computed as shown in Figure 2.2.

Using the expression 2.21 we were able to test the systolic array for A.N.C.. The model used for the simulation is shown in Figure 2.3. The systolic array had 16 reference inputs thus a corresponding LMS filter with 16 taps was used for comparison. The desired input was composed of sine wave and white noise (Figures 2.4, 2.5), and the reference input was white noise (with variance=1). Whenever averaging was involved, it was done by running the algorithm for 500 times. Each time with new random sequence and different initial phase of the sine wave. The error squared defined by 2.21 is shown in (Figure 2.6). It is interesting to see the shape of the systolic array error compared to that of the LMS. It is low at $n = 1$ and then it increases with time. The reason for this lies in the initial conditions of the array that are set to zero, as the desired input reaches the bottom of the array after $2N$ clock cycles, it is multiplied through $\hat{\gamma}(n)$ that also increases with time as shown in Figure 2.7. Although the processed desired input

$\hat{\alpha}(n)$ may have a large value the product $\hat{\alpha}(n)\hat{\gamma}(n)$ will have the general shape of $\hat{\gamma}$. It is also interesting to see the curves of the error for different values of λ (Figure 2.10). From these plots it seems that the misadjustment is very large for $\lambda < 1$, however, as it will be shown in the next section, this misadjustment is not only due to noise in the weight solution but also due to scaling that occurs in the array. The filtered output (sine wave) is presented in Figures 2.8, 2.9.

2.4 Rescaled Filtering with Systolic Arrays

In Figure 2.10 we saw the effect of λ on the performance of the array. These curves look quite awkward due to two reasons; {1}.The self noise generated by the array appears to be very large (much larger than for RLS or LMS) for $\lambda < 1$, {2}. The self noise seems to affect the signal power itself, thus the signal power in the filtered output is smaller then the signal power in the desired input. Or in other words we have some signal cancellation along with noise cancellation. This result has to be explained and as we will see at the end of this section it is due to scaling that occurs in the array. The expression 2.21 was derived by 'close examination' of the matrix $\hat{\mathbf{Q}}(n)$ which was partitioned to its building components. However, since $e(n)$ is the last element on the error vector $\mathbf{E}(n)$ it must correspond to the a posteriori error as in (1.36). To prove this we must reexamine (A.18) which, after carrying out the multiplication can be written as;

$$\lambda \mathbf{G}(n)\hat{\mathbf{R}}(n-1) + \mathbf{g}(n)\mathbf{X}^T(n) = \hat{\mathbf{R}}(n) \qquad (2.22)$$

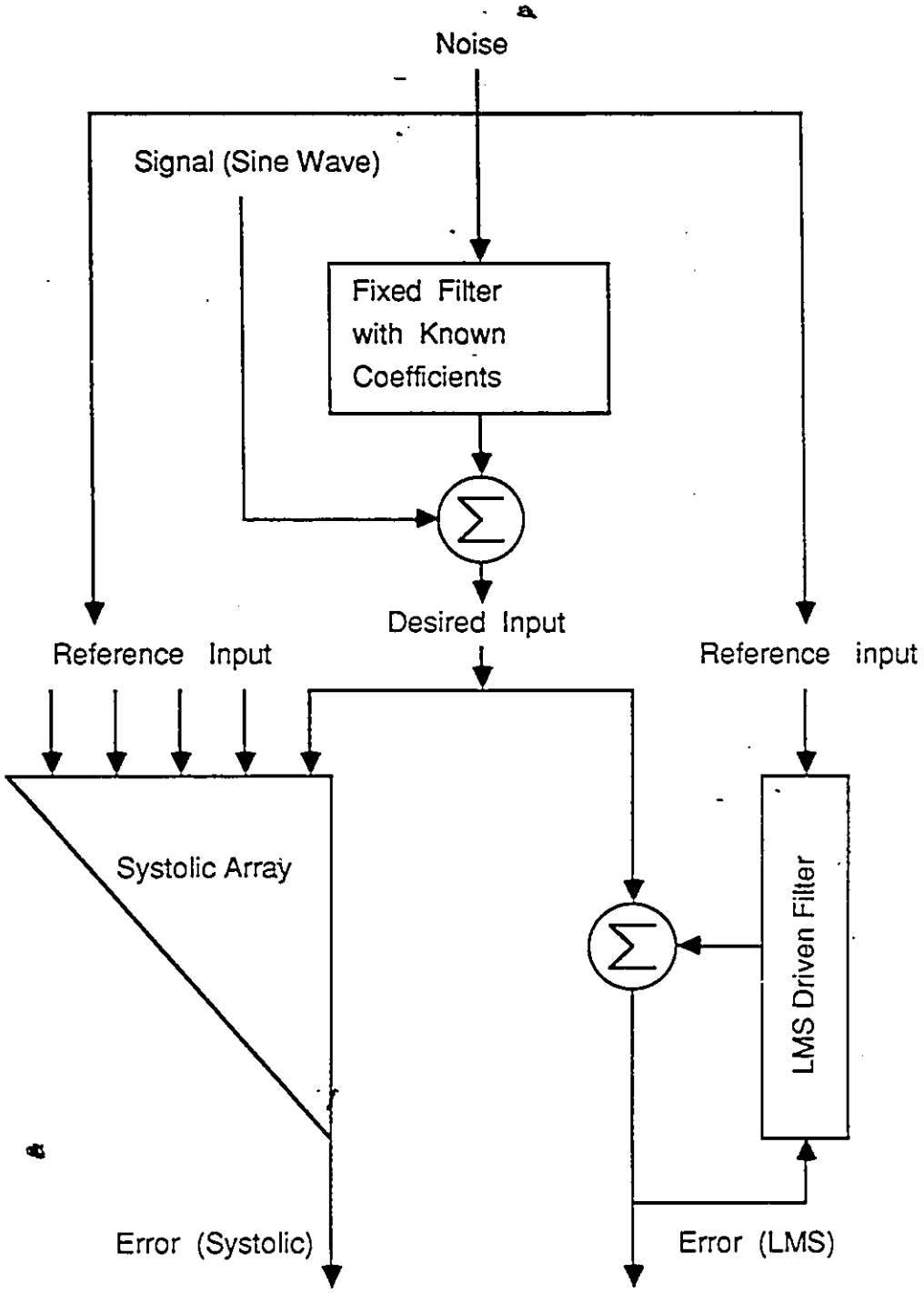


Figure 2.3: Model Used in Simulations

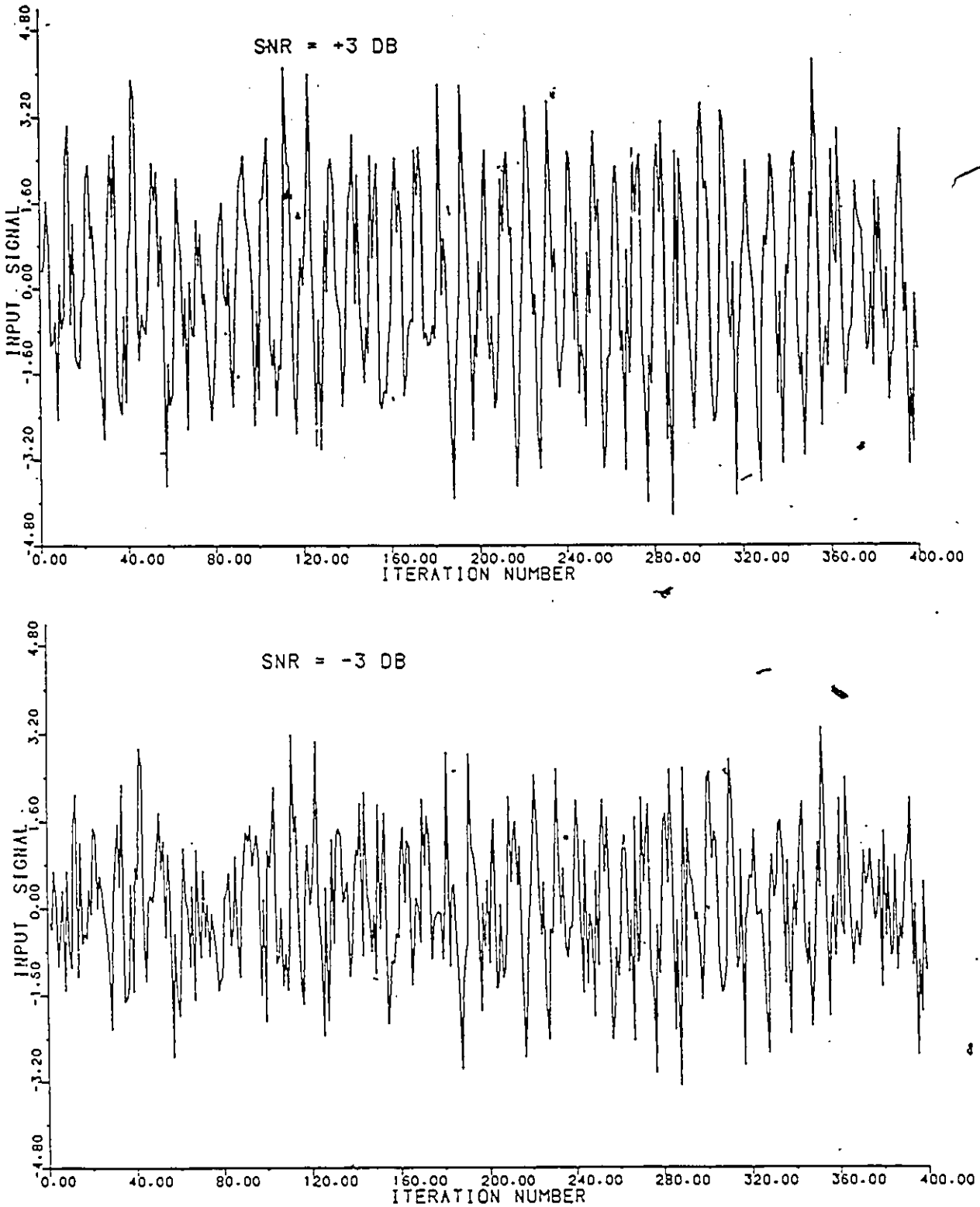
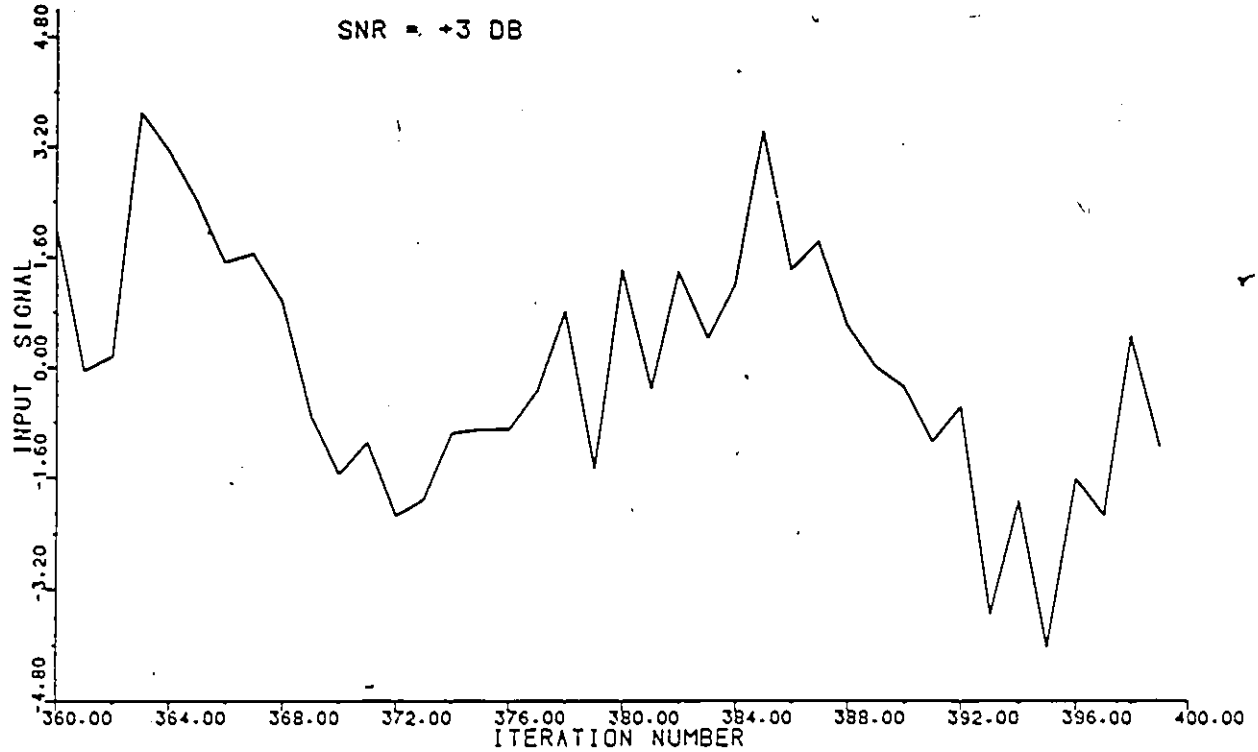


Figure 2.4: Input Signal



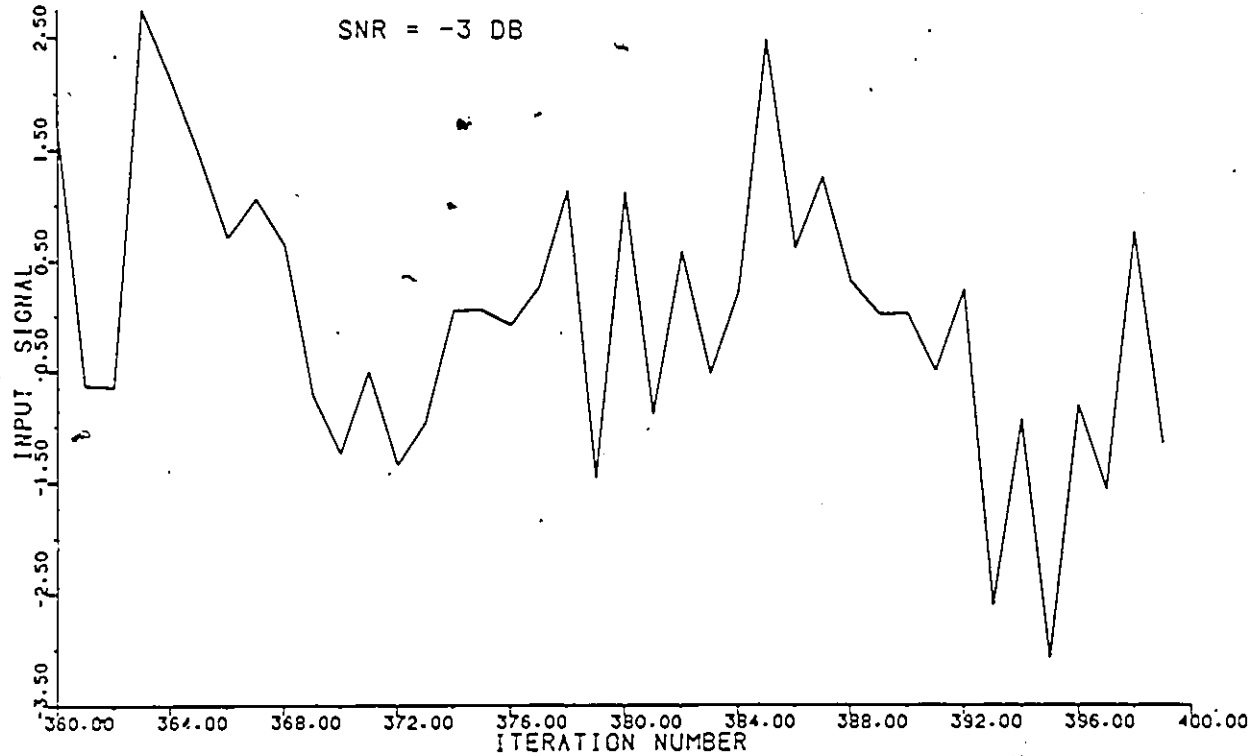


Figure 2.5: Segment of Input Signal

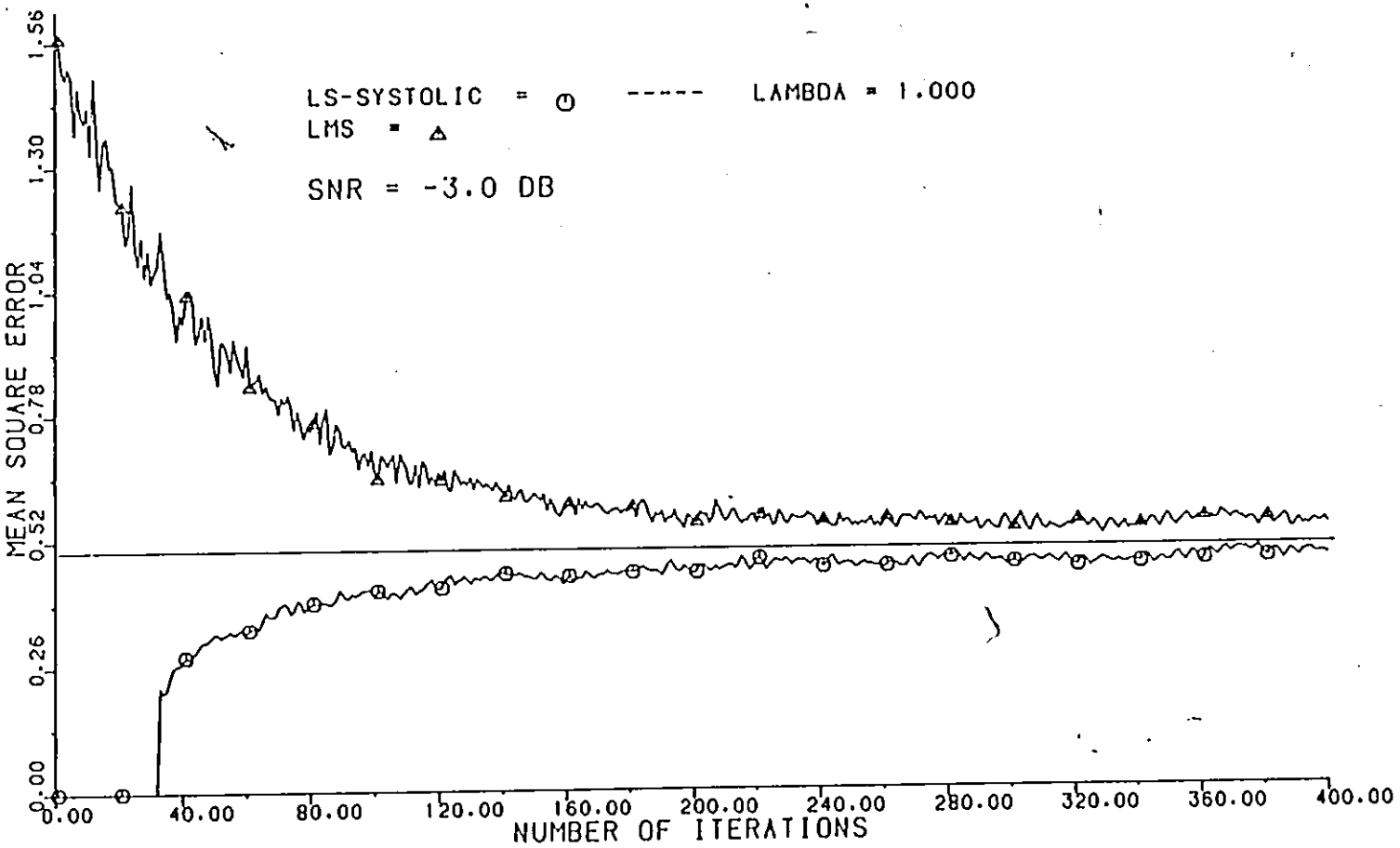


Figure 2.6: Systolic Output Compared with LMS ($N = 16$)

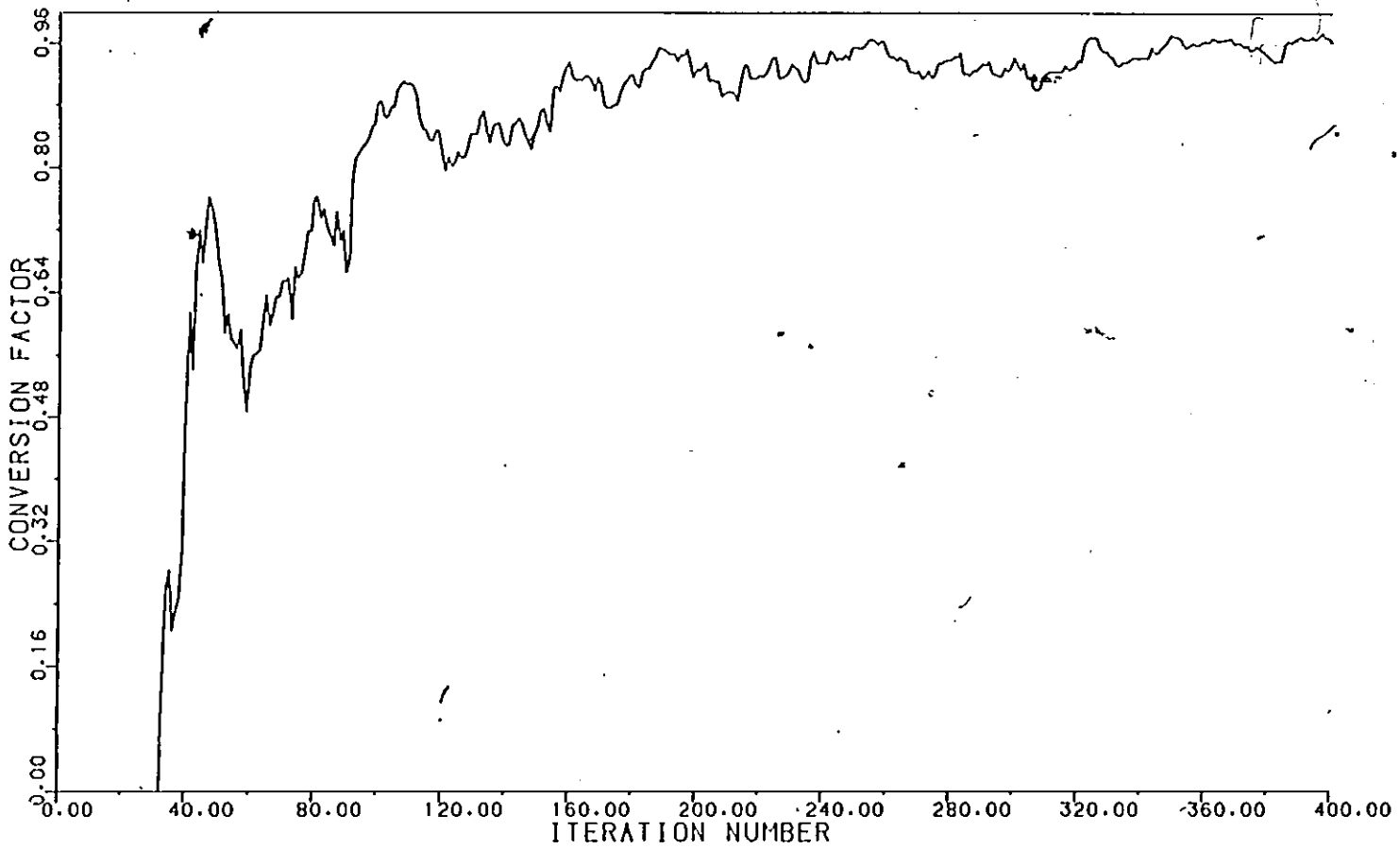
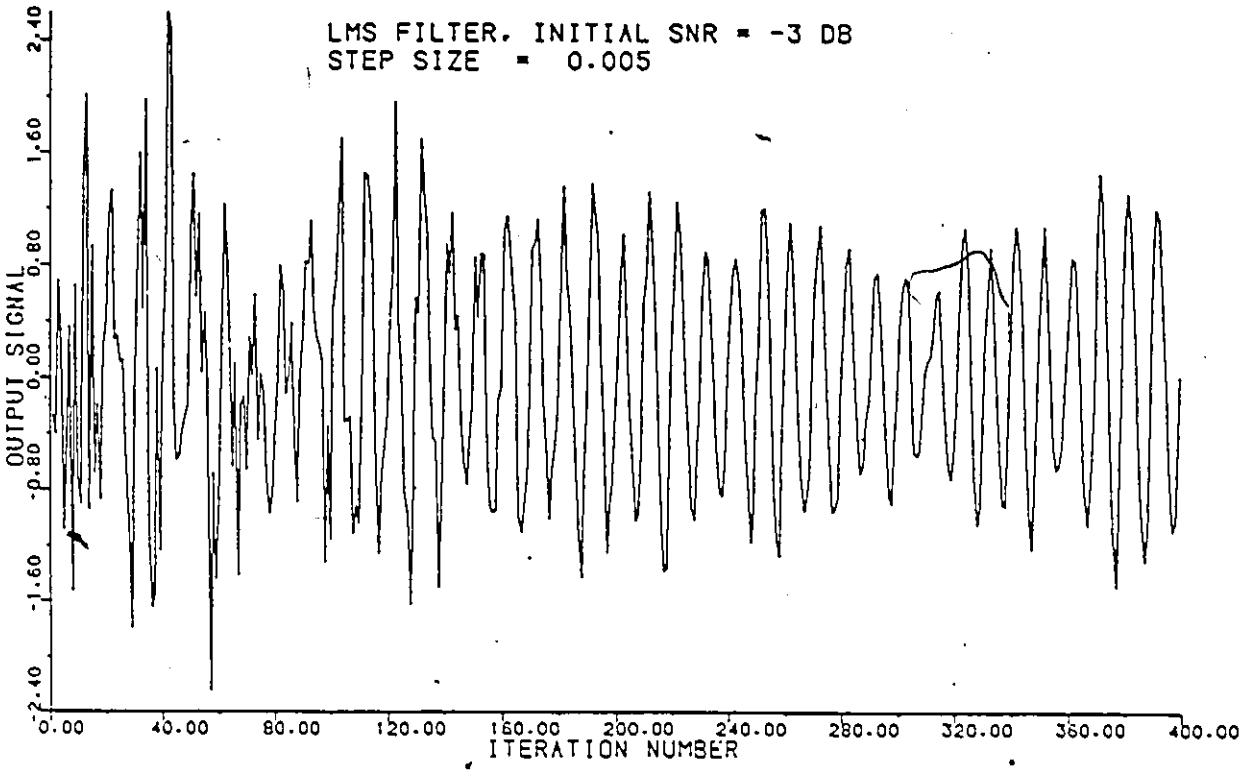


Figure 2.7: The Conversion Factor $\hat{\gamma}(n)^2$



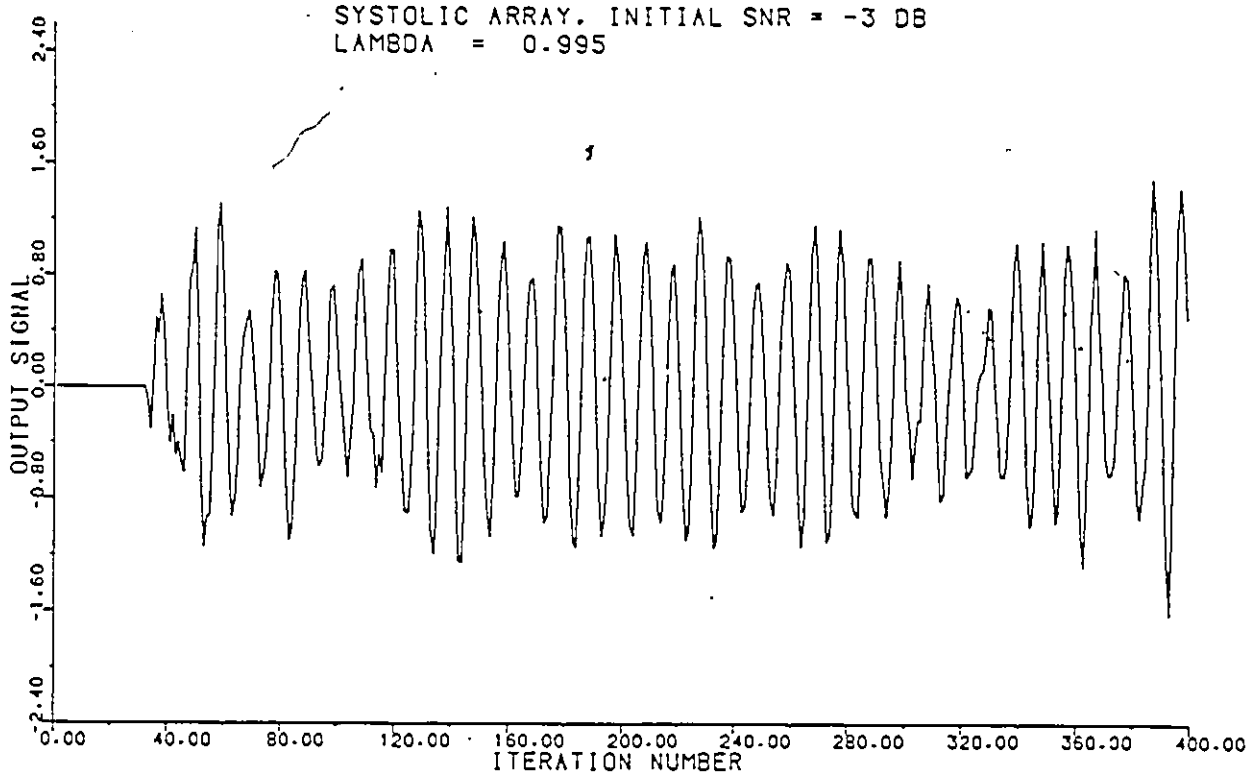
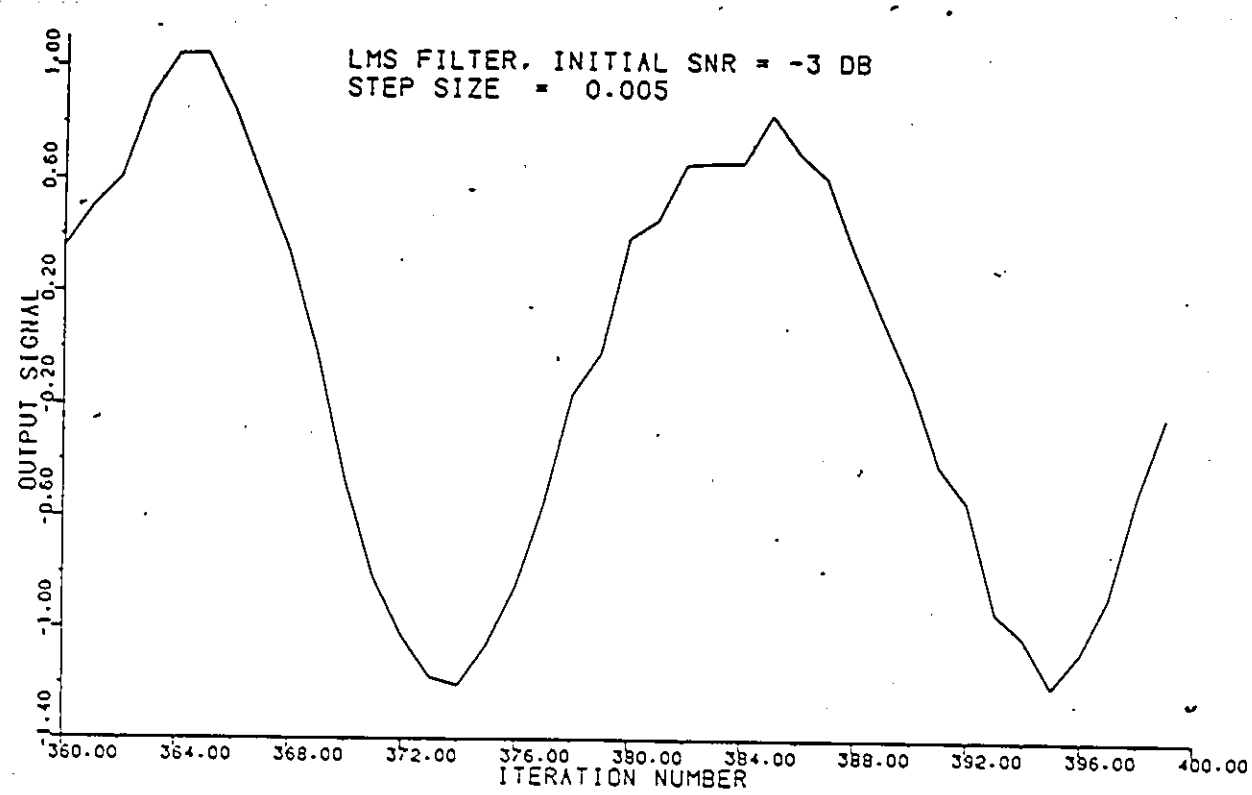


Figure 2.8: Filtered Output for LMS and Systolic Array



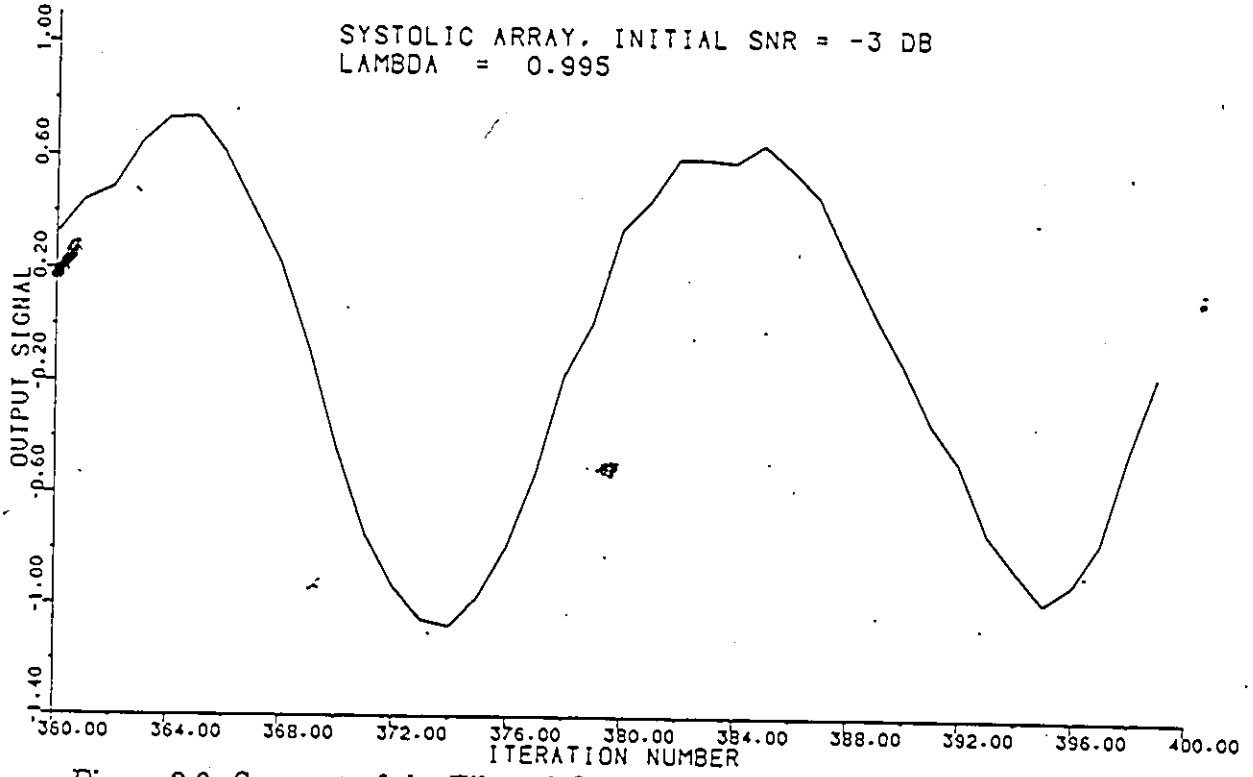


Figure 2.9: Segment of the Filtered Output for LMS and Systolic Array

$$\lambda \mathbf{d}^T(n)\hat{\mathbf{R}}(n-1) + \hat{\gamma}\mathbf{X}^T(n) = 0 \tag{2.23}$$

now combining with (A.21) gives;

$$\hat{\alpha}(n) = \hat{\gamma}(n)\,[d(n) - \mathbf{X}^T(n)\mathbf{W}(n-1)] \tag{2.24}$$

Clearly from this expression we can make the connection to the theory developed in section 1.3 where the term in the square brackets in 2.24 was defined as the a priori error. To obtain the a priori error from the triangular array we must divide the output $\hat{\alpha}(n)$ that emerges at the bottom of the array by $\hat{\gamma}(n)$ $\alpha(n) = \frac{\hat{\alpha}(n)}{\hat{\gamma}(n)}$. As discussed before the a priori error has the general shape of the LMS learning curve which can be seen at Figure 2.11. It also converges faster than the LMS algorithm with smaller misadjustment but, as will be discussed in the next section the misadjustment is not consistent with the misadjustment equations developed for the RLS algorithm. We have identified the a priori error in 2.21 as $\frac{\hat{\alpha}(n)}{\hat{\gamma}(n)}$, does that imply that $\hat{\gamma}^2(n)$ is the conversion factor just as in (1.40) ?. One answer could be supplied just by examining the error in terms of a priori error that can be expressed as:

$$e(n) = \hat{\gamma}^2(n)\,[d(n) - \mathbf{X}^T(n)\mathbf{W}(n-1)] \tag{2.25}$$

in this expression we have related the a priori error to the a posteriori error through $\hat{\gamma}^2(n)$ which implies that $\hat{\gamma}^2(n)$ is equal to $\gamma(n)$ defined by (1.41). This answer can be proven to be incomplete by looking closely at the building components of $\hat{\gamma}(n)$ which are the cosine elements of $\hat{\mathbf{Q}}(n)$. Each cosine element is generated in the boundary cell by computing the

ratio between the stored $\hat{R}(n-1)$ to the newly computed $\hat{R}(n)$.

$$c = \frac{\lambda \hat{R}(n-1)}{\hat{R}(n)} \tag{2.26}$$

The stored components $\hat{R}$ in the boundary cells are also the eigenvalues of the matrix $\hat{\mathbf{R}}$ since the matrix is in upper triangular form. Denoting the eigenvalues by β and using the fact that,

$$det[matrix] = \beta_1 \beta_2 \ldots \beta_n \tag{2.27}$$

we can rewrite $\hat{\gamma}(n)$ as;

$$\hat{\gamma}(n) = c_1 c_2 \ldots c_n = \frac{\lambda\beta_1(n-1)\lambda\beta_2(n-1)\ldots\lambda\beta_n(n-1)}{\beta_1(n)\beta_2(n)\ldots\beta_n(n)} \tag{2.28}$$

or;

$$\hat{\gamma}(n) = \frac{\lambda^N det[\hat{\mathbf{R}}(n-1)]}{det[\hat{\mathbf{R}}(n)]} \tag{2.29}$$

Plots of averaged $\hat{\gamma}(n)$ are shown in Figure 2.12 and it can be seen that when the process converges $\hat{\gamma}(n)$ converges to λ^N (where N is the number of boundary cells or the number of taps in an equivalent transversal filter) as expected. Since $\mathcal{R}(n) = \hat{\mathbf{R}}^T(n)\hat{\mathbf{R}}(n)$ the eigenvalues squared of $\hat{\mathbf{R}}(n)$ must be equal to the eigenvalues of $\mathcal{R}(n)$.

$$\hat{\gamma}^2(n) = \frac{\lambda^{2N} det[\mathcal{R}(n-1)]}{det[\mathcal{R}(n)]} \tag{2.30}$$

Plots of $\hat{\gamma}^2(n)$ are shown in Figure 2.13. Except for the scaling factor λ^{2N} $\hat{\gamma}^2(n)$ has the same structure as the conversion factor defined by (1.40), thus we can draw the conclusion that the error defined by 2.21 is the same error as in (1.42). The need for rescaling was first mentioned in section 1.3

where in order to achieve the correct output we had to rescale with λ. In the systolic array case the scaling is due to λ^{2N} as shown before in Figure 2.10. Thus in order to achieve the right output we have to rescale by the same amount. The rescaled a posteriori error is shown in Figure 2.14. The mean square error converges in this figure to a value greater then 0.5 for $\lambda < 1$. This is in full agreement with the theory since the sine wave representing the minimum mean square value has the energy of 0.5 and any value above 0.5 will represent the self noise (excess mean square error) generated by the filter with finite memory length. The misadjustment in these curves is only due to noise and is not quite consistent with the theoretical values dictated by expressions (1.37) and (1.38) developed for the RLS algorithm. It must be noted that the QR algorithm realized in systolic array form produces the same parameters as the RLS algorithm (a posteriori error, a priori error and conversion factor).

2.5 The Misadjustment in the Array

As pointed out in previous section and shown by simulations, the misadjustment in the systolic array is not consistent with the theoretical values. The misadjustment in the simulations is twice as large as the theoretical one. The reason for that lies in the decomposition itself as the weighted normal equation for the RLS algorithm is $\Lambda\bar{\mathbf{X}}^T(n)\bar{\mathbf{X}}(n)\mathbf{W} = \Lambda\bar{\mathbf{X}}\bar{\mathbf{d}}$ by using the $\mathbf{QR}$ decomposition with the same weighting matrix in the form $\mathbf{Q}(n)\Lambda\mathbf{E}(n) = \mathbf{Q}(n)\Lambda\bar{d} - \mathbf{Q}(n)\Lambda\bar{\mathbf{X}}(n)\mathbf{W}(n)$ we can rewrite the normal equations in terms of the decomposed components as, $\Lambda^2\mathbf{R}^T(n)\mathbf{R}(n)\mathbf{W} = \Lambda^2\mathbf{R}(n)\mathbf{Q}^T(n)\bar{d}$. Keep-

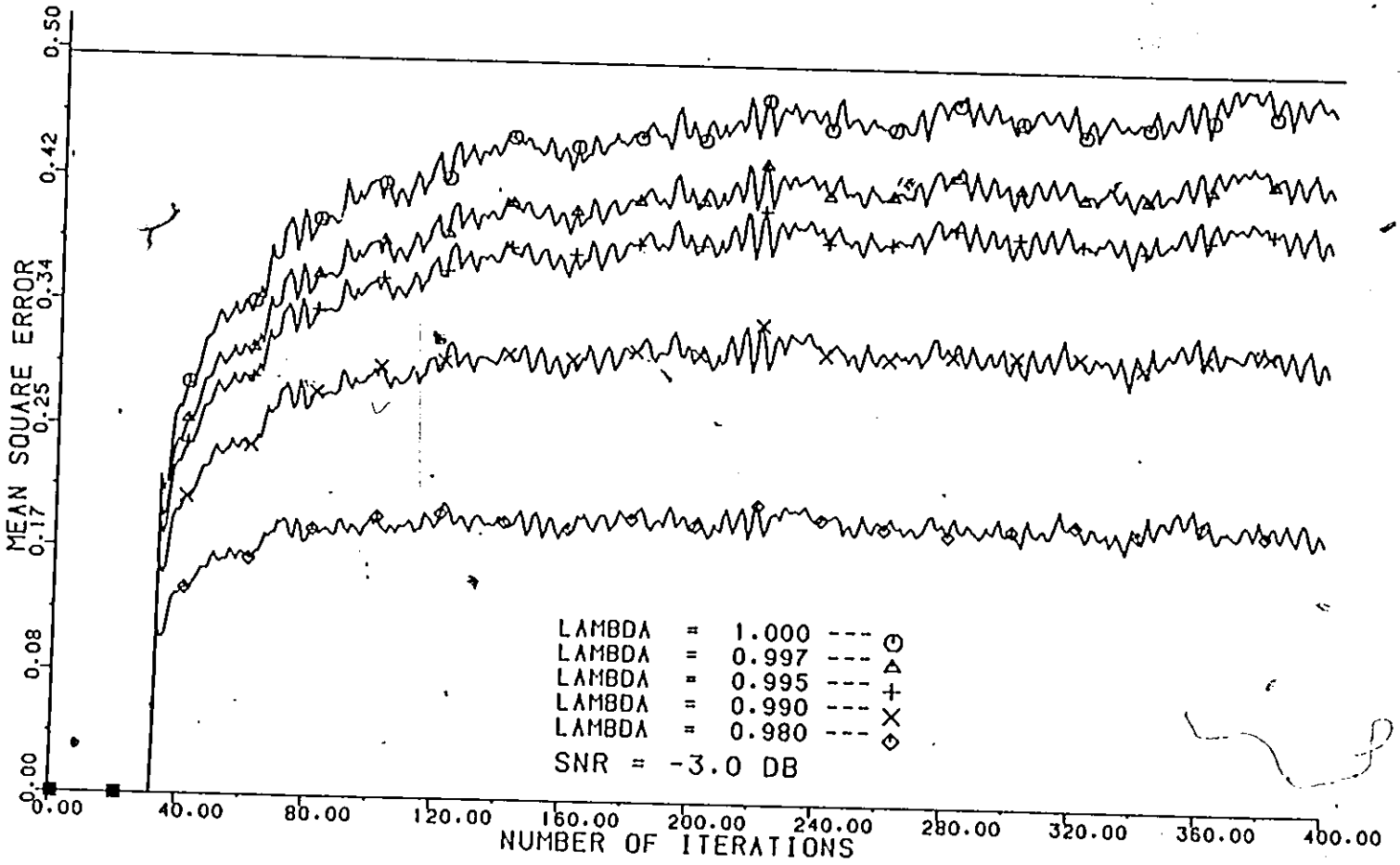


Figure 2.10: A posteriori error for different λ ($N = 16$)

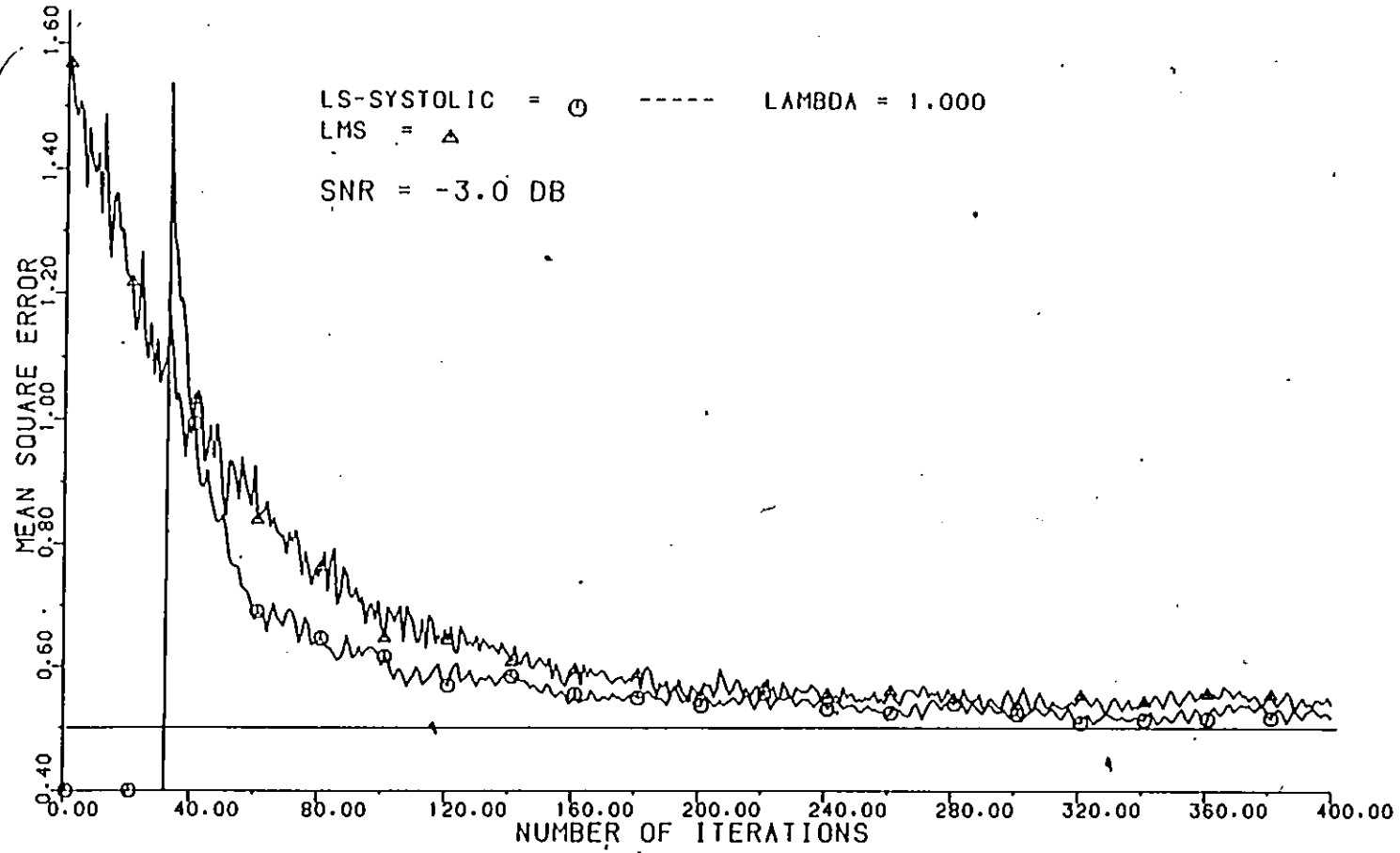


Figure 2.11: A priori error compared to LMS ($N = 16$)

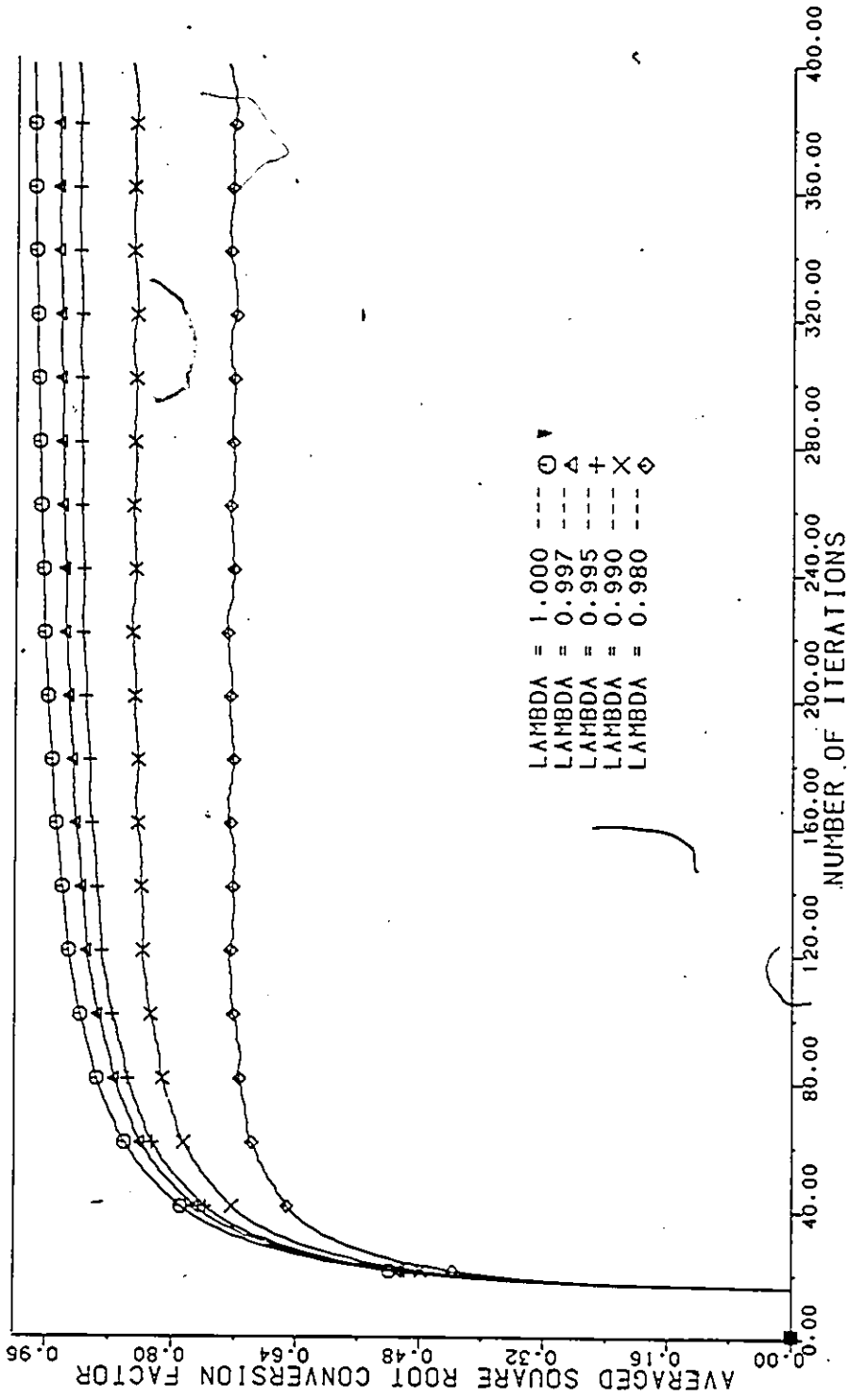


Figure 2.12: Averaged $\hat{\gamma}(n)$

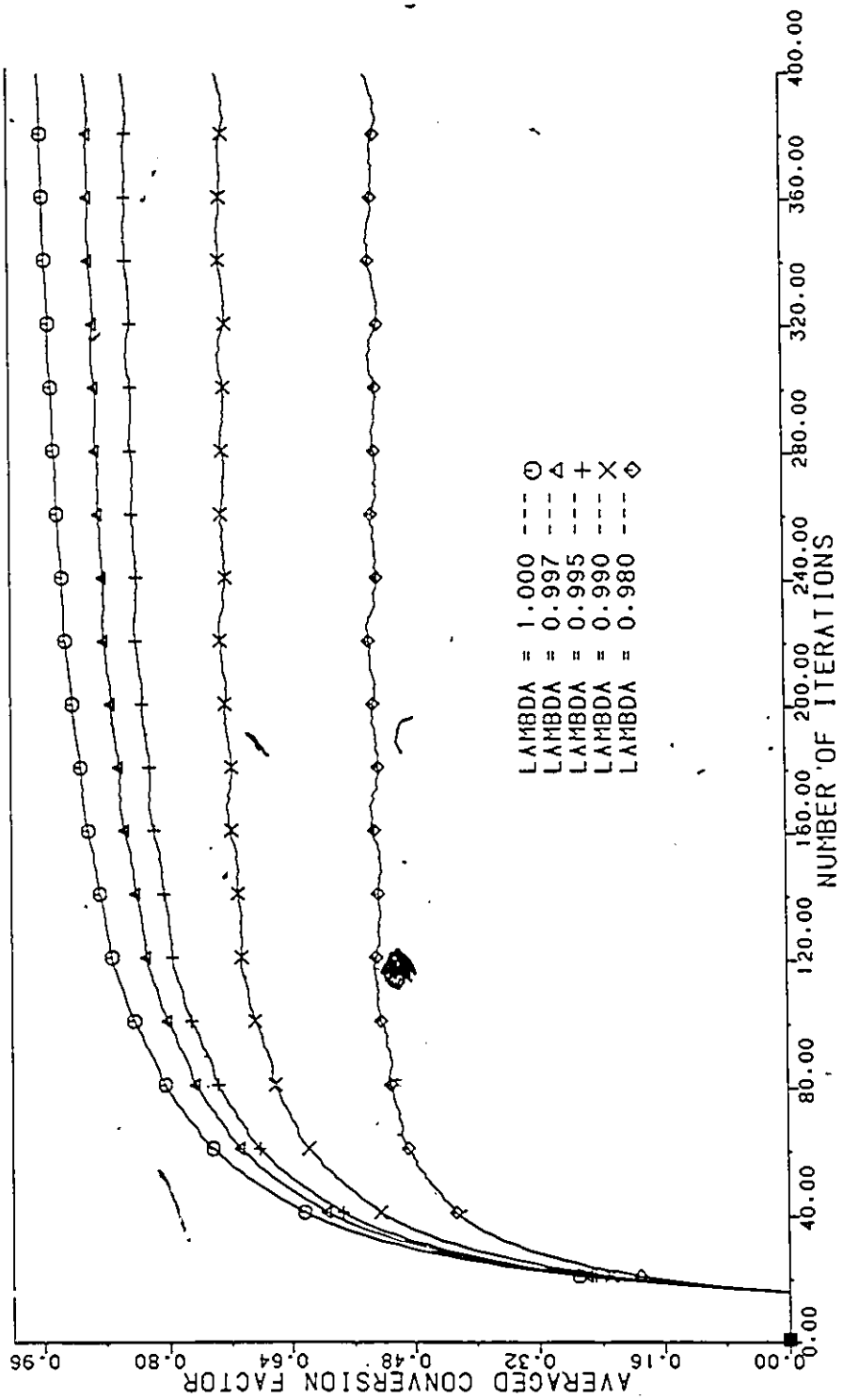


Figure 2.13: Averaged $\hat{\gamma}^2(n)$

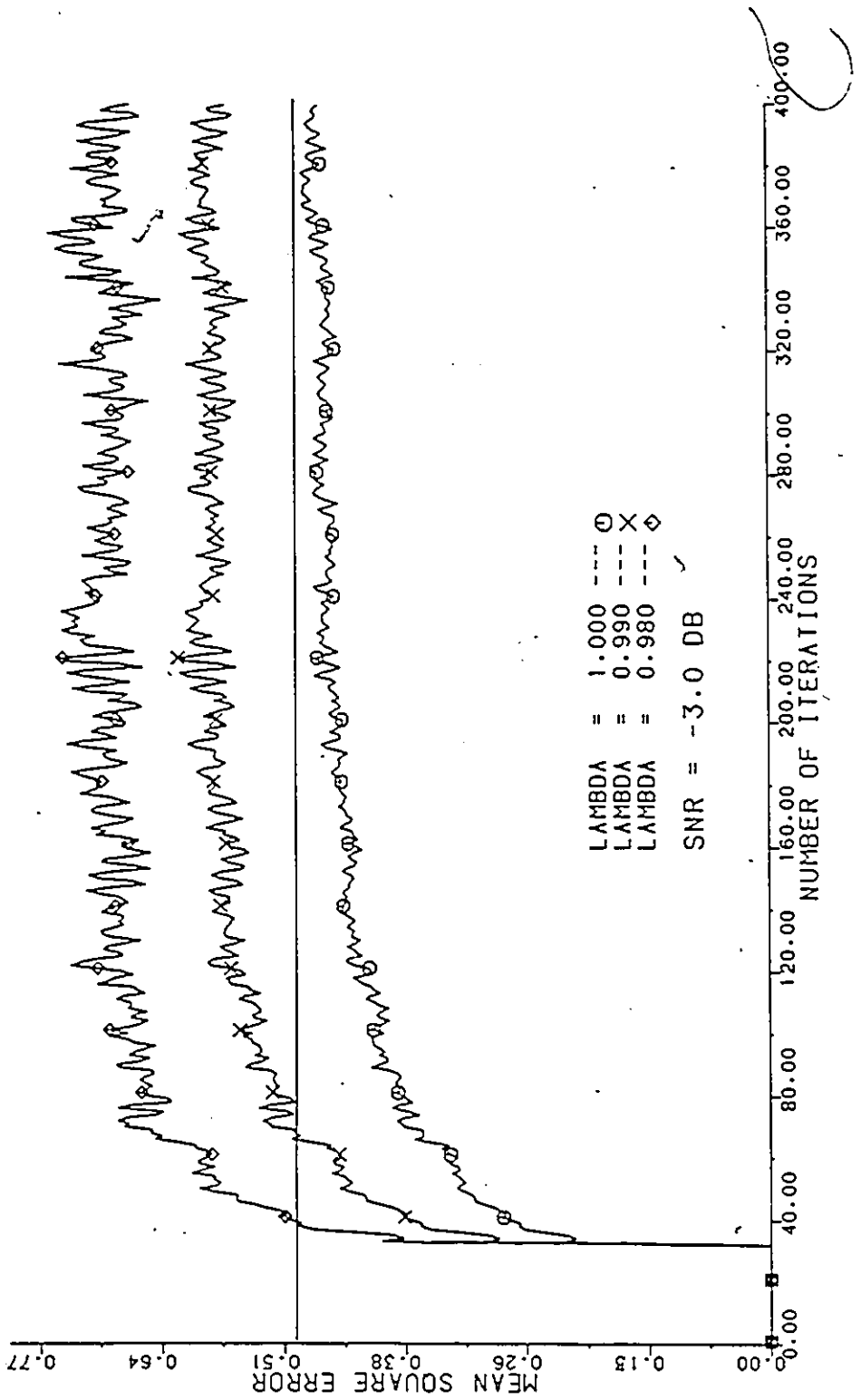


Figure 2.14: Rescaled a posteriori error ($N = 16$)

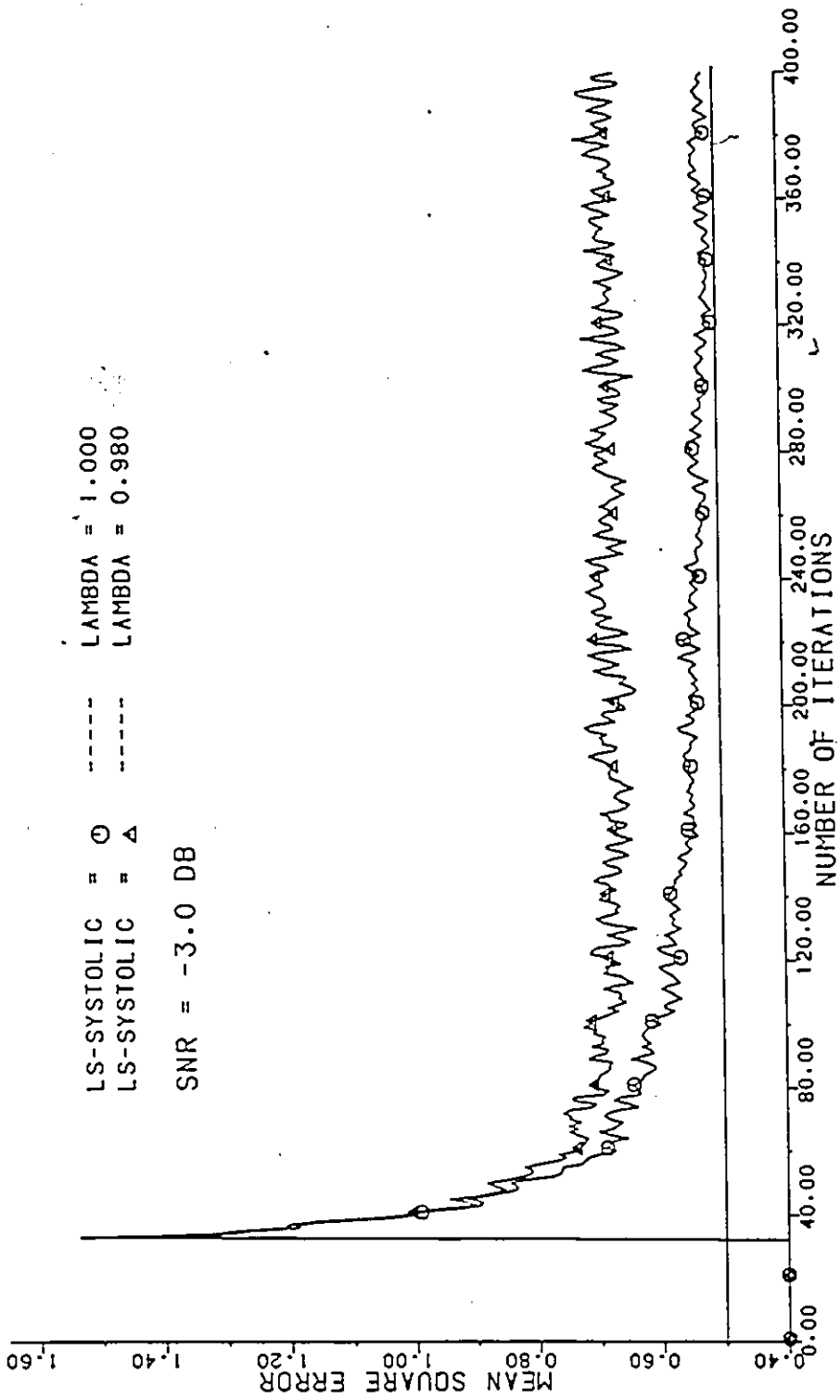


Figure 2.15: A priori error for different λ ($N = 16$)

ing in mind that except for the weighting matrix (assume for a moment that $\lambda = 1$ then Λ becomes an identity matrix) both normal equations are equivalent to each other and thus we can replace one expression with another, the normal equation developed through the use of **QR** decomposition can be written as $\Lambda^2\bar{\mathbf{X}}^T(n)\bar{\mathbf{X}}(n)\mathbf{W} = \Lambda^2\bar{\mathbf{X}}(n)\bar{d}$. From here we can see that by the use of the **QR** decomposition our weighting matrix became Λ^2. That implies larger misadjustment as we are using smaller λ then originally intended but also faster convergence. Using the same analysis as in Ling and Proakis [Ling 84] it is possible to develop a new expression (similar to (1.38)) for the misadjustment of the systolic array which now becomes;

$$\mathcal{M}_{sys}(\infty) = \frac{1-\lambda^2}{1+\lambda^2}N \tag{2.31}$$

or assuming that λ is very close to 1 this expression simplifies to;

$$\mathcal{M}_{sys}(\infty) = (1-\lambda)N \tag{2.32}$$

It follows under the assumption $1 + \lambda \approx 2$ that the misadjustment of the QR decomposition with systolic array is twice as large as the misadjustment of a RLS algorithm (1.38). It also follows that for equal eigenvalue spread when the step size in LMS algorithm is equivalent to $1 - \lambda$ of a systolic array both algorithms will produce the same misadjustment. These theoretical results are confirmed through simulation shown in Figures 2.16 and 2.17 for the step size corresponding to $1 - \lambda$, and also the true speed of convergence is shown in Figure 2.18 when both algorithms produce the same misadjustment. It is possible to achieve the same speed of convergence for both algorithms by adjusting the step size or in an equivalent form

increasing the signal power input without modifying the step size, however, then the misadjustment in the LMS algorithm is larger then the systolic array as shown in Figure 2.19. The parameter λ has a tremendous effect on the performance of the systolic array or the RLS algorithm. For small values of λ the convergence rate is faster with larger misadjustment whereas when λ approaches 1.0 the algorithm slows down but gives better performance. The curve in Figure 2.15 shows it quite clearly. This can be explained by the short memory in the algorithm dictated by λ, in that case the system forgets quickly the first estimates and assigns more weight to the newly computed ones which are more accurate. This approach also produces more fluctuations in the weight solution since it will be affected by even small changes in the environment and as a result larger misadjustment will be achieved. So far we have evaluated the QR decomposition versus the RLS algorithm and we discovered that some differences do exist in performance. One main difference is that the QR decomposition with systolic array will produce twice as much self noise as the RLS algorithm in systolic structure for the same value of λ. Another difference is also in the speed of convergence. From the theory we know that the RLS algorithm should converge in 2N iterations, in the QR decomposition (convergence occurs when 50 percent of the noise power has been cancelled, can be also be estimated from (1.37)) with systolic arrays, however, this is not true since the initialization procedure requires 2N iteration and only after that the convergence could start which takes also 2N iterations. Thus we can say with support of the simulations that the systolic array initially converges in 4N iterations. This result is somewhat disappointing, and although the

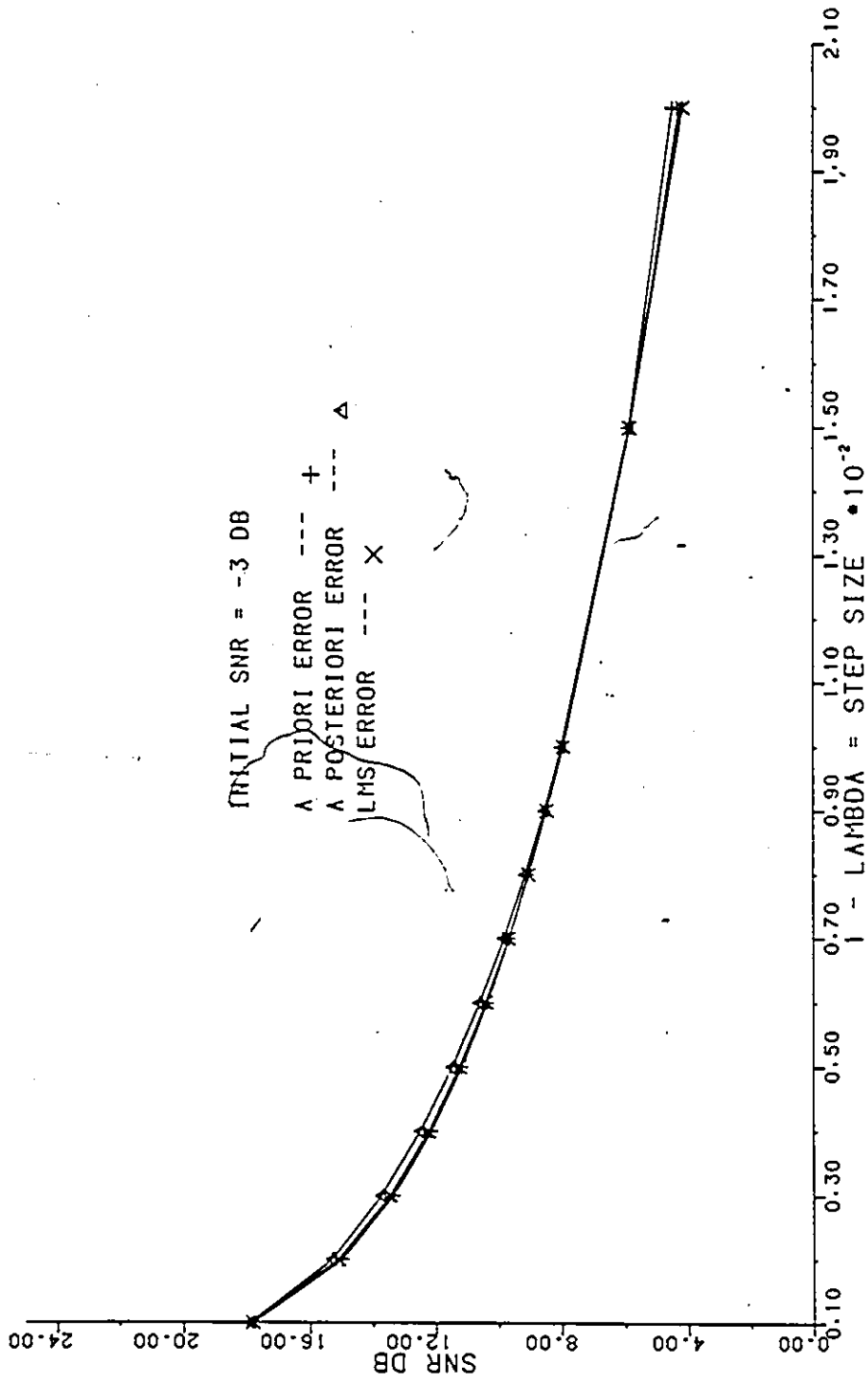


Figure 2.16: Plots of SNR of the LMS and the Systolic Array

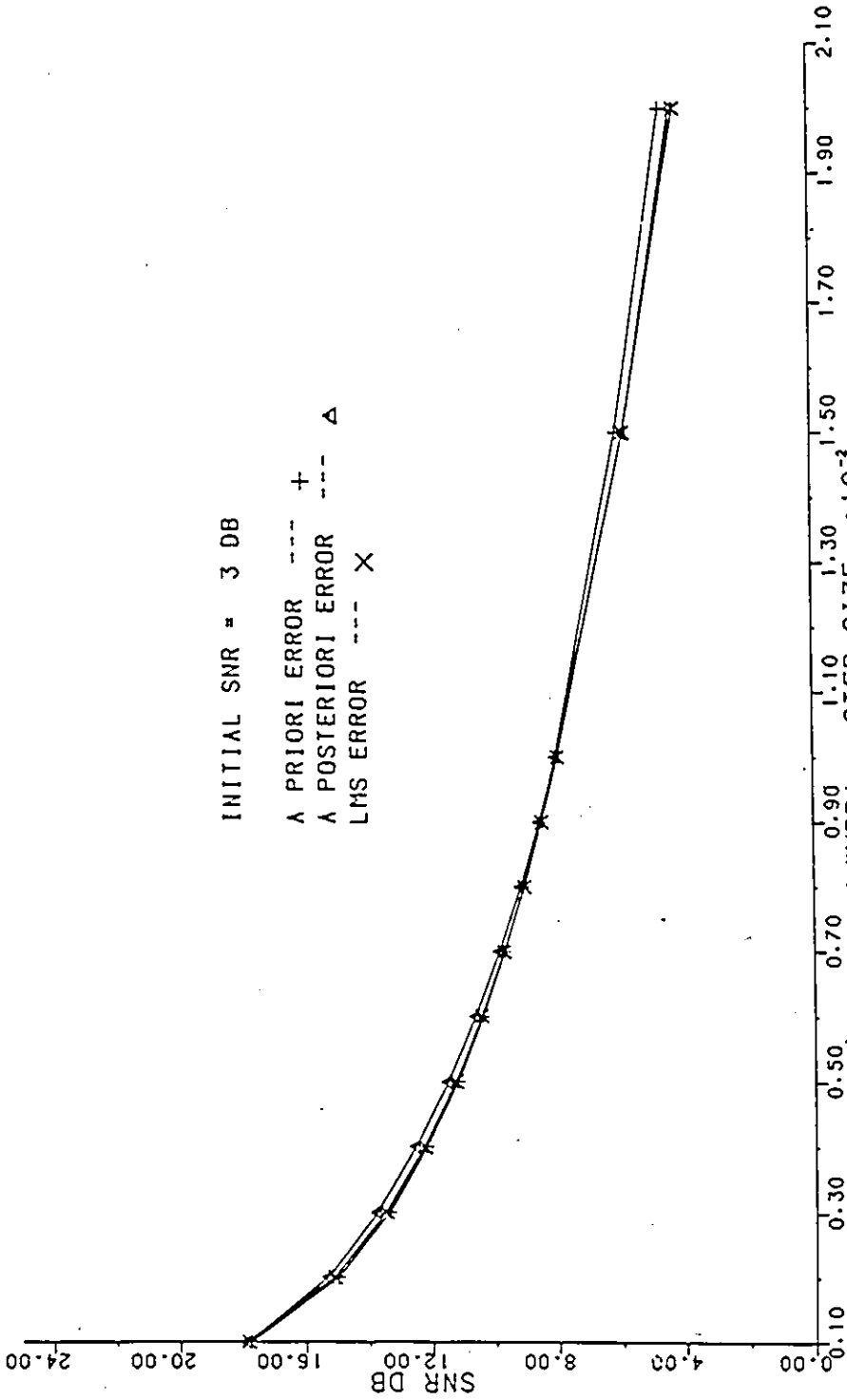


Figure 2.17: Plots of SNR of the LMS and the Systolic Array

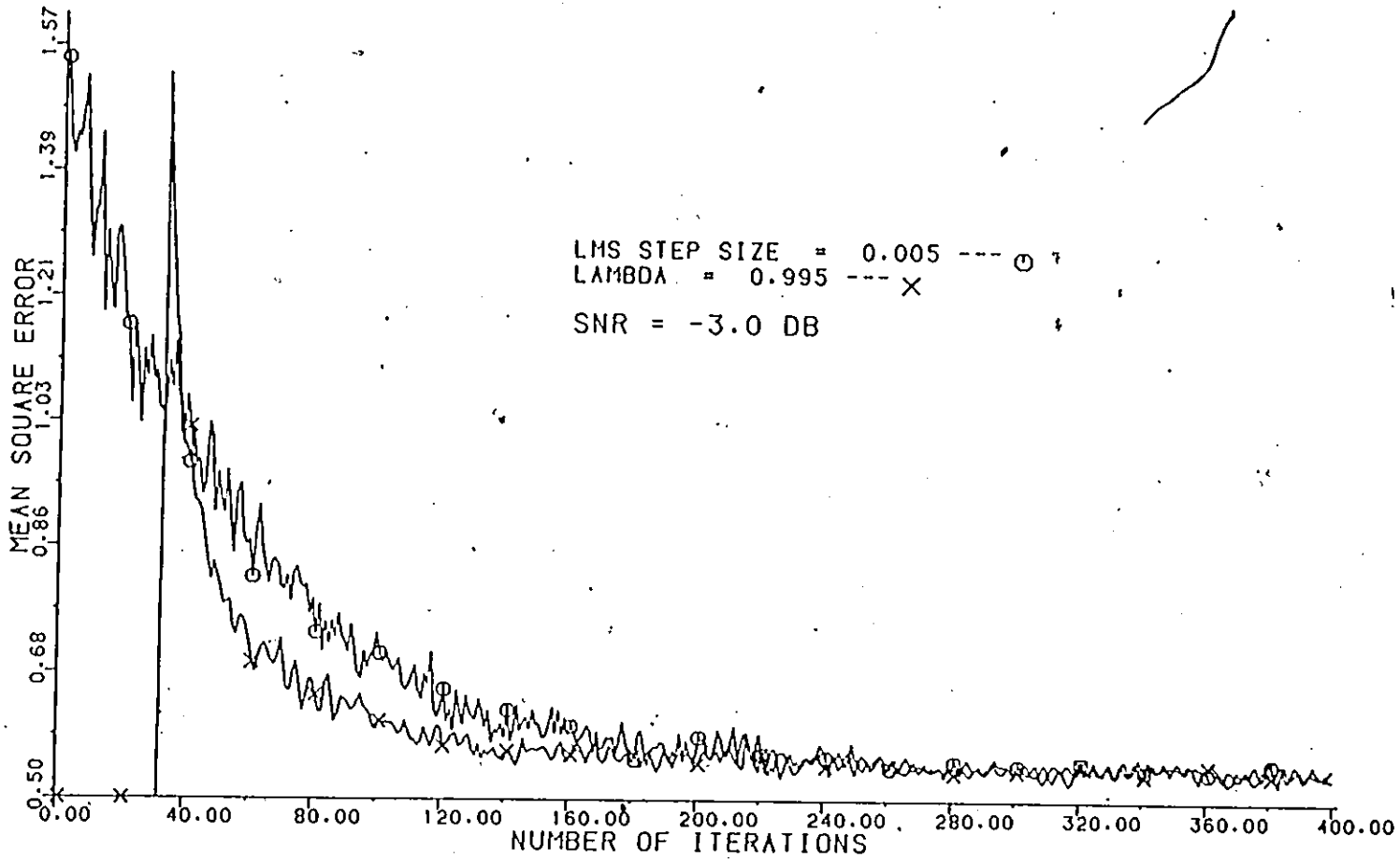


Figure 2.18: Mean Square Error plots with equivalent misadjustment for the LMS and the Systolic Array ($N = 16$)

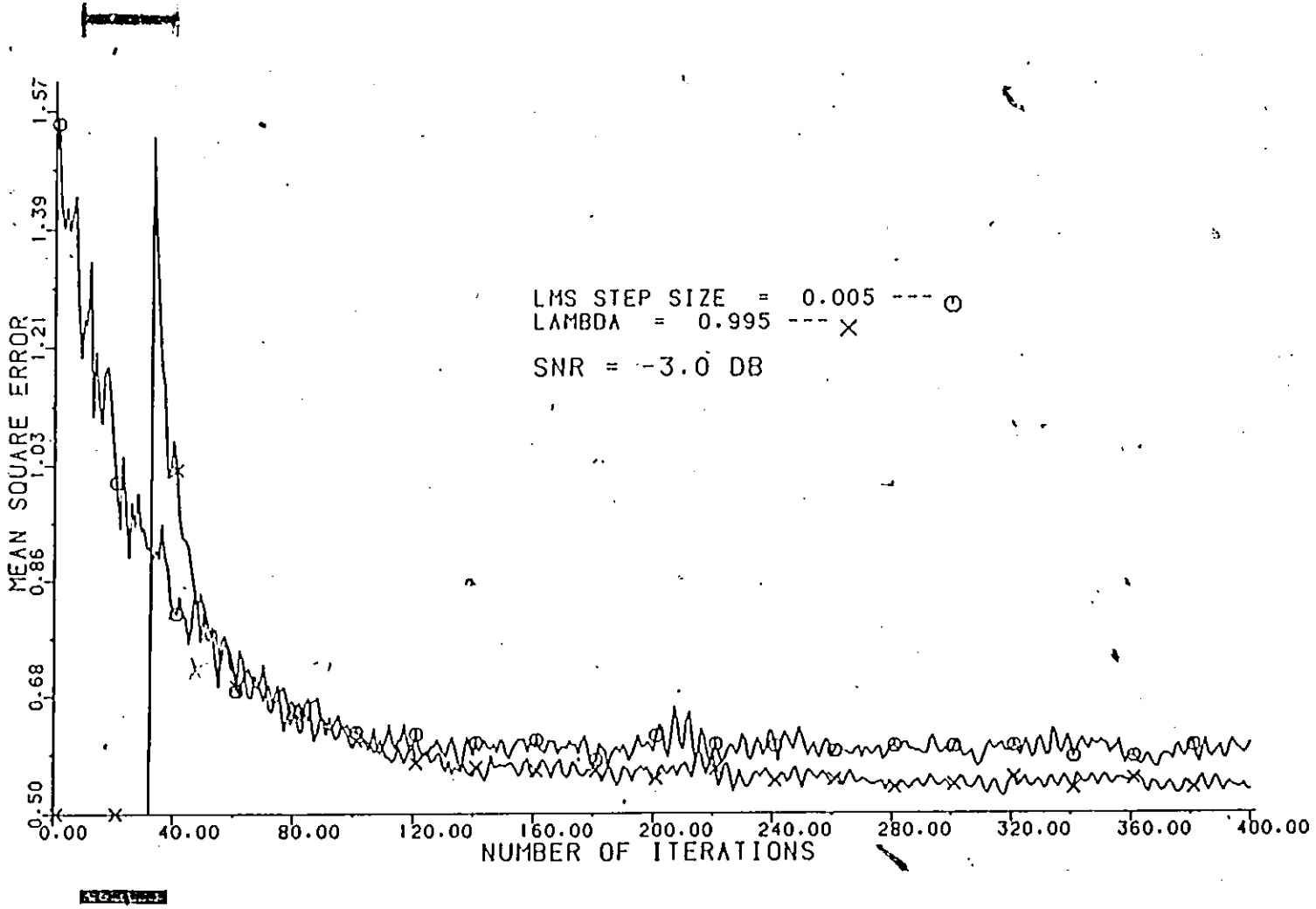


Figure 2.19: Mean Square Error plots with equivalent speed of convergence for the LMS and the Systolic Array ($N = 16$)

systolic array is still faster then the LMS algorithm, the advantage in the speed is not big. It must be clear, however, that the 4N convergence occurs only in the initial period, during tracking the system will converge in 2N to any changes in the environment. Also for white noise input the LMS algorithm can achieve the same rate of convergence at the expense of SNR (larger self noise).

Chapter 3

Hardware Structure

3.1 Realization of the QR decomposition array

In attempting to implement the QR decomposition one must first choose to realize in hardware or software. One attempt has been made by D.S. Broomhead [Broomhead at all 85] to implement the triangular systolic array (Figure 2.2) in software (programming) by using the INMOS transputer chip. Each cell in the array was represented by one transputer chip and the array was programmed in OCCAM a language developed by INMOS LTD. This type of implementation resulted in quite complex architecture since the equivalent of an Nth order filter will require $\frac{1}{2}[N^2 + 3N]$ processing cells or in this case transputer chips thus for a 16 tap filter it is necessary to use 152 chips. This array functions as a wavefront array and not as a systolic array. Thus it eliminates the synchronization problem but at

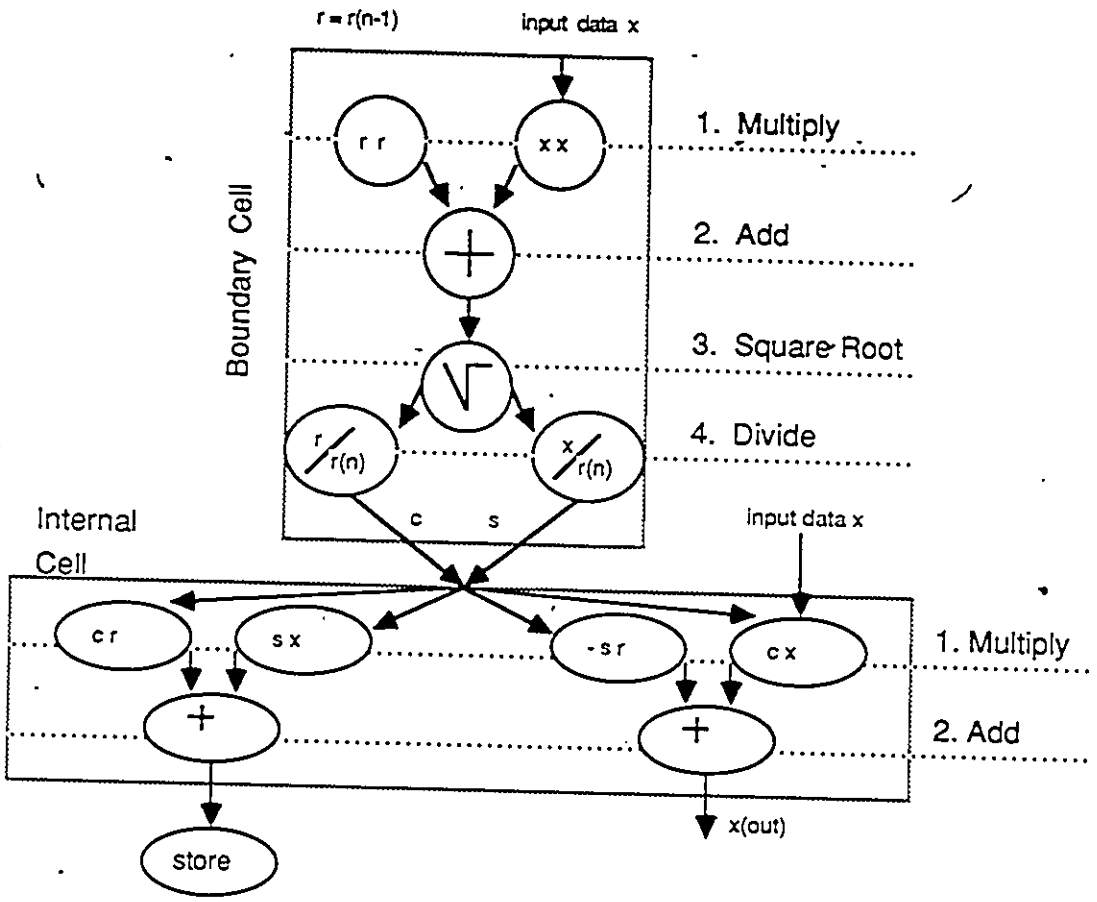


Figure 3.1: Order of operation within the cells

the same time it requires local control which means global connections, a factor which we try to overcome. Furthermore the transputer chip array is using the square root free Givens rotations, a modification introduced which eliminates the need for the square root operation. This modification can result in an unstable version of the algorithm and it also requires more complicated control structure for the array. The complexity is due to extra parameters that need to be introduced to the boundary cells.

Since the triangular array was developed for the purpose of VLSI technology, direct hardware realization as in [Ciminiera 86] realizes the full advantage of the algorithms regularity and modularity. This hardware re-

Company	size-bits	Delay (ns)	Technology	Operation
TI	16 * 16	14	CMOS	Addition
TRW	16 * 16	40	CMOS	Fixed Point Multip.
TI	16 * 16	100	CMOS	Fixed Point Multip.
Harris	12 * 12	120	CMOS	square-root look-up with ROM
Signetics	16 * 16	200	CMOS	square-root look-up with ROM

Table 3.1: Arithmetic speed summary

alization will also require $\frac{1}{2}[N^2 + 3N]$ processing elements, where N is the number of taps for an equivalent transversal filter. Out of the total there are N boundary cells and $\frac{1}{2}[N^2 + N]$ internal cells. The achievable speed of operation of the array is limited by the slowest processing cell which, as can be seen in Figure 3.1, is the boundary cell.

In Figure 3.1 the data $X(n)$ is loaded into the boundary cell where it is added to the previous $r^2(n-1)$. After performing the square root operation (compute 2.16), which is the slowest operation in the array, the newly computed $r(n)$ replaces $r(n-1)$ at the top of the boundary cell. The rotating elements c and s are computed using (2.14), (2.15), and are passed to the internal cell on the next clock cycle along with the next data component. The internal cells perform the add and multiply arithmetic and pass their result to the next row. Based on the table above (Table 3.1), it is possible to estimate the speed of operation of each cell. The boundary cell requires 2 multiplications that are done in parallel, 1 one addition, one square root operation with the help of a look up table and

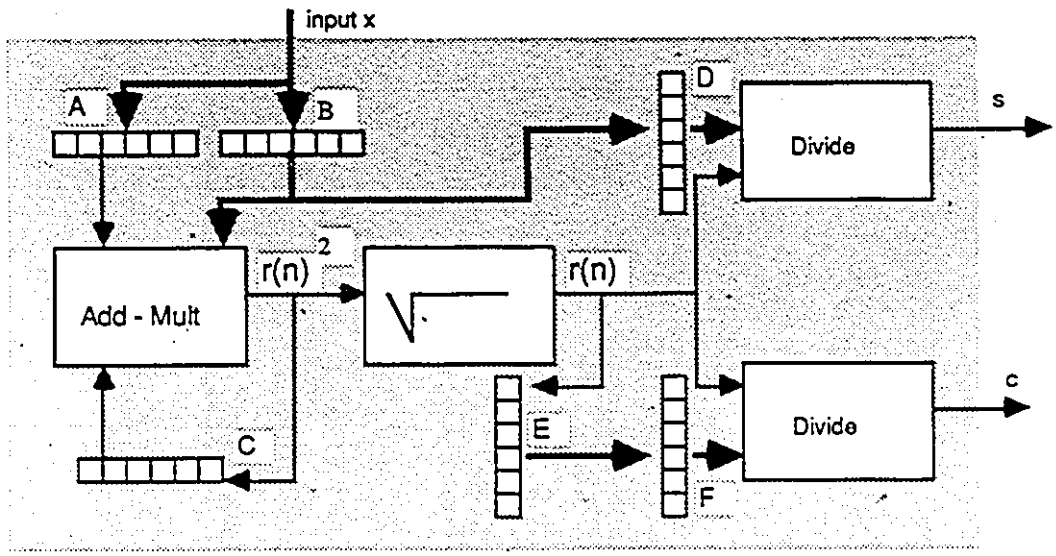


Figure 3.2: Internal structure of the boundary cell

two divisions in parallel (assume division as 5 multiplications), with possible processing time of 375 ns ($f = 2.67Mhz$) (TRW-MPYO16 multiplier, TI-SN54AS181A adder and Harris-HM6646 ROM) The internal cell requires 4 multiplications in parallel and two additions in parallel with processing time of 54 ns ($f = 18.5Mhz$). In Figure 3.2 the internal diagram of the boundary cell in the array is presented. The computation of $r^2(n)$ (computing 2.16) is performed recursively since the value $r^2(n-1)$ is equal to the value of $r^2(n)$ computed in the previous step. Thus the value of $r^2(n)$ ($r^2(n) = r^2(n-1) + x^2(n)$) is the sum of the square values of the number previously stored in the cell and the input x. The entering data is stored in A and B registers and the value of previously calculated $r^2(n)$ (which now becomes $r^2(n-1)$) is accumulated in the C register. Then the result produced by the multiplication addition block is passed to the square root look up table (executing 2.16). The output obtained ($r(n)$) is used for computing c and s, furthermore it is stored in the E register as the current value of $r(n)$ is the value of $r(n-1)$ in the next step. The registers D and F are used to

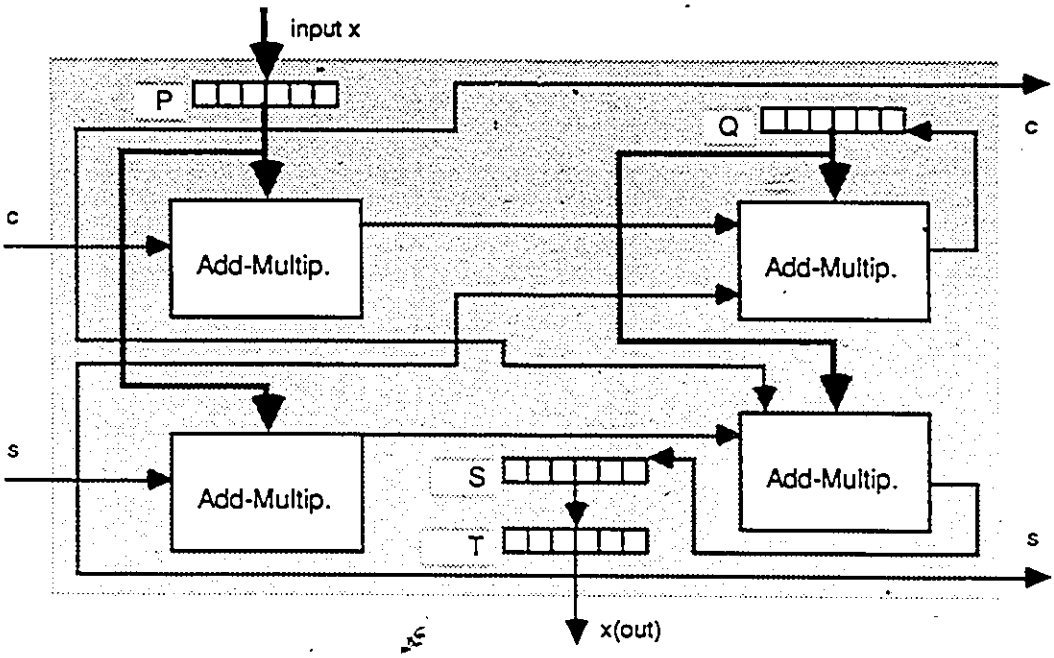


Figure 3.3: Internal structure of the internal cell

feed the division blocks with the parallel inputs x and $r(n-1)$.

The internal structure of the internal cell is shown in Figure 3.3. The new data enters the P register and the Q register holds the result of the previous step computations, that is the value of $r(n-1)$. After each step of computation $r(n)$ is stored in the Q register for one cycle delay as it becomes $r(n-1)$ in the next step. The output component x_{out} is stored in the S register and is ready to be transmitted to the corresponding neighboring cell. The T register is necessary as to delay the loading of the P register so that the operation begins when the digits s and c have been transmitted by the boundary cell. The achievable clock speed is 2.67 MHz (dictated by the boundary cells) regardless of the system order. Consequently we can sample the input signals at the same frequency. Thus it is possible for the array to process signals in the range of 1.3 Mhz. The order of the system can be determined from the requirements in the quality of interference

cancellation (smoothness of the filtered signal). The number of cells needed for realization is proportional, as before, to $\frac{1}{2}[N + 3N]$.

3.2 Modified structure

The hardware realization described in previous section realizes the array presented in chapter two (Figure 2.2) and it sets a large ROM requirement on the square root look up table. Also the complex boundary cell slows down the throughput rate of the array. Rearranging the arithmetic of the algorithm as in [McWhirter 86] to reduce the hardware required for realization and to speed up the performance is a necessary step towards practical realization. We can start by reexamining the operation done by the boundary cell (based on [McWhirter 86]), let $r_b(n)$ denote the element stored in the boundary cell and x_b as the input data to that cell, similarly $r_i(n)$ is the accumulated component stored in the internal cell and x_i is the input to that cell. Now we can rewrite the operation in the boundary (operation 2.16) cell as;

$$r_b{}^2(n) = r_b{}^2(n-1) + x_b{}^2 \tag{3.1}$$

and in the internal cell the accumulated component (operation 2.18) can be expressed as;

$$r_i(n) = \frac{r_b(n-1)}{r_b(n)} r_i(n-1) + \frac{x_b}{r_b(n)} x_i \tag{3.2}$$

or;

$$r_i(n) r_b(n) = r_i(n-1) r_b(n-1) + x_i x_b \tag{3.3}$$

and the component emerging at the bottom of the internal (operation 2.17) cell can be expressed as;

$$x_{out} = \frac{r_b(n-1)}{r_b(n)} x_i - \frac{x_b}{r_b(n)} r_i(n-1) \tag{3.4}$$

or,

$$x_{out} = \frac{x_i {r_b}^2(n-1) - x_b r_i(n-1) r_b(n-1)}{r_b(n) r_b(n-1)} \tag{3.5}$$

$$x_{out} = \frac{x_i {r_b}^2(n-1) - x_b [r_i(n) r_b(n) - x_b x_i]}{r_b(n) r_b(n-1)} \tag{3.6}$$

using (3.1) the above expression can be written as;

$$x_{out} = \frac{x_i {r_b}^2(n) - x_b r_i(n) r_b(n)}{r_b(n) r_b(n-1)} \tag{3.7}$$

and finally recalling that $c = \frac{r_b(n-1)}{r_b(n)}$,

$$x_{out} = [x_i - x_b \frac{r_i(n) r_b(n)}{{r_b}^2}] \frac{1}{c} \tag{3.8}$$

This is a very important result since now the function of the boundary cell and the internal cell can be simplified. For the modified boundary cells these operations are now necessary;

1.

$${R_b}^2 = R_b^2(n-1) + {X_b}^2 \tag{3.9}$$

2.

$$c = [\frac{R_b^2(n-1)}{R_b^2(n)}]^{\frac{1}{2}} \tag{3.10}$$

3.

$$\gamma_{out} = \gamma_{in}\, c \tag{3.11}$$

similarly for the internal cell, these steps are now required;

1.

$$R_i(n)R_b(n) = R_i(n-1)R_b(n-1) + X_iX_b \tag{3.12}$$

2.

$$X_{out} = [X_i - X_b\frac{R_i(n)R_b(n)}{R_b^2}]\frac{1}{c} \tag{3.13}$$

We note that there is no need to square the terms in (3.9) and multiply the terms in (3.12) every time since these are simple accumulations. This can be verified by defining $K_b(n) = R_b^2(n)$ and $K_i(n) = R_i(n)R_b(n)$ and the operation of the cells can be written in a simple manner as;

$$K_b(n) = K_b(n-1) + X_bX_b \tag{3.14}$$

$$K_i(n) = K_i(n-1) + X_bX_i \tag{3.15}$$

$$X_{out} = [X_i - X_b\frac{K_i(n)}{K_b(n)}]\frac{1}{c} \tag{3.16}$$

The proposed operations within the cells change the accumulated component and by that the accumulated element no longer represents the factored autocorrelation matrix or the factored cross correlation vector, hence we can not compute the weight vector $\mathbf{W}_{opt}$ according to (2.5) (Kung's array Figure 2.1) from this array. However, for the purpose of deriving the filtered signal (error) directly, the case which we are interested in, there is absolutely no change in performance. Exactly the same component emerges at the bottom of the array as with the array described in sections 3.1, and the end result has to be multiplied by γ to produce the a posteriori error. In terms of mathematical operations the proposed array has a big advantage over the previous structure. Even though the operations in the internal cells cannot be performed in parallel as before and the throughput rate

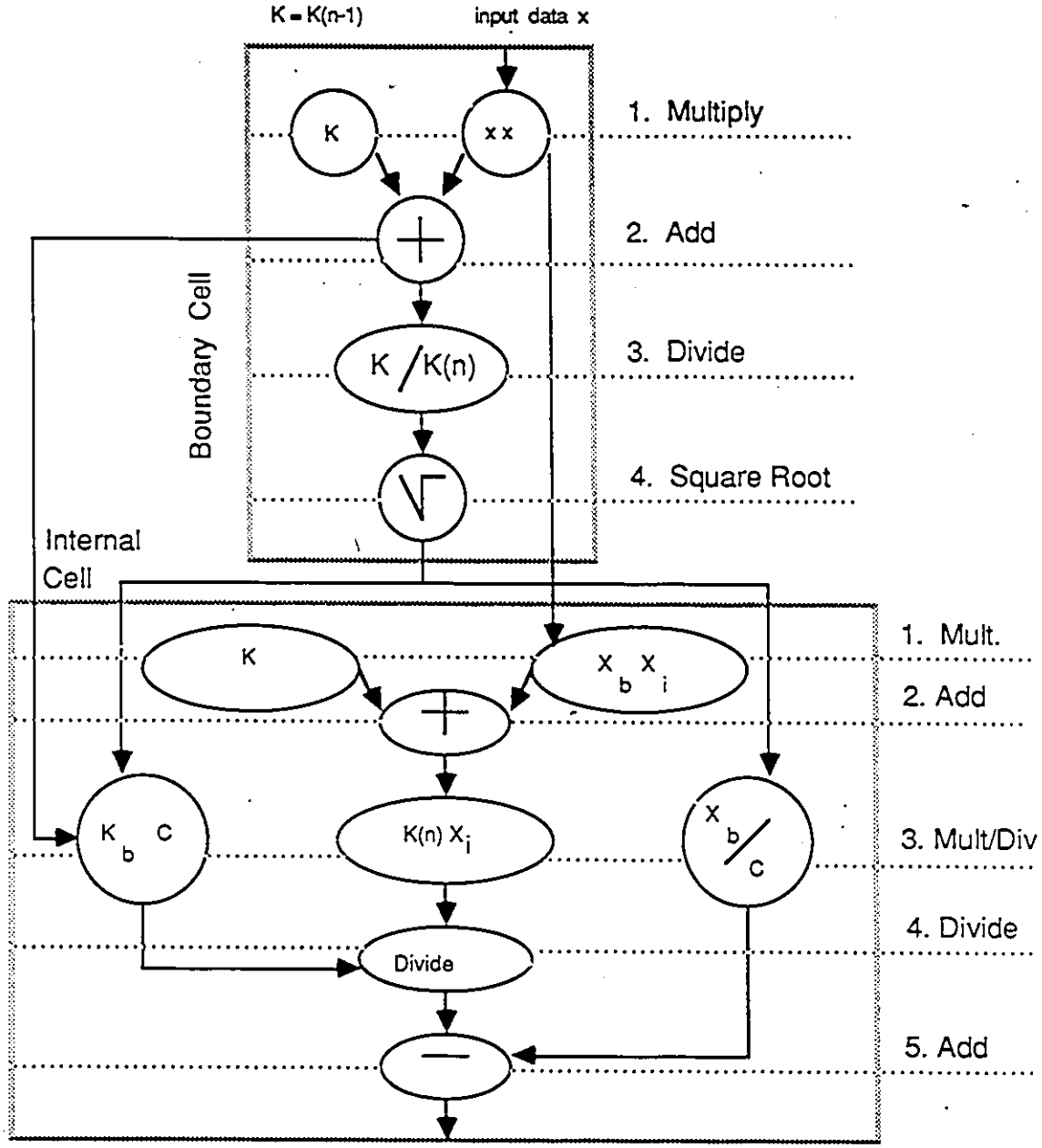


Figure 3.4: The sequence of operation of the modified cells

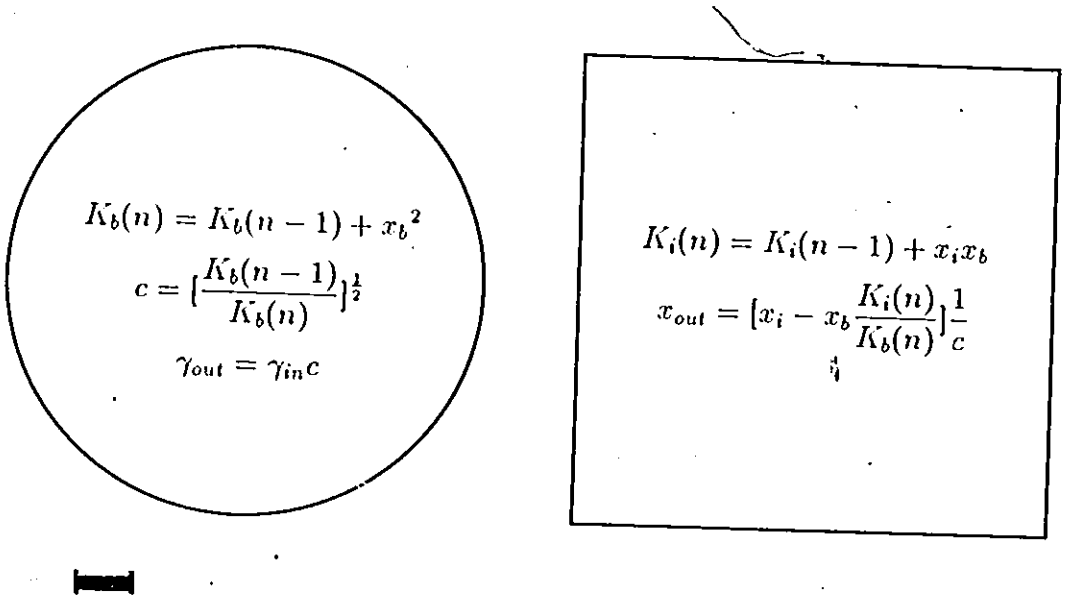


Figure 3.5: Operations in the modified cells

of the internal cells has increased to 300 ns, the throughput of the array has increased since the speed of operation of the boundary cells decreased to 300 ns as well (assuming 8 bits accuracy of ROM). The boundary cells perform the same operations as before, however not in the same order, thus the number of operations is equivalent. The big advantage of this structure comes into effect in the square root computations [McWhirter 86]. In the previous structure the square root operation would determine the $r(n)$ component in the boundary cell which is also one of the eigenvalues of the autocorrelation matrix, this value depends on the input signal and can take any positive numerical value. This makes the look up table very large, since it is necessary to assign large number of bits in order to achieve the necessary range and accuracy. In the new structure the square root operation calculates the cosine component of the Givens rotation, thus the square root operation can take only the value between 0 and 1, starting with 0 the cosine is being updated up to the point when the process converges and then it takes the value of 1. Clearly the fact that the lookup table has to store only the values between 0 and 1 *regardless* of the input signal and with improvement in processing time, which now is 300 ns ($f = 3.3Mhz$), makes this structure more attractive then the previous one. The disadvantage of this structure as said before is that it is not suitable for derivation of the weight vector explicitly.

It is necessary now to come back to the comparison with the LMS algorithm and introduce a chip which already has been designed to work as an adaptive filter. The DSP56200 by Motorola is a cascadable adaptive finite impulse response digital filter which can work in a fixed or adaptive

mode. For the case of 8 cascaded chips giving 256 taps the filter can achieve a clock rate of 115Khz, and for the case of a single chip with 32 taps the throughput rate can be as fast as 123Khz. Such a fast chip exists and can be used for noise cancellation. The modified structure of previous section in comparison to this chip still seems to be a bit complicated thus further simplifications are necessary in order to make the systolic array more compatible with the LMS. This will be considered further in the next section.

3.3 The Sequential Decorrelation Algorithm

The modification in the mathematical operations resulted in a simpler hardware realization, however it did not reduce the number of operations in the boundary cells leaving the throughput rate of the array only slightly improved. In order to make the array suitable for higher speed operation and making it more compatible with the LMS algorithm, simpler and fewer operations are necessary in the cells. Orthogonalization of matrix could be done with different methods then the Givens rotations. Another method is the Gram Schmidt orthogonalization [GSO] which has very good numerical properties, and due to its regularity, it is suitable for systolic implementation as was done by Yuen [Yuen 85]. This structure is not as efficient and suitable for VLSI as a structure using Givens rotations since it requires some global connections. The procedure is carried out column by column and the entire data matrix must be known a priori. This results in large amount of memory and control circuitry. The entire procedure must

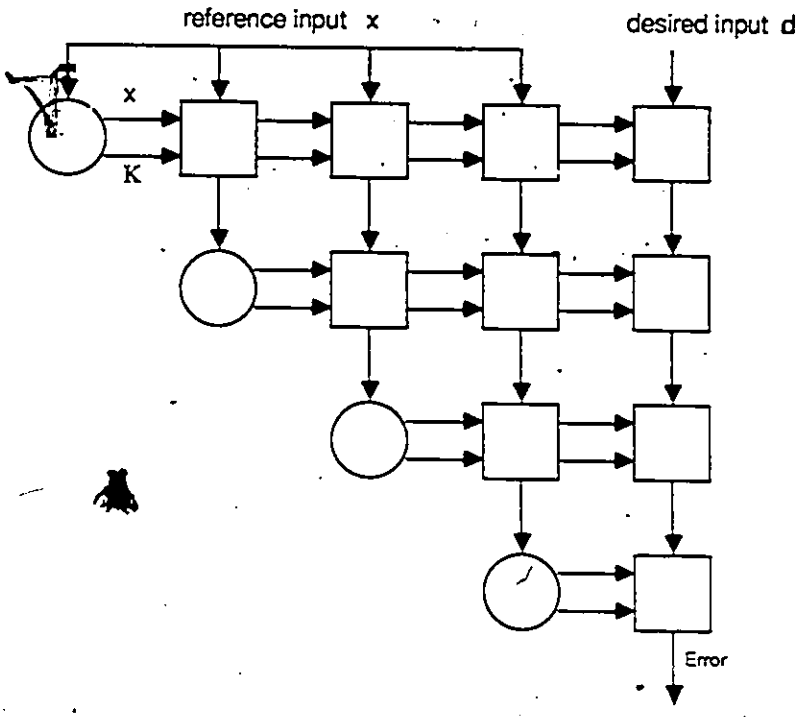


Figure 3.6: The SD array.

also be repeated for every new data point that enters. This properties are making the GSO suitable for sequential block operation where a solution is given for each block. An algorithm which operates row by row as the Givens method is much more suitable for continued operation and for that reason the GSO is often approximated by the sequential decorrelation [SD] algorithm [McWhirter 86]. The operation of the sequential decorrelation algorithm is shown in Figure 3.6.

The SD algorithm does not have the good properties of the GSO algorithm since it orthogolises the output of each cell before the entire input data has been received and as a result the output is not truly orthogolised, however, the very simple mathematical operations and the clear connection to the Givens method as it will be seen later makes this algorithm extremely attractive. The operation of the cells are shown in Figure 3.7, the boundary cell needs only to square the input and add it to the previously stored

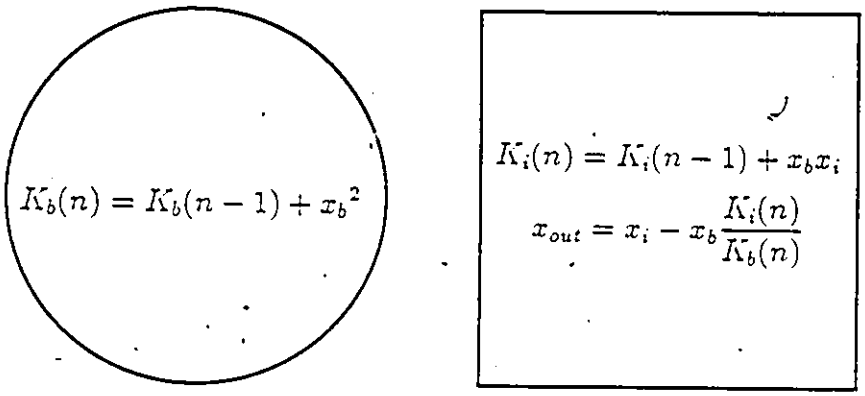


Figure 3.7: Operations of the SD array within the cells

one. The internal cells, which are now more complicated than the boundary cells, need to multiply the input and accumulate the result. Followed by additional operation of division multiply and add, which cannot be done in parallel with the multiply accumulate. Based on Table 3.1 in section 3.1 the speed of operation of this array can be as low as 250ns.

There is a very close resemblance between the SD array and the modified Givens array proposed in the previous section, equations (3.14) (3.15) and (3.16) are almost identical to the operation of the SD algorithm. The only difference between the two is the cosine component in the Givens structure and the final multiplication through γ at the bottom of the array. From simulations in previous chapter (Figure 2.7) and from theory we know that the cosine component in Givens rotations approaches 1 as the process converges and in the same manner so does γ, thus we can say that for slow varying environment and for large n the performance of the SD algorithm and the Givens rotation method should yield the same misadjustment. The difference between the two comes in the adaptation period

where in the Givens case we can expect the signal with some noise that diminishes with the increasing number of iterations and in the SD case the noise and the signal could be initially cancelled and only after the process converges the signal could be completely recovered. Also since the algorithm only approximates the solution at the beginning, it will require much more data in order to give the correct weight values, thus slower convergence. The forgetting factor λ should be very close to one in order to give the algorithm sufficient amount of data. We must note that the convergence period will not become unstable (diverge), due to the approximation, we refer to this period as unexpected since the signal can be affected in some way. Ignoring this short transition period in which we are not interested in practice (can be seen as zero output) the SD algorithm which can be seen also as modified Givens offers a very attractive structure with simple operations compatible to the LMS and faster processing time then the original Givens structure. The convergence curves of the SD algorithm are shown in Figures 3.8, 3.9. Hence the SD array may have slower convergence rate then the original Givens, but faster processing time can result in a faster performance overall.

In general for all the Givens arrays special care should be given when assigning the value of λ. If the value is to low additional self noise (misadjustment) caused by the fourth order statistics of the input signal will be present [Eleftheriou 84] along with the misadjustment formula developed in Chapter 2, this is already noticeable when giving λ the value of 0.9. On the other hand assigning λ a value very close to 1 will slow down the convergence and will set high requirements on the word length, thus some

compromise must be achieved between the two. A good value of λ, for fixed point arithmetic, can be obtained from the expression below where $\sigma_q^2 = \frac{1}{12}\, 2^{-2B}$ and B is the number of bits used in the process, σ_x^2 is the variance of the input signal and σ_n^2 is the variance of the noise in the desired input.

$$\lambda_{opt} = 1 - \frac{\sigma_q \sigma_x}{\sigma_n} \tag{3.17}$$

Finite precision simulations (fixed point), were performed for the SD array and the LMS algorithm (Figures 3.11, 3.12). The systolic array does not perform well for precision under 14 bits, this was expected since the array is iterative accumulating errors can result in poor performance. This result agrees with the RLS algorithm which also requires high precision. The long word length requirements of 16 bits (fixed point), are typical to all the Givens structures. The LMS, as expected, was found to be reliable even for 8 bits word length.

To summarize and compare the possible hardware realizations of the **QR** systolic array of this chapter, array we can say:

- The square root free Givens rotation is the fastest version of the algorithm, however, it requires the largest amount of circuitry . Due to some instability reported [Speiser 86] and the complex hardware structure it was not considered in this thesis.
- The conventional Givens rotations are a slower version of the algorithm because of the square root operation in the boundary cells. On the other hand it requires less hardware. The operation of this array is fully described by (2.14), (2.15), (2.16) for the boundary cells, and

(2.17), (2.18) for the internal cells. As a result of complex boundary cells and simple internal cells, there is a significant difference in processing time between them. The speed of operation is dictated by the boundary cells.

- By rearranging the operations in the conventional Givens structure, the modified Givens was obtained offering simpler boundary cells. The modified Givens operations are described in Figure 3.5. In this case the square root operation is bounded to the values between 0 and 1, making the look up table simpler. This development comes at the expense of more complex internal cells which now have processing time identical to the boundary cells. Thus there is no difference in processing time between the cells. The speed of operation of this array is slightly better (75 ns improvement) to the previous one.

- The sequential decorrelation algorithm, derived from Gram Schmidt orthogonalization, has a very similar structure to the modified Givens array. The SD algorithm is described by Figures 3.6, 3.7. The SD produces the same level of self noise as the Givens structure after convergence. Thus these algorithms are equivalent in terms of misadjustment. In terms of speed of convergence the SD is slower then the Givens array, since it only approximates the solution during the learning period. Consequently it will require more data in order to converge. On the other hand, due to simpler cells, the throughput of this array is faster then for the Givens rotation algorithm making it suitable for high speed operation. The boundary cells become very

simple and could be realized by a simple multiply accumulate.

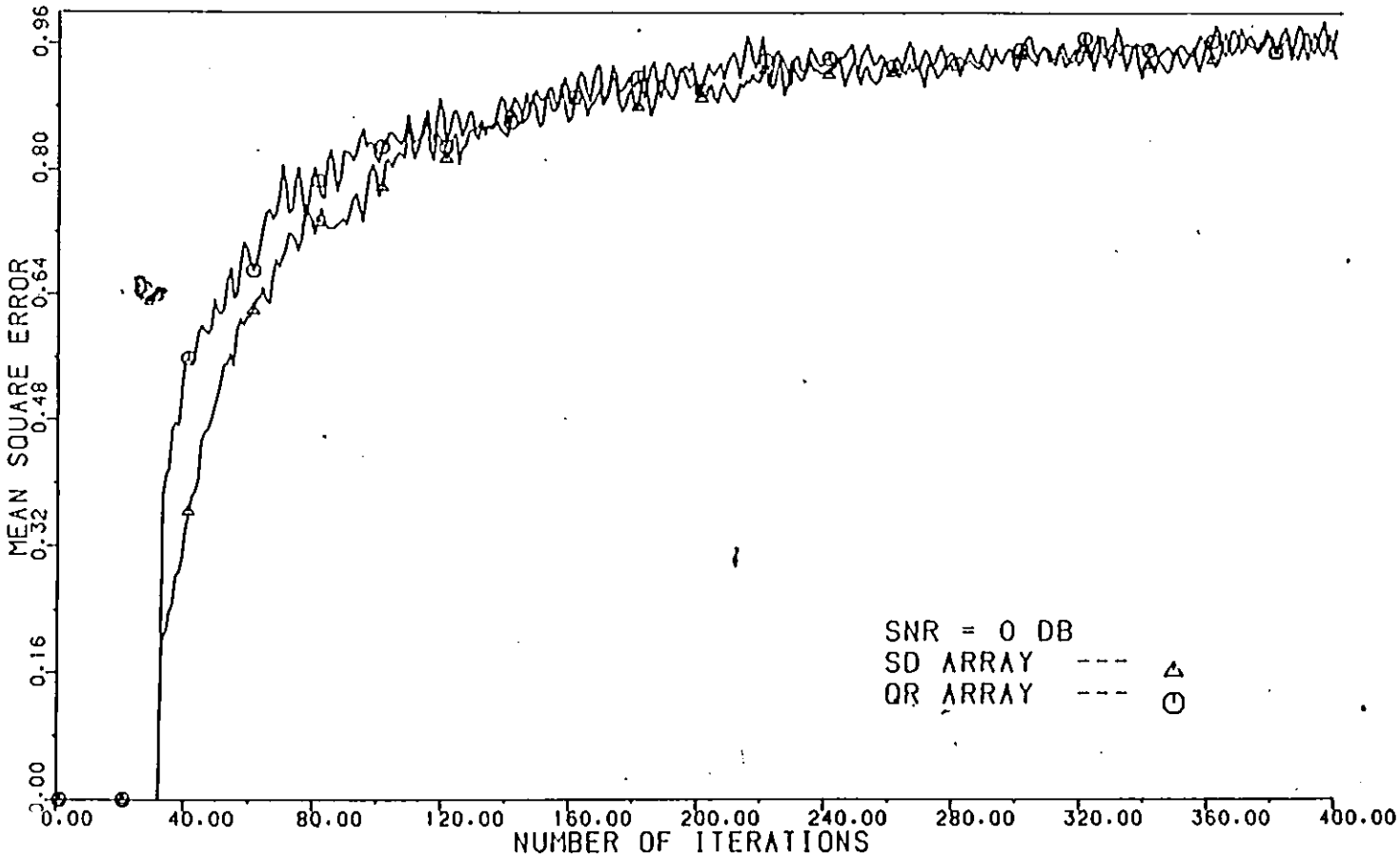


Figure 3.8: SD mean square error compared to QR decomposition $\lambda = 1$ ($N = 16$)

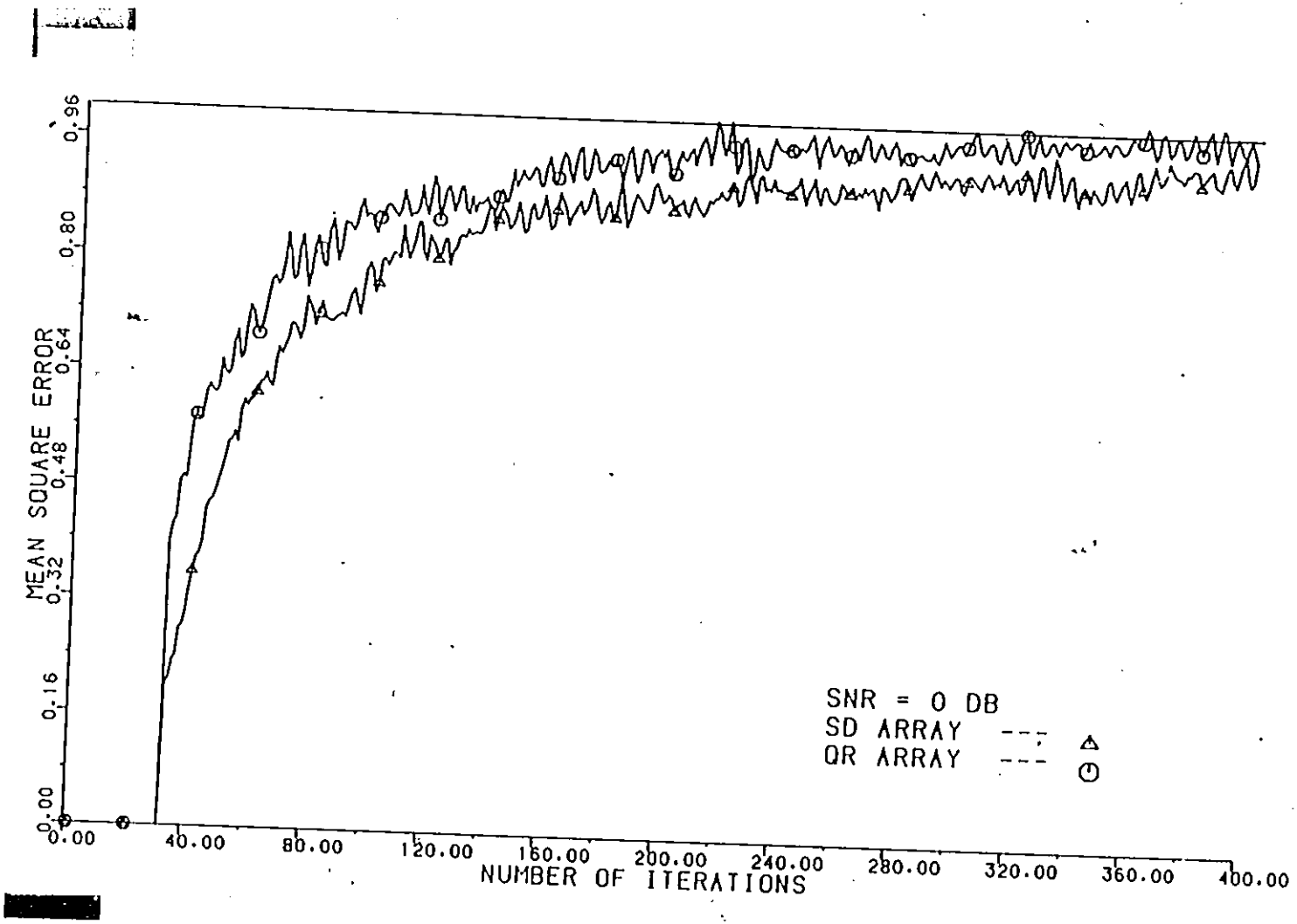


Figure 3.9: SD mean square error compared to QR decomposition $\lambda = 0.999$ ($N = 16$)

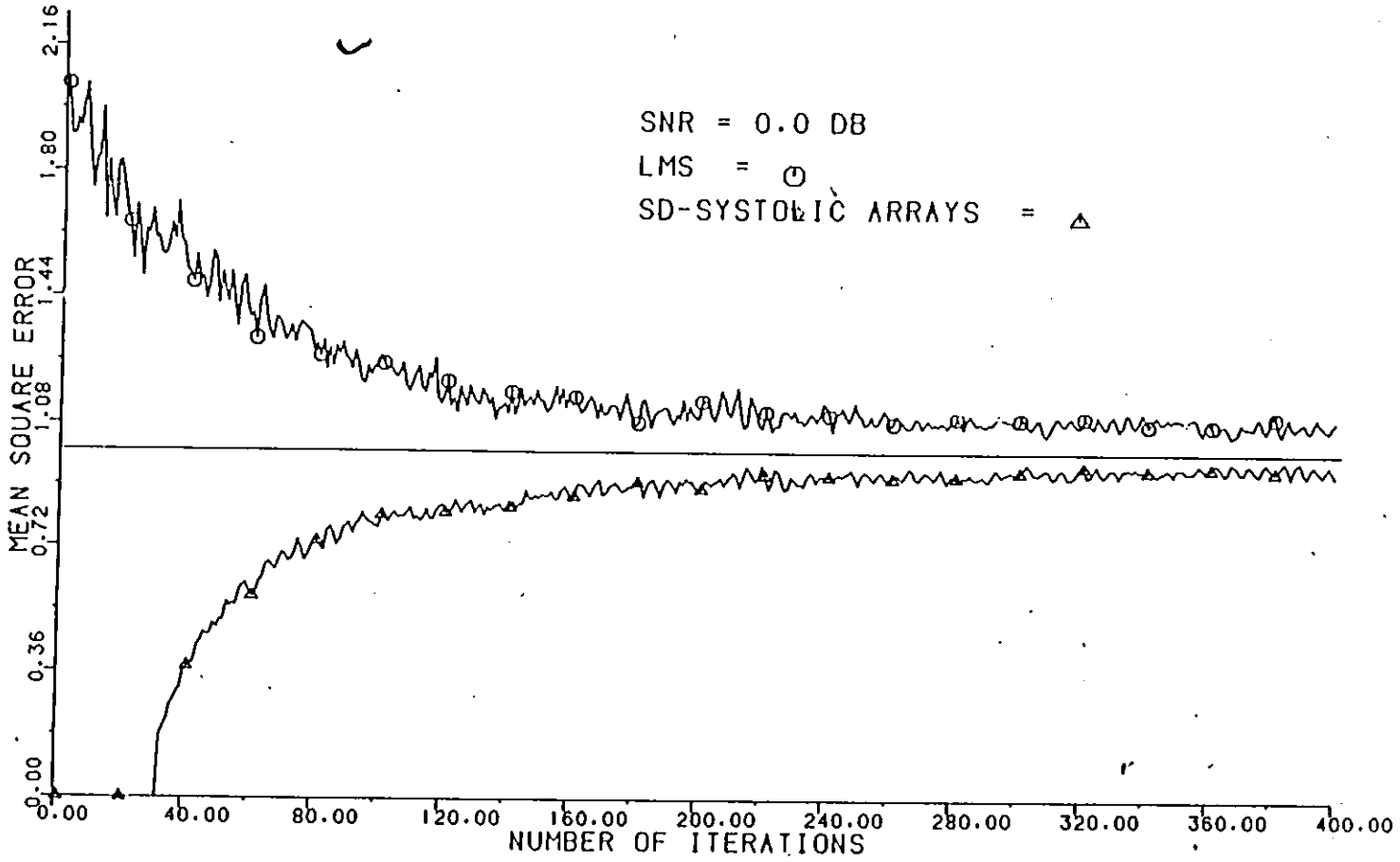


Figure 3.10: SD mean square error compared to LMS ($N = 16$)

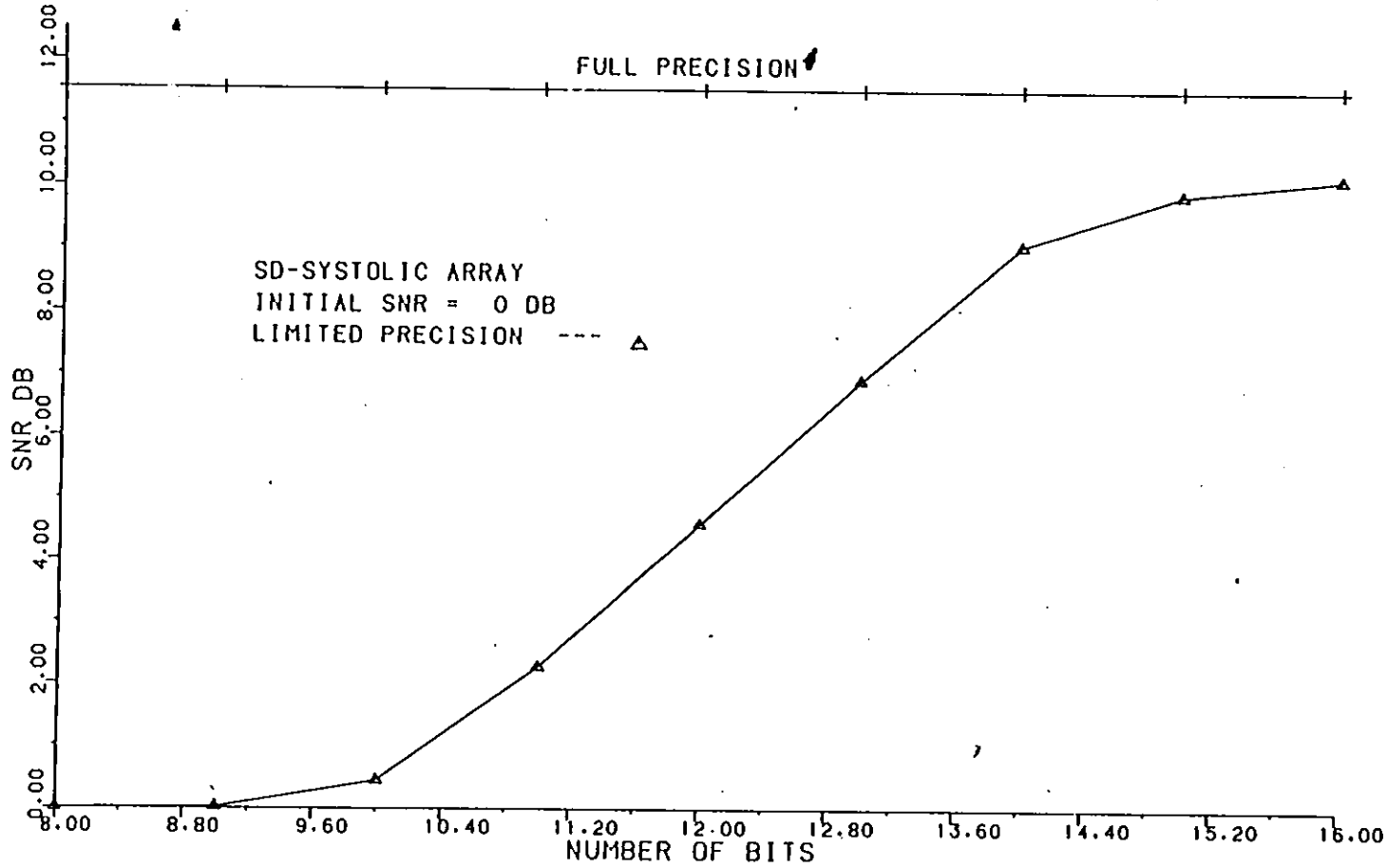


Figure 3.11: Finite precision plot (Fixed point arithmetic) for the SD algorithm

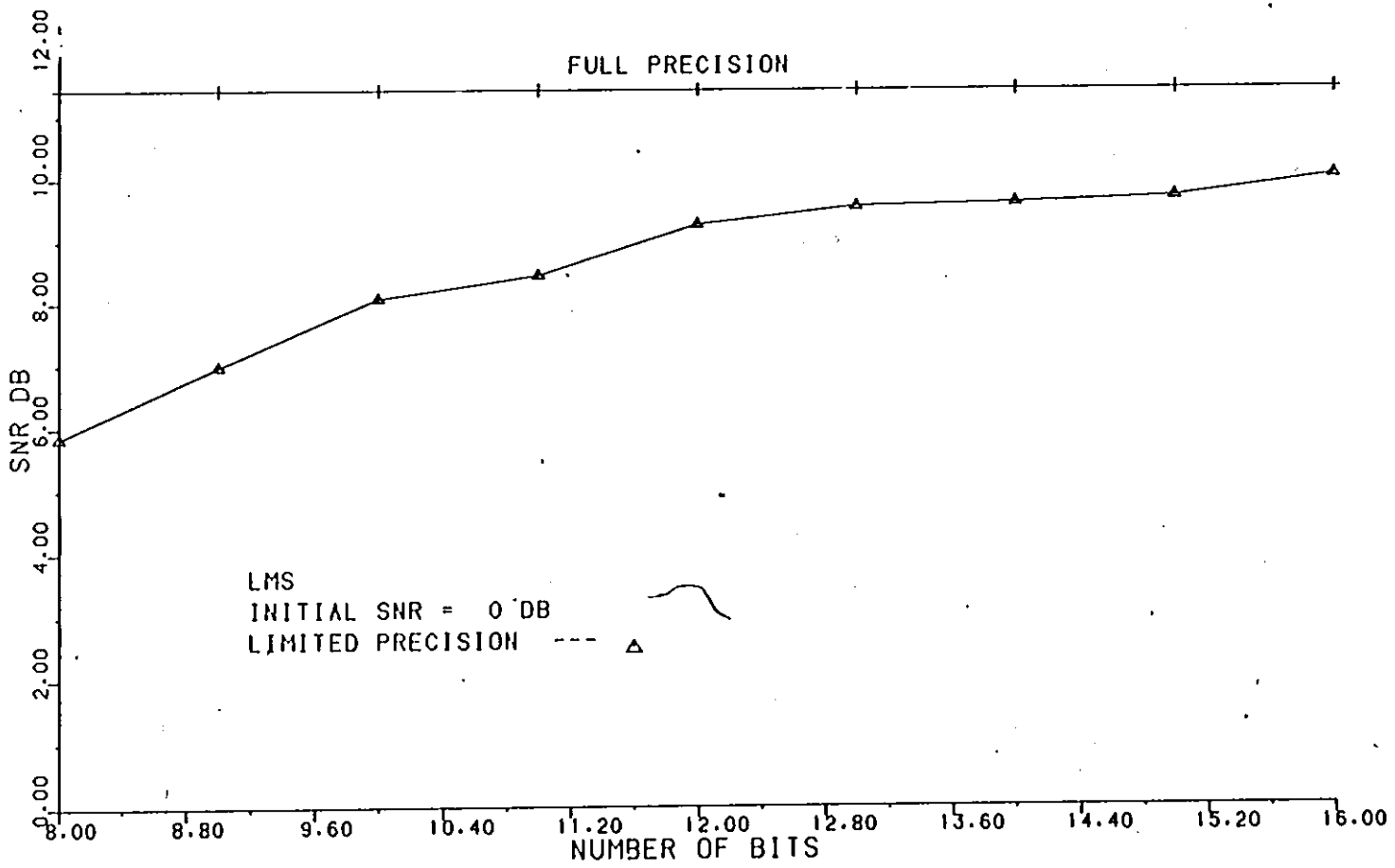


Figure 3.12: Finite precision plot (Fixed point arithmetic) for the LMS algorithm

Chapter 4

Systolic LMS Realization

4.1 Conventional implementation of the LMS

One possible implementation of the LMS algorithm using the Motorola 56200 chip was mentioned in chapter 3, however, this method of implementation is not widely used. A more conventional method using one delay line is shown in Figure 4.1. In this realization one tap delay line of length N (N is the filter order) will update the coefficients and as well will perform the convolution (filtering process). For the convolution part of the filter N multipliers will be necessary and one adder. For the update part N multiply-accumulate will be needed, one multiplier and one adder. Thus for every iteration the LMS in this form will perform 2N+1 multiplications and N+2 additions.

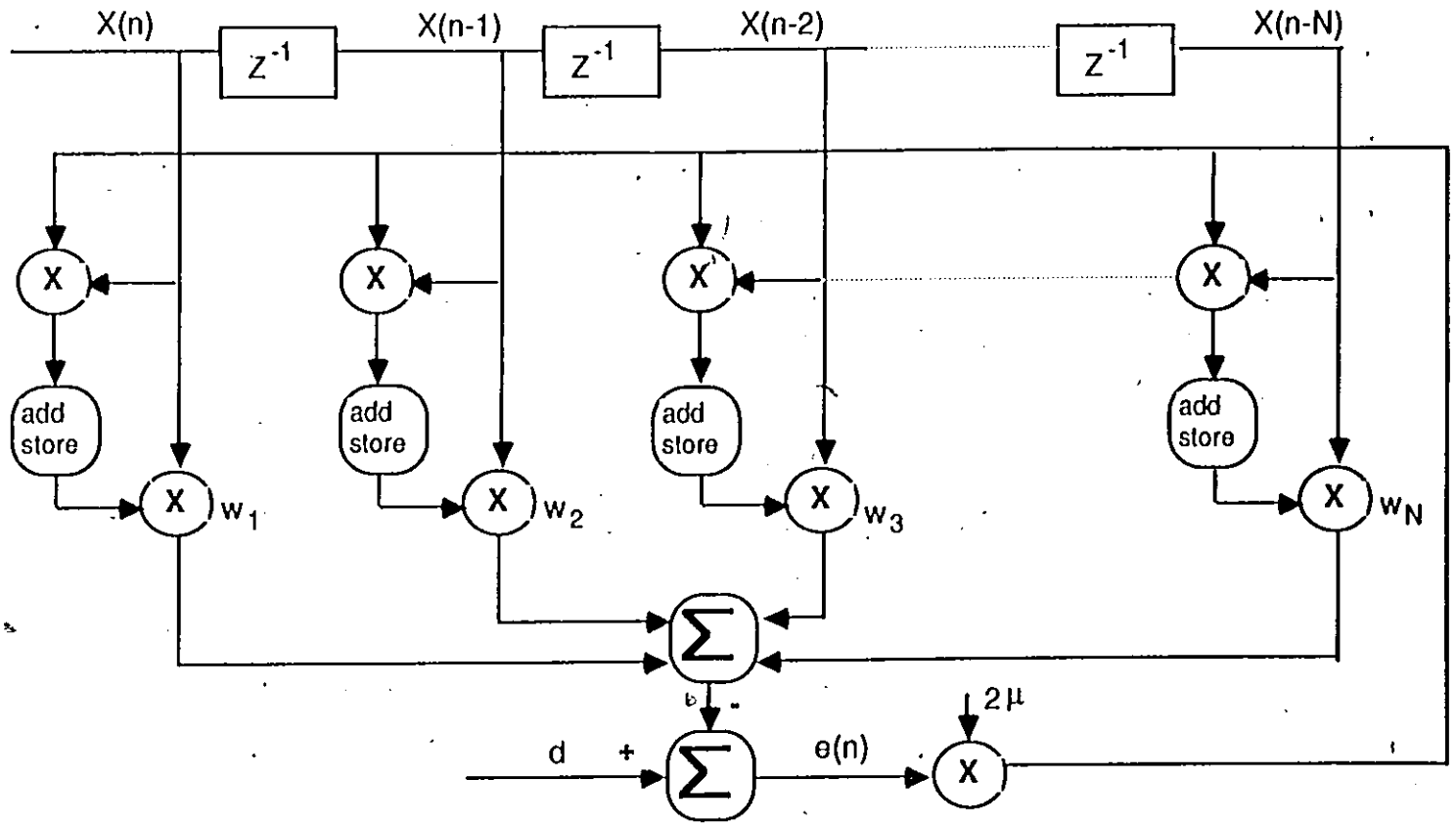


Figure 4.1: A direct implementation of the LMS algorithm

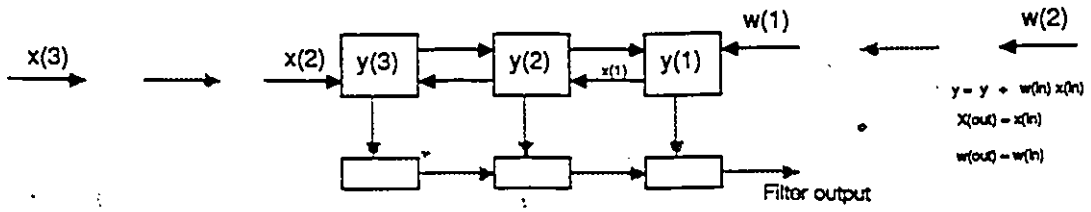


Figure 4.2: Systolic convolution array (1st structure)

4.2 Systolic convolution with LMS update

The systolic architecture has the great advantage of regularity and modularity over conventional architectures and is realizable in VLSI. So far our comparison has been between systolic arrays and a transversal adaptive filter realizing the LMS algorithm. One of the conclusions arising from this comparison is that the LMS algorithm is powerful even in comparison to QR decomposition, in terms of wordlength required, hardware and speed. Implementing the LMS algorithm in a systolic array structure is possible, to obtain both the excellent numerical properties and a structure for VLSI implementation. A systolic realization of the LMS algorithm, the linear combiner (1.3) and the update (1.19), is discussed in the following two sections.

There are several structures of systolic arrays suitable to perform convolution some of these can be found in [Kung 82]. In Figure 4.2 a structure is shown where the data X_i and the weights W_i move systolically in opposite directions, and are multiplied in the cells. The product y is accumulated within the cells. In order for each data point to meet the appropriate weight point, the data points X_i are separated by two clock cycles as well as the weight points W. The outputs from the cells are collected sequentially which eliminates the need for a bus or any other network.

In another structure (Figure 4.3) both the data and the weights systolically move from left to right, however, the X vector moves twice as fast as the W vector, so the W vector takes twice as long to move through the array than the data vector. Compared to the previous structure this structure is more efficient since all the cells work all the time, while performing the convolution.

Both structures are suitable for adaptive filtering since the weights can be updated every clock cycle and fed into the array in a continuous manner.

A fast systolic FIR filter chip (20 Mhz) already exists on the market (TMC2243 by TRW) and making this filter adaptive, would allow high speed operation. However, the problem arises in the update equation, $W_{n+1} = W_n + 2\mu e X^T(n)$. In the systolic structure of Figures 4.2, 4.3 the output error e can only be obtained after N clock cycles, it is dependent on the amount of time required to complete the convolution y. The weights in the LMS algorithm are usually updated in parallel and as well are loaded into the filter in parallel. Thus all the weights in the FIR filter are changed at the same time. Savov and Steenaart [Savov 87] have shown that the weight values can be loaded serially into the filter while maintaining the same speed as parallel access. To accomplish this, the weight values are being updated using a multiplier accumulator outside the filter every clock cycle and are loaded to the filter every N clock cycles (N is the filter order). Although the speed of operation remains equivalent to parallel access this type of structure will require more hardware.

Another structure, which will require less hardware, is possible by updating one coefficient every clock cycle and loading this coefficient into the

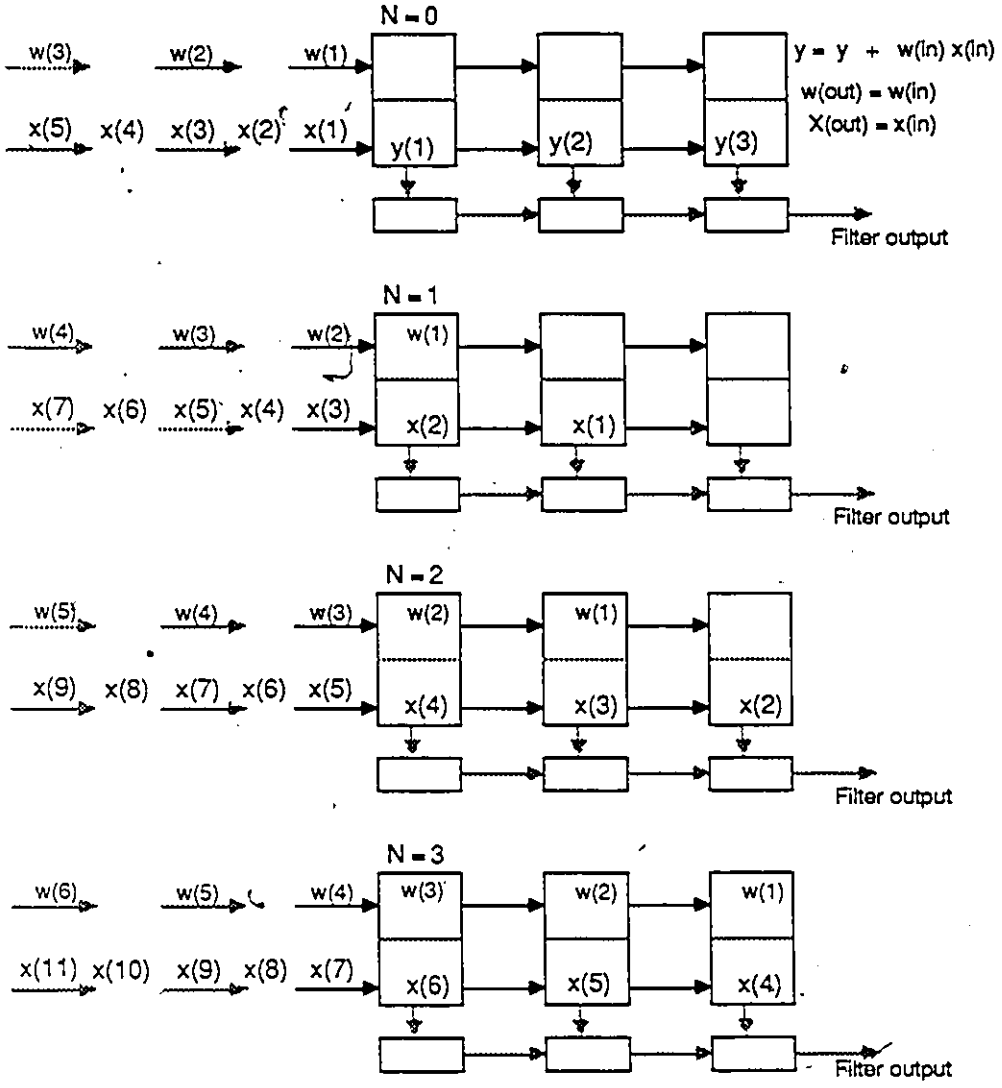


Figure 4.3: Systolic convolution array (2nd structure)

filter. In this case one multiplier accumulator will be necessary and each coefficient will be updated every N clock cycles. The reduction in hardware requirement comes at the expense of speed. In both realizations mentioned above, at every clock cycle the error e, is computed by using a weight vector which is composed of $N-1$ old coefficients and one new coefficient. In order to compute the error e due to all new vector W it is necessary to update the weight values all at the same time every N clock cycles (time required for the convolution to be complete). Thus all the coefficients are updated in parallel every N clock cycles, the update equation is given in (4.1). This type of architecture is practical when there is a need to share the adaptive part among few filters. It is also the slowest case of coefficient update.

$$W_{n+1} = W_{n-N} + 2\mu e(n)X^T(n) \tag{4.1}$$

Simulations for this expression (4.1) (worst case) were performed and the results show that the systolic array with this update can be used in adaptive filtering (other structures mentioned will give better results). Still, however, only the convolution part of this structure is implemented as a systolic structure, leaving the update equation to be evaluated with other conventional method. The aim now will be to implement the update equation in a systolic structure also, and this is discussed in the next section.

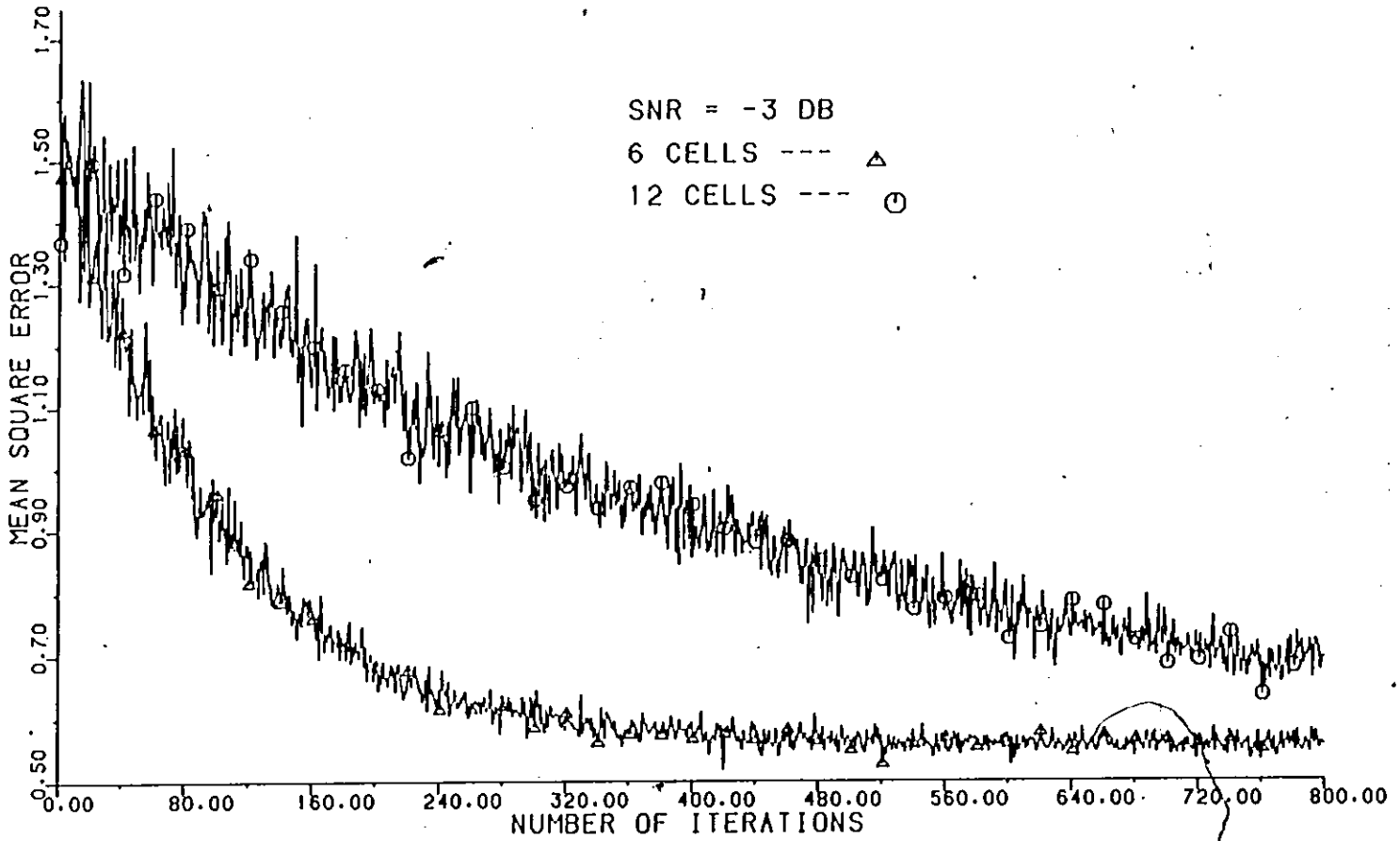


Figure 4.4: LMS learning curve for systolic convolution update

4.3 Convolution and update in systolic structure

The output of a transversal filter is given by;

$$y(k) = \sum_{n=1}^{N} x(t-n)h_n(t) \tag{4.2}$$

where $x(k-n)$ is the input signal at time $k-n$, h_n is the n^{th} sample of the filter impulse response, and N is the total number of sample values. When the signal $x(k-n)$ is coded it has the form of a digital word with k bits which take on the logical values of 0 or 1. However, if the word $b_i(k-n)$ is coded in offset binary, logical 0 takes the value -1, the signal sample can be expressed as;

$$x(k-n) = \sum_{i=1}^{m} b_i(k-n)2^{-i} \tag{4.3}$$

$b_1(k-n)$ is the most significant bit and $b_m(k-n)$ is the least significant bit. Substitution of (4.3) into (4.2) produces the filter output signal;

$$y(k) = \sum_{n=1}^{N} h_n[\sum_{i=1}^{m} b_i(k-n)2^{-i}] \tag{4.4}$$

With the distributed arithmetic approach the summations in (4.4) are reversed so that;

$$y(k) = \sum_{i=1}^{N} 2^{-i}[\sum_{n=1}^{m} b_i(k-n)h_n] \tag{4.5}$$

By reversing the summations the bits $b_i(k-n)$ from different signal samples are first multiplied by the weights h_n and the result is presented as a scaled accumulations. We define the following vector as;

$$F^T = (f(1), f(2), \ldots, f(k)) \tag{4.6}$$

$f(k)$ is the k^{th} partial product used in the output accumulation of distributed arithmetic filter. Writing the scaling factor S as a vector;

$$S^T = (2^{-1}, 2^{-2}, \ldots, 2^{-k}) \tag{4.7}$$

B is the $N \times K$ array of bit values (binary offset) or B is X decomposed into component bits. Using these definitions (4.6), (4.7) we can rewrite (4.3) the input signal, as;

$$X = BS \tag{4.8}$$

With the column H representing the set of N filter coefficients, the output of the filter is given by the convolution;

$$y = X^T H \tag{4.9}$$

or

$$y = (BS)^T H \tag{4.10}$$

$$y = S^T B^T H \tag{4.11}$$

now express the vector F as;

$$F = B^T H \tag{4.12}$$

(F is the product in brackets in 4.5)

$$y = S^T F \tag{4.13}$$

The vector F is the set of partial products used in the output accumulations. Thus in order to compute y the filter coefficient vector H goes through two operations. First it is multiplied by each row vector of B^T to produce k

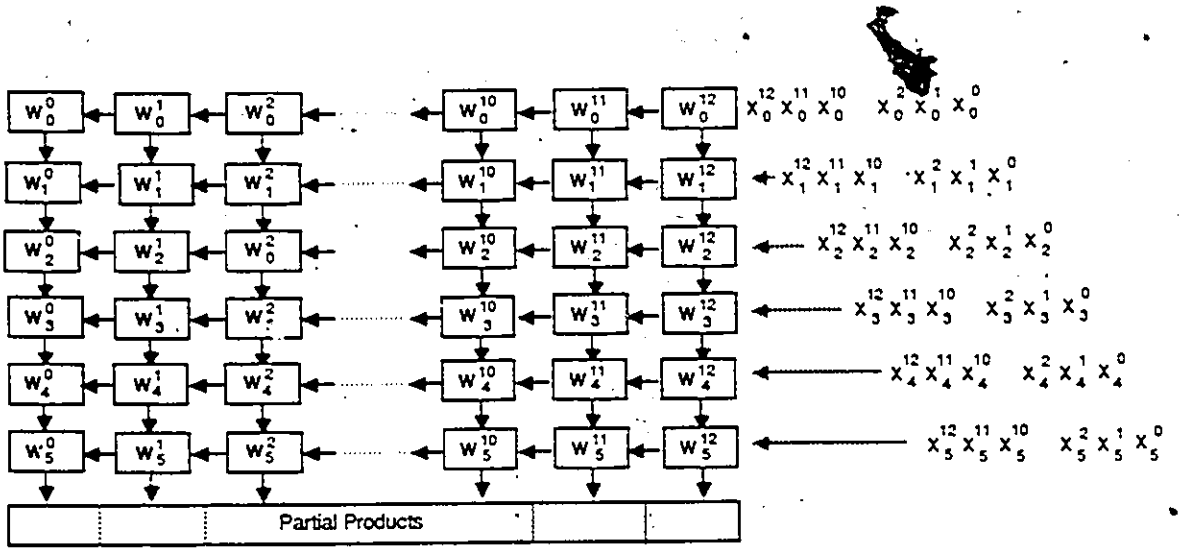


Figure 4.5: Distributed arithmetic filter in a systolic structure

components of partial product vector F (4.12) and second it is multiplied by the scaling (weighting) factor S (4.13).

This type of distributed arithmetic filter can be implemented with a systolic array as shown in Figure 4.5 [Walicki 85]. The w_j^i are bits of a digital representation of the weight coefficient W which are stored in the RAM locations of the individual processing elements. Partial products appear on the bottom edge of the array. Every processing element performs the following basic operations;

$$x_{left} \longleftarrow x_{right} \tag{4.14}$$

$$p_{down} \longleftarrow p_{up} + x_{right} w_{stored} \tag{4.15}$$

If the least significant bits of x and W interact first, then the partial products bits of equal significance emerge at the bottom of the array at the same time. These partial products can be accumulated using a systolic output path to produce the filter output y. Now that the convolution part

has been implemented in distributed arithmetic we can turn to the update equation and rewrite it in terms of the partial products. The LMS algorithm is given as, $W(n+1) = W(n)+2\mu e.X(n)$. We can rewrite the partial product update in terms of weight update such that;

$$F(n+1) = B^T(n)W(n+1) \tag{4.16}$$

also multiply the LMS expression by B^T to give;

$$B^T W(n+1) = B^T[W(n) + 2\mu e.X(n)] \tag{4.17}$$

Substituting 4.16 with the left hand side of 4.17 yields;

$$F(n+1) = B^T[W(n) + 2\mu e.X(n)] \tag{4.18}$$

this can be simplified to;

$$F(n+1) = B^T[W(n) + 2\mu eBS] \tag{4.19}$$

$$F(n+1) = F(n) + 2\mu eB^T BS \tag{4.20}$$

However, the matrix product $B^T B$ is a symmetric matrix with diagonal elements equal to N. This is because of the offset binary signal coding which implies that the square of any element of B is unity (the square products are on the diagonal). If we assume that the input signal is zero mean and that successive samples are uncorrelated, then the expectation of any nondiagonal element of $B^T B$ is zero, and the matrix reduces in the mean to the scalar N ($E\{B^T B\} = N$). And the LMS algorithm in terms of partial products becomes;

$$F(n+1) = F(n) + 2\mu NeS \tag{4.21}$$

The term $2\mu N$ is constant and the scaling vector S is just a constant shift of k bits, thus the update is simply the shifted error. If the gain constant μ is chosen properly the linear error e could be replaced by the sign of the error such that;

$$e = \begin{cases} +1 & e > 0 \\ -1 & e < 0 \end{cases} \tag{4.22}$$

and the correction in the update equation becomes either a positive or a negative constant. Hence now we have an adaptive algorithm in terms of the partial products, however since the convolution part uses the explicit values of weights, it is necessary to transform the partial products (after they have been updated) back into weight values. This could be done using the relation developed in [Cowan 83] $\sum_{n=1}^{N} b_n f = h_n 2^{N-1}$ and another systolic array similar to the one that performs the convolution. The weights relate to the partial products as given below;

$$B(n)F(n+1) = NW(n+1) \tag{4.23}$$

(this was obtained by multiplying (4.16) by $B(n)$) and an array similar to the one shown in Figure 4.5 could compute the weight values. This array works almost in an identical way as the convolution array with one difference, the stored values in the array are the partial products and not the weight values. The results emerging at the bottom of the array are the explicit weight values (after proper scaling) which again are to be assigned to the convolution array and the system could work in a continuous manner. The entire system is shown in Figure 4.6 and could be divided into three parts: 1.Convolution array 2.Computing the error and updating the partial products 3. Transforming the partial products into updated weight values.

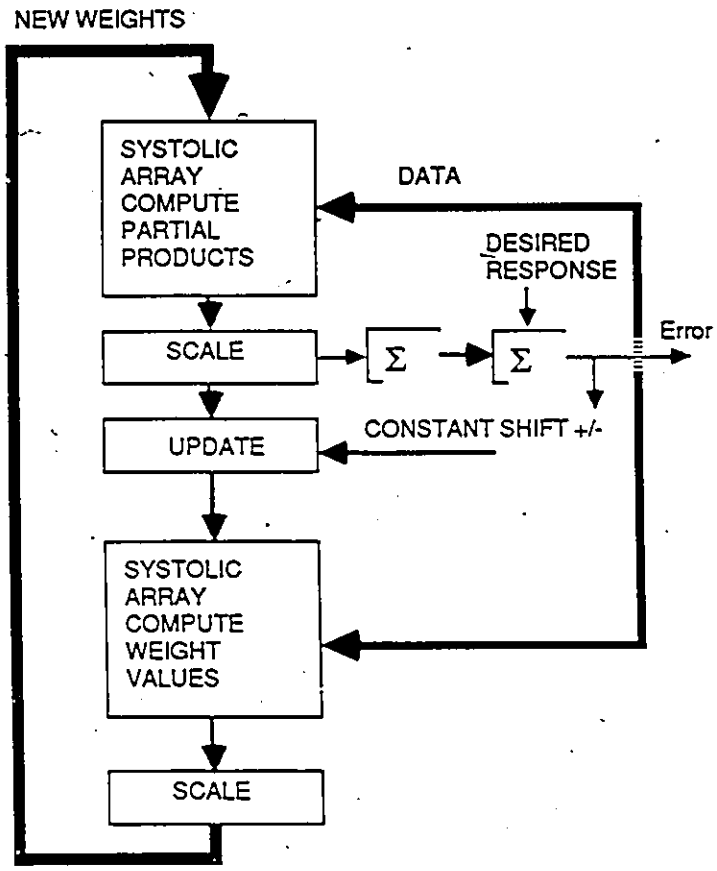


Figure 4.6: Systolic LMS algorithm

It must be noted that the update equation within this system takes the same form as a conventional LMS algorithm. Thus the update and the loading of the coefficients occurs in parallel. Also there is a initial delay between the convolution and the corresponding update. If we assume that the time required for the convolution period is (t) then the time required for the weight computations by the second array will also be (t) since both arrays are identical. Furthermore we can assume that the error computation and the coefficient-partial products loading also each require (t) sec. Then the initial delay in the system will be 3(t). While loading the arrays with coefficients the data input should be zero in order to initialize the arrays, thus every third sampling period we should insert a sample of zeros.

Chapter 5

Conclusion

5.1 Conclusion of the Thesis

It has been seen in this thesis that systolic arrays could be used very effectively in noise cancelling or in general in adaptive filtering. In Chapter one the fundamentals of adaptive filtering have been discussed which later turned out to be essential in understanding and linking QR decomposition with systolic arrays to adaptive theory. Chapter two presented the concept of systolic arrays as a simple means for implementing signal processing algorithms into VLSI technology. The advantages of QR decomposition over the matrix inversion method have been discussed and existing triangular array based on Givens rotations have been studied. Different parts of the array have been identified and explained, and it was shown that this array in conjunction with a systolic back substitute array and an FIR filter could be used in adaptive noise cancelling. Further development into the

triangular structure which eliminated the need for the back substitute array and the FIR filter has been studied [McWhirter 83]. This derivation was quite complex in nature and was based on the ability to partition the rotating matrix Q into its building components. These components have been identified and explained on how they relate to the current inputs and previously stored products. Finally this structure has been linked to the RLS algorithm in terms of its outputs (i.e. showing that the output at the bottom of the array is the a posteriori error which is the product of the conversion factor and the a priori error). Original simulations for all the outputs of the systolic array and the LMS algorithm have been presented.

The first conclusion arising from this chapter is that the exponential weighting has a drastic effect on the performance of the array in terms of self noise and the speed of convergence. Also when using exponential weighting there is a need to rescale the output error in proportion to λ^{2N}, this fact has been proven and discussed in the thesis. We found that the self noise (misadjustment) in the array is proportional to λ^2 and a closed form expression for the misadjustment in the systolic array was presented. It is interesting to see that although the systolic array has been linked to the RLS algorithm , its performance is different from a conventional RLS algorithm. For the same value of the forgetting factor (thus equal speed of convergence in theory) the systolic array will produce twice as much self noise as the conventional RLS algorithm, and vice versa for equal amount of misadjustment for both methods the systolic array will have slower convergence. Furthermore since the systolic array has a $2N$ clock cycles delay time, typical to the systolic structures, its initial convergence time is pro-

portional to $4N$, unlike the RLS which converges in $2N$ clock cycles. We can conclude that the systolic QR decomposition can enjoy some of the advantages typical to the RLS algorithm, however, not all of them at the same time, leaving its performance inferior to the conventional RLS. The second conclusion comes from the comparison to the LMS algorithm. Based on Figure 2.18 when both algorithms produce the same misadjustment, we can say that there is no significant advantage of the systolic array structure over the conventional LMS algorithm and again this is due to the $2N$ delay typical in the systolic structure. It is possible (as can be seen in Figure 2.19) for the LMS algorithm to achieve the same speed as the systolic array with the penalty of increased self noise, same case as the RLS in systolic structure compared to conventional RLS. Also we have found that there is a close connection between the gain constant in the LMS algorithm and the forgetting factor in the systolic array. When $\mu = 1 - \lambda$ both algorithms will produce the same misadjustment. We must remember however, that the above conclusions were made under the assumption of equal eigenvalues of the covariance matrix, under this conditions the LMS will have its best performance. Thus if this condition is not met the advantage of the systolic arrays will become more significant. The final conclusion of this chapter is that the QR decomposition with systolic arrays lies somewhere in between the RLS algorithm and the LMS algorithm in terms of speed and misadjustment.

Chapter three deals with possible implementation of the systolic arrays in hardware. We started by presenting the order of operations within the array with the conclusion that most of the operations could be done in

parallel, the internal structure of the cells has also been proposed. A modification to this structure makes the use of the look up table very attractive since the values in the ROM can be bounded between 0 and 1 regardless of the input signal. The square root operation was a very limiting factor in the previous structure since the values of the square root operation were dependent upon the input signal characteristics. Another very simple array derived from the Gram Schmidt orthogonalization has been presented, this array does not require the square root operation [McWhirter 86]. The array was compared to the QR decomposition with Givens rotations with the conclusion that the GSO array has the same performance as the Givens array after convergence. Some simulations supporting this conclusion were presented. The final conclusion of this chapter is that the GSO has the simplest structure with equal performance to Givens rotations and is the best candidate for hardware implementation.

Due to the conclusion drawn from the comparison in chapter two, the idea of implementing the LMS in systolic structure was initiated. The LMS is a very efficient algorithm (even compared to RLS) with quite a simple structure, and in the first part of Chapter four it was successfully implemented with the systolic convolution array. We have presented three possible arrangements for the LMS algorithm.

1. Parallel update with serial access.

2. Serial update with serial access.

3. Parallel update every N^{th} clock cycle with serial access.

In the second part of this chapter the update part namely the LMS itself was implemented in a systolic structure. This was done with the help of bit level systolic arrays and the distributed arithmetic method. This array is a very new method for adaptive filtering developed by the author and further studies are necessary in order to evaluate the efficiency of this structure. Particularly the need of simulation and synchronization of this structure open the door for a new study.

Appendix A

Direct extraction of the error signal from triangular array in Figure 2.1

This derivation is tedious in nature (matrix form) [Mcwhirter 83] and few of the variables used in the analysis are not explicitly accessible. These can be referred to as dummy variables (all elements of the matrix $\hat{\mathbf{Q}}$). The reason for this is that we do not store the rotating matrix $\mathbf{Q}$ since we are only interested in the factored autocorrelation matrix and crosscorrelation vector. The components that build the matrix $\hat{\mathbf{Q}}$ thus consecutive s_k and c_k are lost after they have been passed through the array. Using (2.2) we

can write;

$$\mathbf{Q}(n)\Lambda\bar{\mathbf{X}} = \begin{bmatrix} \hat{\mathbf{R}}(n) \\ ---- \\ 0 \end{bmatrix} \tag{A.1}$$

where $\hat{\mathbf{R}}(n)$ is an upper triangular matrix and $\mathbf{Q}(n)$ is the rotating matrix of all the data $\bar{\mathbf{X}}$ composed of individual vectors $\mathbf{X}$. In a similar way;

$$\mathbf{Q}(n)\Lambda\bar{d} = \begin{bmatrix} \mathbf{U}(n) \\ ---- \\ \mathbf{V}(n) \end{bmatrix} \tag{A.2}$$

$\mathbf{U}(n)$ is the accumulated crosscorrelation factor in the right hand side of the array as shown in Figure 2.2. $\mathbf{V}(n)$ is the rotated $\bar{d}$ that emerges at the bottom of the right hand side of the array. Thus $\mathbf{V}(n)$ is a growing vector, as the new output that emerges at the bottom of the array becomes the last element on the vector $\mathbf{V}(n)$. For example after 8 clock cycles $\mathbf{V}(n)$ will have 8 elements. In future development, however, we will only be interested in the last element on this vector. We can say that the vectors $\mathbf{U}(n)$ and $\mathbf{V}(n)$ describe completely the desired input as it passes through the array. The matrix $\mathbf{Q}(n)$ is partitioned as well, since some of its components rotate the stored vector $\mathbf{U}$, and some are applied to $\mathbf{V}$;

$$\mathbf{Q}(n) = \begin{bmatrix} \mathbf{P}(n) \\ ---- \\ \mathbf{S}(n) \end{bmatrix} \tag{A.3}$$

Now we can rewrite the elements in A.2 as;

$$\mathbf{U}(n) = \mathbf{P}(n)\Lambda\bar{d} \tag{A.4}$$

$$\mathbf{V}(n) = \mathbf{S}(n)\Lambda\bar{d} \tag{A.5}$$

The error vector $\mathbf{E}(n)$ is composed of consecutive errors due to input vector $\mathbf{X}$ and desired response d, thus e_n ($e_n = e(n)$) is the most recent error and e_1 is the error produced at the first clock cycle.

$$\mathbf{E}(n) = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} \tag{A.6}$$

or in terms of the index of performance defined by (1.23) in section 1.4 an the rotating matrix of all the data $\bar{\mathbf{X}}(n)$;

$$\mathcal{E} = \|\mathbf{Q}(n)\Lambda\mathbf{E}(n)\| = \left\| \begin{bmatrix} \mathbf{U}(n) \\ \hline \mathbf{V}(n) \end{bmatrix} - \begin{bmatrix} \hat{\mathbf{R}}(n) \\ \hline 0 \end{bmatrix} \mathbf{W}(n) \right\| \tag{A.7}$$

from the expression above we deduct that for the index of performance to be at its minimum $\mathbf{U}(n) - \hat{\mathbf{R}}(n)\mathbf{W}(n) = 0$ must be true, that is equivalent to the factored normal equation in (2.5). It is possible to show, using the above definitions, how the error e_n, which is the last element of the vector $\mathbf{E}(n)$, can be derived directly without the need to solve this equation. To derive the error start with;

$$\Lambda\mathbf{E}(n) = \Lambda\bar{d} - \Lambda\bar{\mathbf{X}}\mathbf{W}(n) \tag{A.8}$$

with the definitions;

$$\Lambda\bar{\mathbf{X}} = \mathbf{Q}^T(n) \begin{bmatrix} \hat{\mathbf{R}}(n) \\ \hline 0 \end{bmatrix} = \mathbf{P}^T(n)\hat{\mathbf{R}}(n) \tag{A.9}$$

and;

$$\Lambda \bar{d} = \mathbf{Q}^T(n) \begin{bmatrix} \mathbf{U}(n) \\ \hline \mathbf{V}(n) \end{bmatrix} = \mathbf{P}^T(n)\mathbf{U}(n) + \mathbf{S}^T(n)\mathbf{V}(n) \qquad (A.10)$$

Substituting the above expressions for $\bar{\mathbf{X}}$ and $\bar{d}$ in A.8 yields the following result;

$$\Lambda \mathbf{E}(n) = \mathbf{S}^T(n)\mathbf{V}(n) \qquad (A.11)$$

The rotating matrix $\mathbf{Q}(n)$ of all the input data $\bar{\mathbf{X}}$ ($\bar{\mathbf{X}}$ is composed of consecutive input vectors $\mathbf{X}$) can be related to the rotating matrix of the current input vector $\mathbf{X}$ by;

$$\mathbf{Q}(n) = \hat{\mathbf{Q}}(n) \begin{bmatrix} \mathbf{P}(n-1) & | & 0 \\ \hline \mathbf{S}(n-1) & | & 0 \\ \hline 0 & | & 0 \end{bmatrix} \qquad (A.12)$$

where the matrix $\hat{\mathbf{Q}}(n)$ was defined as in (2.12) and takes the form;

$$\hat{\mathbf{Q}}(n) = \begin{bmatrix} \mathbf{G}(n) & | & 0 & | & \mathbf{g}(n) \\ \hline 0 & | & \mathbf{I} & | & 0 \\ \hline \mathbf{d}^T(n) & | & 0 & | & \hat{\gamma} \end{bmatrix} \qquad (A.13)$$

The matrix $\mathbf{G}(n)$ and the vectors $\mathbf{g}(n)$ and $\mathbf{d}^T(n)$ are composed from the sequence of Givens rotations s_k and c_k that eliminated the vector $\mathbf{X}$. For

example for the case when $N = 4$ (Equivalent to a 4 tap FIR filter):

$$\mathbf{G}(n) = \begin{bmatrix} c_1 & 0 & 0 & 0 \\ -s_1 s_2 & c_2 & 0 & 0 \\ -s_1 s_3 c_2 & -s_3 s_2 & c_3 & 0 \\ -s_4 s_1 c_3 c_2 & -s_4 s_2 c_3 & -s_4 s_3 & c_4 \end{bmatrix} \tag{A.14}$$

$$\mathbf{g}(n) = \begin{bmatrix} s_1 \\ s_2 c_1 \\ s_3 c_2 c_1 \\ s_4 c_3 c_2 c_1 \end{bmatrix} \tag{A.15}$$

$$\mathbf{d}^T(n) = \begin{bmatrix} -s_1 c_4 c_3 c_2 & -s_2 c_4 c_3 & -s_3 c_4 & -s_4 \end{bmatrix} \tag{A.16}$$

$$\hat{\gamma}(n) = c_4 c_3 c_2 c_1 \tag{A.17}$$

The current input vector $\mathbf{X}$ is related to the previously stored $\hat{\mathbf{R}}(n-1)$ by the matrix $\hat{\mathbf{Q}}(n)$

$$\hat{\mathbf{Q}}(n) \begin{bmatrix} \lambda \hat{\mathbf{R}}(n-1) \\ - - - - \\ 0 \\ - - - - \\ \mathbf{X}^T \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{R}}(n) \\ - - - - \\ 0 \\ - - - - \\ 0 \end{bmatrix} \tag{A.18}$$

in the same manner the current input d relates to the previously stored vector $\mathbf{U}(n-1)$ by,

$$\hat{\mathbf{Q}}(n)\begin{bmatrix} \lambda\mathbf{U}(n-1) \\ ---- \\ \lambda\mathbf{V}(n-1) \\ ---- \\ d \end{bmatrix} = \begin{bmatrix} \mathbf{U}(n) \\ ---- \\ \mathbf{V}(n) \end{bmatrix} \tag{A.19}$$

Carrying out the multiplication in A.19 will yield,

$$\mathbf{V}(n) = \begin{bmatrix} \lambda\mathbf{V}(n-1) \\ ---- \\ \hat{\alpha} \end{bmatrix} \tag{A.20}$$

$$\hat{\alpha} = \lambda\mathbf{d}^T(n)\mathbf{U}(n-1) + \hat{\gamma}d \tag{A.21}$$

Now the error in A.11 can be written in terms of the product $\mathbf{S}^T(n)\mathbf{V}(n)$;

$$\mathbf{S}^T(n)\mathbf{V}(n) = \lambda\begin{bmatrix} \mathbf{S}^T(n-1)\mathbf{V}(n-1) \\ --------- \\ 0 \end{bmatrix} + \hat{\alpha}\begin{bmatrix} \mathbf{P}^T(n-1)\mathbf{d}(n) \\ --------- \\ \hat{\gamma}(n) \end{bmatrix} \tag{A.22}$$

It follows that the last element or the current error e_n on the error vector $\mathbf{E}(n)$ is given by;

$$e(n) = \hat{\alpha}(n)\hat{\gamma}(n) \tag{A.23}$$

References and Bibliography

Ardalan S.H. and S.T. Alexander, 'Fixed-Point Roundoff Error Analysis of the Exponentially Windoewd RLS Algorithm for Time-Varying Systems,' IEEE Trans. on ASSP, vol. 35, no. 6, June 1987, pp. 770-783.

Broomhead D.S et at.,'A Practical Comparison of the Systolic and Wavefront Array Processing Architectures,' Proc. IEEE Workshop on VLSI Signal Processing, Los Angeles, 1984, pp. 375-386

Cowan C.F.N et al., 'A Digital Adaptive Filter Using a Memory Accumulator Architecture: Theory and Realization,' IEEE Trans. on ASSP, Vol. 31, No. 3, June 1983, pp. 541-549

Cowan C.F.N et al, 'A New Digital Adaptive Implementation Using Distributed Arithmetic Techniques', IEE proc., vol. 128, Pt. F, No. 4, 1981.

Eleftheriou, E., Falconer D.D., 'Tracking Properties and Steady State Performance of RLS Adaptive Filter Algorithms,' Report SCE-84-14, Department of Systems and Computer Engineering, Carleton University, Ottawa, 1984.

Gentleman, W.M., 'Least Squares Computations by Givens Transformations without Square Roots,' J. Inst. Maths. Applics.1973, vol. 12, pp. 329-336.

Gentleman, W.M., and H. T. Kung, 'Matrix Triangularization by Systolic Arrays,' Proc. SPIE., vol. 298, Real Time Signal Processing IV, 1981, pp. 298-303.

Givens, W., 'Computation of Plane Unitary Rotations Transforming A General Matrix to Triangular Form,' J. Soc. Indust. Appl. Math. Vol. 6, No. 1, 1958.

Haykin, S., Adaptive Filter Theory, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

Kalman, R.E., 'A New Approach to Linear Filtering and Prediction problems,' Trans. ASME J. Basic Engineering, vol. 83, pp. 95-108

Kung, H.T., 'Why Systolic Architectures?,' IEEE Computer, vol. 15, 1982, pp. 37-46.

Kung, H.T., 'Systolic Algorithms and their Implementation,' Proc. of the Seventeenth Annual Hawaii Inter. Conf. on System Sciences 1984, pp. 5-11.

Kung, S.Y., Arun, K.S., Gal-Ezer, R.J., and Bhaskar Rao, D.V., 'Wavefront Array Processor: Language, Architecture, and Applications,' IEEE Trans. on Computers, Special Issue on Parallel and Distributed Computers, 1982, pp. 1054-1066.

Kung, S.Y., H.J. Whitehouse, and T. Kailath, eds. VLSI and Modern Signal Processing, Prentice-Hall, Englewood Cliffs, N.J., 1985.

Kung, S.Y., 'VLSI Array Processors,' IEEE ASSP Magazine, vol. 2, July 1985, pp. 4-22.

Kretschmer, Jr. F.F, and, L.L. Lewis., 'A Digital Open-Loop Adaptive Processor,' IEEE Trans. on Aerospace and Electronic Systems, vol. 14 No. 1, January 1978, pp. 165-171.

Lawson, C.L., and R.J. Hanson, Solving Least Squares Problems, Prentice-Hall, Englewood Cliffs, N.J. , 1974.

Ling, F., and J.G. Proakis., 'Nonstationary Learning Characteristics of Least Squares Adaptive Estimation Algorithms,' Proc. ICASSP, San Diego, Calif., 1984, pp. 30.3.1-30.3.4.

Makhoul, J., 'Linear Prediction: A Tutorial Review,' Proc. IEEE, 1975, vol. 63, pp. 561-580.

McCanny, J.V., and, J.G. McWhirter, 'Implementation of Signal Processing Functions using 1-bit Systolic Arrays,' Electron. Letts., vol. 18, no. 6, 1982, pp. 241-243.

McCanny, J.V., J.G Mcwhirter, and D. Wood., 'Optimised Bit Level Systolic Array for Convolution,' IEE Proc., Vol. 131, Pt. F, No. 6, October 1984.

McWhirter, J.G., 'Recursive Least Squares Minimization Using a Systolic Array,' Proc. SPIE, vol. 431, Real Time Signal Processing VI, 1983, pp. 105-112.

McWhirter, J.G. et al, 'A Comparison of the Gram Schmidt and QR Decomposition Algorithms for Adaptive Beamforming,' Parallel Algorithms and Architectures, M. Cosnard et al. (editors), Elsevier Science Publishers B.V. (North Holland), 1986.

Mead, C. and L. Conway, Introduction to VLSI Systems, Addison Wesley, Reading, Mass., 1980.

Monzigo, R.A., and, T.W. Miller., Introduction to Adaptive Arrays, John Wiley & Sons, New York, 1980.

Morf, M., T. Kailath, and L. Liung, 'Fast Algorithms for Recursive Identification,' Proc. 1977 Conf. Decision and Control, Clearwater Beach, Fla., pp. 916-921.

Papoulis, A., Probability Random Variables, and Stochastic Processes, McGraw-Hill Book Company, 1984.

Qureshi, S.U.M., 'Adaptive Equalization,' Proc. IEEE, vol. 73, No. 9, September 1985, pp. 1349-1387.

Savov, E. and Steenaart, W., 'The LMS Algorithm Applied to High Speed Adaptive Digital Signal Processing,' Proc. Symposium on Circuits and Systems, IEEE 1987, Philadelphia, Pa, Vol. 3, pp. 430-433.

Speiser, J.F., 'Linear Algebra Algorithms for Matrix Based Signal Processing,' Proc. SPIE, vol. 614, Highly Parallel Signal Processing Architectures, 1986, pp. 2-7.

Strang, G., Linear Algebra and its Applications, Academic Press Inc., New York, New York, 1980.

Walicki, J.S., Andrews M,, 'Parallel Implementation of the LMS algorithm,' ICPP, Aug. 1985, St Charles, IL., pp. 312-316

Ward, C.R. et al, 'Application of a Systolic Array to Adaptive Beamforming,' IEE Proc., vol. 131, pt. F, No. 6, October 1984, pp. 638-645.

Ward, C.R. et al, 'A Novel Algorithm and Architecture for Adaptive Digital Beamforming,' IEEE Trans. on Antennas and Propagation, vol. 34, No. 3, March 1986, pp. 338-346.

Watkins, D.S., 'Understanding the QR Algorithm,' SIAM Review, vol. 24, No. 4, October 1982, pp. 427-440.

Widrow, B. et al., 'Adaptive Noise Cancelling: Principles and Applications,' Proc. IEEE, vol. 64, 1975, pp.1151-1162.

Widrow, B. and M.E. Hoff, Jr., 'Adaptive Switching Circuits,' IRE WESCON Conv. Rec., 1960, Part 4, pp.96-104.

Widrow, B. , and S.D. Stearns., Adaptive Signal Processing, Prentice Hall, Englewood Cliffs, N.J., 1985.

Wiener, N., Extrapolation, Interpolation and Smoothing of Stationary Time Series, with Engineering Applications, MIT Press Cambridge, Mass. 1949.

Young, P., Recursive Estimation and Time-Series Analysis, Springer Verlag, 1984.

Yuen, S.M., 'Algorithm and Systolic Architecture for Solving Gram Schmidt Orthogonalization (GSO) systems,' International Journal of Mini and Microcomputers, Vol. 7, No. 2, 1985.