The design and implementation of a microprocessor controlled adaptive filter

Ahmed, Kadrya Mohammed

THE DESIGN AND IMPLEMENTATION OF A MICROPROCESSOR CONTROLLED ADAPTIVE FILTER

by

Kadrya Mohammed Ali Ahmed, B.Sc.

A thesis submitted in accordance with the regulations for the degree of Doctor of Philosophy in the University of Durham
Department of Applied Physics & Electronics

1986

Thesis
1986/AHM

The Design and Implementation of a Microprocessor Controlled Adaptive Filter

Kadrya Mohammed Ali Ahmed

ABSTRACT

This thesis describes the construction and implementation of a microprocessor controlled recursive adaptive filter applied as a noise canceller. It describes the concept of the adaptive noise canceller, a method of estimating the recieved signal corrupted with additive interference (noise). This canceller has two inputs, the primary input containing the corrupted signal and the reference input consisting of the additive noise correlated in some unknown way to the primary noise. The reference input is filtered and subtracted from the primary input without degrading the desired components of the signal. This filtering process is adaptive and based on Widrow-Hoff Least-Mean-Square algorithm. Adaptive filters are programmable and have the capability to adjust their own parameters in situations where minimum a piori knowledge is available about the inputs. For recursive filters, these parameters include feed-forward (non-recursive) as well as feed-back (recursive) coefficients. A new design and implementation of the adaptive filter is suggested which uses a high speed 68000 microprocessor to accomplish the coefficients updating operation. Many practical problems arising in the hardware implementation are investigated. Simulation results illustrate the ability of the adaptive noise canceller to have an acceptable performance when the coefficients updating operation is carried out once every N sampling periods. Both simulation and hardware experimental results are in agreement.

ACKNOWLEDGEMENTS

I wish to express my gratitude to Prof. C.T. Spracklen, at Engineering Department, Aberdeen University, for his supervision, encouragement and securing the required facilities. I am indebted to Dr. B.J. Stanier at Durham University for his guidance, constructive suggestions, invaluable advice and kindly help throughout the period of this work.

I would also like to thank Prof. G.G. Roberts who made available to me the facilities of the Applied Phyisics and Electronics Department at Durham University. I like to give my thanks to the members of the workshop at Durham and Aberdeen University for providing their advice and technical assistance.

I greatfully acknowledge with thanks the valuable discussions and help of my colleagues in the Digital Electronics Group during the past years. I am grateful to the staff of the Computer Unit and the Science Library at Durham and Aberdeen Universities for their co-operation and friendly help. I also give my deepest thanks to Dr. Patrick Hughes at British Telecom Research Laboratories, Dr. Colin Cowan and Bernard Mulgrew at Edinburgh University who gave me the opportunity for interesting and helpful discussions.

I would also like to express my sincerest thanks to the State of Qatar and the University of Qatar for the award of a research studentship and financial support.

Finally, a special note of appreciation and gratitude to my parents, brothers and sisters for their encouragement, motivation, moral and intuitive support which provided me with a relaxed and secure life.

I DEDICATE THIS WORK TO MY BELOVED PARENTS, WHO HAVE INSTILLED IN ME A LOVE OF THE PURSUIT OF EXCELLENCE. BY THEIR CONSTANT SUPPORT AND GUIDANCE THEY HAVE ENCOURAGED ME TO PERSEVERE IN MY GOALS, BROTHERS AND SISTERS.

Glossary of Terms

ADC	Analogue to Digital Converter
ALE	Adaptive Line Enhancer
ALP	Adaptive Linear Prediction
ALU	Arithmetic Logic Unit
ANC	Adaptive Noise Canceller
CCD	Charge Coupled Device
CPU	Central Processing Unit
DAC	Digital to Analogue Converter
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
EPROM	Eraseable Programmable Read Only Memory
FAD chip	Digital Filter and Detect chip
FIR	Finite Impulse Response
IC	Integrated Circuit
IIR	Infinite Impulse Response
LMS	Least Mean Square
LSB	Least Significant Bit
LSI	Large Scale Integration
LSW	Least Significant Word
M	Misadjutment
MMSE	Minimum Mean Square Error
MSB	Most Significant Bit
MSE	Mean Square Error
MSW	Most Significant Word
MUX	Multiplexer
NRCOEFF	Non-Recursive Coefficient
PIA	Peripheral Interface Adapter

PROM	Programmable Read Only Memory
RAM	Random Access Memory
RCOEFF	Recursive Coefficiet
ROM	Read Only Memory
SAW	Surface Acoustic Wave
SNR	Signal to Noise Ratio
SYNC pulse	Synchronisation pulse
VLSI	Very Large Scale Integration

CONTENTS

CHAPTER -1-

Background

1.1 Introduction

In any communications system, the transmission of signals from the transmitter to the receiver involves their contamination with noise. This is the basic problem of communication and limits the capability of the system.

By noise we mean unwanted or unpredictable signals that corrupt the message. Noise can be classified according to the source into two categories:

(a) internal noise, which is generated by components within the communication system.

(b) external noise, which includes man-made noise and extraterrestrial natural sources (1,2).

To suppress this noise, signal processors such as filters can be used.

In the decade 1960-1970 high speed digital computers were developed and became widely available for serious research and development work. Consequently it became possible to use the theoretical basis of digital signal processing, such as Fourier analysis, waveform sampling, Z-transform, etc., in digital signal design (3,4).

This background provided the impetus for the introduction of digital signal processors as a means for implementing digital filters.

The term digital signal processing (DSP) implies the description of a complete set of operations such as arithmetic calculation and numerical manipulations. These represent the processing of the signal considered as a sequence of numbers or symbols in order to produce a form which is in some sense more desirable. Many different functions can be accomplished in this way; spectral analysis, filtering (which is of particular interest) transcoding, modulation, detection, and estimation and extraction of parameters by using a suitable digital computer (5,6).

DSP is concerned with signals or systems that are the discrete-time counterpart of the more familiar continuous-time systems. DSP has many applications in a very wide range of fields such as the analysis of biomedical signals, vibration-analysis, picture processing, analysis of seismic signals and speech analysis. However, telecommunication forms a very important field of applications which provides a major stimulus for research and development (4,5).

DSP has become an established method of filtering electrical waveforms, and the associated theory of discrete-time systems can often be employed in a number of disciplines (4).

The term 'filter' implies any frequency selective device or processor which passes certain ranges of frequencies and rejects others. Filters can be classified according to the frequency ranges they pass or reject into; low-pass, high-pass, pass-band, and stop-band filters. For example, a low-pass filter passes a low frequency input signals below a certain value, and attenuates the signals of frequency above this value (7).

Filters can also be classified according to their signal type into:

analogue (or continuous filters).

and digital filters.

1.2 Comparison of Digital and Continuous Filters

Digital and continuous filters have exactly the same aim of suppressing a noisy signal and producing one with less distortion, but the physical realization is different, since continuous filters are constructed from linear electrical components, such as resistors, capacitors and inductors, and linear continuous filter theory is based on linear differential equations and the Laplace transform. Digital filters however are based on linear difference equations and the Z-transform (which is a special form of the Laplace transform, which will be described in detail in the next chapter). It can be shown that digital filters are mathematically equivalent to continuous filters with sampled data inputs and outputs.

Digital filters have several advantages over continuous filters and some of these are given below:

(1) the absence of impedance-matching problems.

(2) the frequency response characteristics can be made to be relatively close to the ideal and can be changed by varying the stored coefficients.

(3) the possibility of implementing any type of filter with the same hardware by using multiplexing and frequency transformation.

(4) adaptive filtering is relatively simple to achieve.

(5) there are no drift problems as a result of realizing stable filters with very high 'Q's'.

(6) low cost.

However, in spite of all these advantages, digital filters also have limitations. The limited word length available on the digital computer leads to quantization errors, which will be investigated in some detail in chapter 2. Much work has been carried out to overcome these errors (4,7,9).

Because of the availability of the digital computer as a research tool in all branches of science and technology, and the existance of analogue-to-digital converter (ADC) and digital-to-analogue converter (DAC), digital filters became widely used. Many established continuous-time filter systems have been replaced by equivalent digital filter systems because of their advantages.

A digital filter is defined (7,9) as the computational process or algorithm by which a sampled signal or sequence of numbers, acting as an input, is transformed into a second sequence of numbers, termed the output, using digital components as the basic elements for implementation. The process can be any arbitrary filtering operation, such as low-pass filtering, high-pass filtering, pass-band filtering or stop-band filtering etc..

The digital filter may be represented by a linear difference equation which defines the output signal as a function of the present input sample and any number of previous input and output samples (4,7).

Digital filters can be fixed or adaptive. The optimal design of fixed filters is based on a complete a priori knowledge of both the signal and the interference (noise). Adaptive filters, on the other hand, require much less prior knowledge in their design and have the ability to adjust their own parameters during operation to optimize their performance. The adjustable parameters of adaptive filters are called the weights (impulse response). These weights are usualy updated by a least mean square (LMS) algorithm which adjusts the filter weights so that the filter output is an estimate of the signal (10,11).

In this project an adaptive filter is used as a noise canceller in which a corrupted signal passes through a filter that tends to supress the noise while leaving the signal relatively unchanged (11).

Fig. 1.1 illustrates the block diagram of a basic adaptive noise canceller (ANC). It has two inputs, the desired response input d_j (primary input) containing the corrupted signal and the input signal x_j (reference input) containing noise correlated in some unknown way with the primary noise. The reference input is adaptively filtered and subtracted from the primary input to obtain the signal with less destortion. The objective of the system is to minimize the error e_j between the filter output y_j and d_j in the LMS manner by iteratively updating the filter weights in accordance with the LMS algorithm (11,12), this is discussed in detail in chapter 3.

To realize an adaptive filter in real time, the digital technique must be fast enough to complete the computational process within the available time.

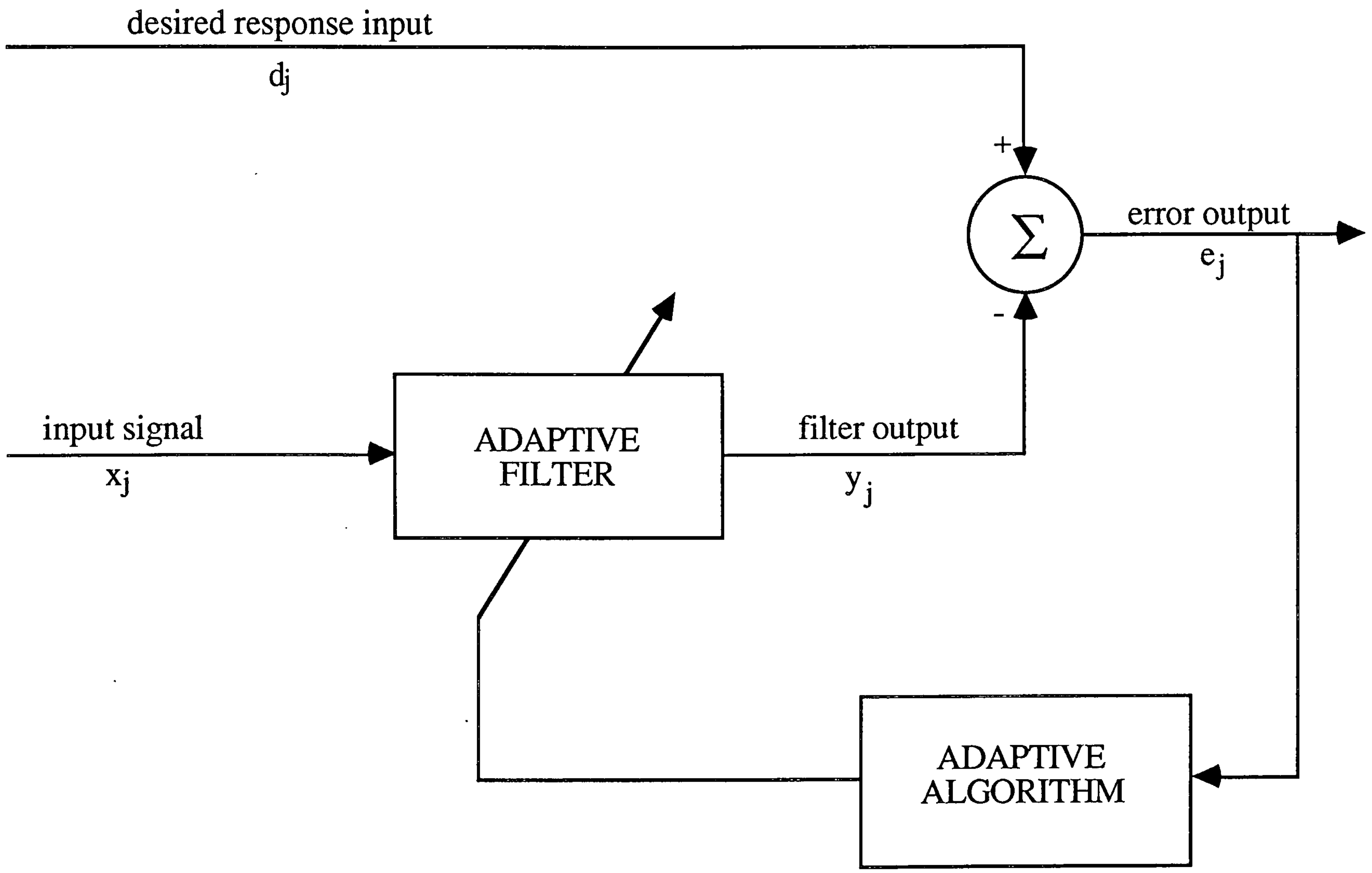


Fig .1.1 Basic adaptive filter

Adaptive filters are versatile signal processing elements which have a wide range of applications where only a limited a priori knowledge of the expected signal and noise is available. These applications include:

(a) Noise canceller

In noise cancelling system the additive noise or interference will be rejected from a waveform containinig a signal of interest. Noise cancelling is a variation of optimal filtering that is highly advantageous in many applications. These applications include for intance the cancelling of 60-Hz interference in electrocardiography, the donor electrodiogram in heart transplant electrocardiography and cancelling noise in speech signals which plays an important roll in our communications-oriented society. Much work has been carried out to remove the unwanted noise components from the speech signals (11,13,14,15,16).

(b) Channel equalizer

The main problem in digital communications is the recovery of the message which, when sent over the channel, is distorted with noise and intersymbol interference due to the nonlinear phase and amplitude characteristics of the channel. To overcome these effects the channel equalizer is used. Many methods have been investigated for the optimal implementation of digital channel equalizers (17,18,19,20).

(c) Adaptive line enhancer (ALE)

The ALE is considered as an adaptive digital filter which is designed to remove uncorrelated components (broad-band noise) from its input, while passing any narrow-band signals or components with little attenuation (21,22,23).

(d) Adaptive linear prediction (ALP)

The ALP signal processing has been applied with great success for many problems such as spectral analysis, system modeling and speech encoding. The conventional approach for implementing ALP involves computing and updating the sample covariance matrix for a block of data and then obtaining the predictor coefficients. Morgan and Craic have published an alternative approach which uses the LMS algorithm, to compute the predictor coefficients (24,25).

Adaptive filters can be implemented by a software program on a digital computer with a special interface and by digital hardware. In this project, an IC implemented as a digital filter which is designed as a general purpose device suitable for filtering noisy signals in the audio frequency band is used to implement a 16th order recursive LMS adaptive filter. The adaptive filter is used as a noise canceller, in which a corrupted signal passes through the filter and this coherently supresses the noise while leaving the desired signal relatively unchanged.

In the next chapter, the basic principles of digital filtering will be described. The first part presents the relationship between input&output of the discrete-time system. The second part dicusses the various methods of realizing the recursive and non-recursive digital filters, and quantization errors due to the limited-word length available on a digital computer.

In chapter 3, the basic concept of adaptive noise cancelling, and the so-called Wiener solution to the statistical noise cancelling problems are considered. The LMS adaptive algorithm for updating the feed-forward (non-recursive) and feed-back (recursive) coefficients will be described and the quantization effects due to the limited-word length and finite precision arithmetic of a digital processor, will be investigated.

In chapter 4, a flexible computer simulation program has been developed to investigate the behaviour of the recursive LMS adaptive filter, employed as an ANC, under different conditions and various parameters. The performance of the filter has been discussed in both fixed point and floating point arithmetic representations

Chapter 5 is concerned with describing step by step approach to the design and construction of a 16th order recursive LMS adaptive filter exposing a brief introduction to the 68000 microprocessor and the digital filter IC used in the design. The interfacing considerations and problems arising in the implementation of the filter have also been investigated.

Chapter 6 discusses extensively how the single board 68000 microcomputer can be applied to update the adaptive filter coefficients in order to write one bit of each recursive and non-recursive coefficient into the filter IC every clock cycle.

Chapter 7 describes the implementation of the filter to be used as an ANC and presents the results of cancelling undesirable interference. This chapter also reflects some light on the effect of the implication of the feed-back in recursive (IIR) filter on its stability.

1.3 History and Development of Adaptive Filters

The earliest work on adaptive noise cancelling was performed by Howells and Applebaum and their colleagues at the General Electric Company between 1957 and 1960. They designed and built a system for antenna sidelobe cancelling that used a reference input derived from an auxillary antenna and a simple two-weight adaptive filter (26).

At the time of this work, only a handful of people were interested in adaptive systems, and development of multiweight adaptive filters was only just begining. Much of this early efforts were proceeding by independent studies in different research organizations. A notable early development occured at Stanford University when Widrow and Hoff devised the LMS adaptive algorithm and the pattern recognition scheme in 1959 (27,28).

Further relevant work being conducted simultaneously at the Institute of Automatics and Telemechanics in Moscow. In Great Britain, in 1961 Gabor and his associates were developing adaptive filters (29).

In the early and middle 1960.s, work on adaptive systems intensified. Many papers on adaptation, adaptive controls, adaptive filtering, and adaptive signal processing appeared in the literature (4,30,31).

The first applicable adaptive filter is credited to Lucky at the Bell Laboratories for his design in 1966 of an equalizer which compansated for distortion in data transmission systems. The first adaptive noise cancelling system at Stanford University was designed and built in 1965 by two students. The purpose was to cancel 60-Hz interference at the output of an electrocardiographic amplifier and recorder.

Since 1965, adaptive noise cancelling has been successfully applied to a number of additional problems as shown in (11).

In 1967 Widrow, et al. (11) proposed a technique for an adaptive digital filter based on the LMS algorithm (or gradient search technique) which has come from the Stanford University pattern recognition work. The main advantage of this algorithm is its computational simplicity for real time processing with little storage which converges toward the optimum solution much more efficiently than do other algorithms.

Various implementations have been discussed and published in the literature. In general, adaptive filters can be implemented in the time domain or the frequency domain. Reed and Feintuch (32) have described the behaviour of a frequency domain adaptive filter configured as a broad-band canceller with white Gaussian inputs. Bershad and Feintuch (23) have presented a mathematical model for predicting the mean weight behaviour of the recursive adaptive filter when used as an ALE in the frequency domain.

Most practical adaptive filters have been realized by computer programs. In recent years Feintuch (33) implemented an adaptive recursive LMS filter in the time domain using two transversal adaptive filters using simulation results to demonstrate its capability (34). Mikhael, et al. (35) has proposed using individual convergence factors to adapt the individual recursive adaptive filter parameters, and then to adjust them in real time, so that their values are kept optimal for each new set of input variables, rather than using the conventional technique, which uses a fixed time constant convergence factor, which is chosen to be the same for all the filter parameters. He has presented computer simulation results which indicate that the individual adaptation approach gives much better results than the conventional approach. Paul (13) has investigated and published two adaptive digital techniques for audio noise cancellation. The first technique, adaptive predictive deconvolution, used an ALP to estimate and cancel correlated noise components of the audio signal. The second technique, adaptive filtering, employs two audio signal inputs.

However, the realization of adaptive filters in hardware has also developed rapidly. The growth of large scale integrated circuits (LSI) decreases the cost and increases the speed of components. Cowan et al. (36) have published a technique for implementing a digital adaptive filter which used no digital multiplication, but instead relies on the use of the operations of memory access, addition and scaling by integer powers of 2. The technique is based on the distributed arithmetic (alternatively read-only memory (ROM)/accumulator) filter architecture originally proposed for a fixed frequency filter implemented by Peled and Liu (37). This technique has the advantage that only standard TTL type logic circuits need to be used without recource to specialised signal processing functions (such as hardware multipliers). Cowan and Grant also published another digital adaptive filter design based on the LMS algorithm which relies heavily on the use of linear digital multipliers (38).

In 1983 Holt and Mullholand (12) published a technique for a high speed micoprogrammed adaptive filter implementation using the AM2900 bit slice device and a hardware multiplier-accumulator. The implementation was based upon a clipped LMS algorithm.

With the continuing advances in digital technology and the availability of LSI components, the adaptive filter is now available on a single chip. South (39) has developed a digital adaptive filter in LSI form for use in the audio band which makes it ideal for solving problems of cancelling unwanted signals in telephony and other fields at low cost.

1.4 Conclusion

The advantages of the adaptive digital filters over the fixed coefficients digital filters, are that they are programmable, so their coefficients (weights) are updated and adjusted in accordance with the incoming signals. They are versatile signal processing elements which can be applied in situations where an absolute minimum knowledge is available about the incoming signals.

The robustness and the simplicity of implementation of the LMS algorithm, make the LMS adaptive filter attractive for many applications. These applications include, noise cancelling, channel equalization, line enhancing and adaptive linear prediction. The rapid advance in LSI and VLSI technology and the desire to provide improved speech communications, make telecommunications a fertile field for the application of adaptive filters.

CHAPTER -2-

Digital Filtering

2.1 Introduction

The early 1970s marked the beginning of a revolution in the electronics world, and computer technology has developed rapidly. With the advent of relatively cheap digital computers and availability of the A/D and D/A converters, powerful digital filters have now become an attractive subject and commonly used. Consequently, the number of practical applications significantly increased.

Some digital filters have found important uses in an increasing number of fields in science and engineering and the required techniques have been developed to achieve the desired filter characteristics.

Many digital filters are fabricated as a single IC making the use of such filter components in commercial systems economically feasible and technically desirable.

Many programmable LSI digital filters have been designed and have been used in many applications (40). British Telecom Research Laboratories (41) have designed an LSI digital filter and detect (FAD) chip as a general purpose device suitable for filtering noisy signals and detecting tones in the audio frequency band.

This chapter describes the basic principles of digital filtering. Two methods of describing the discrete-time systems, namely the Z-transform and state variables technique, will be investigated. The mathematical concept of the Z-Transform, which is the basic mathematical tool of digital filtering, is described in the first section. The linear difference equation, the central element in the concept of the digital filter, is then investigated. Different realisations of recursive digital filters and the considerations that should be taken into account in choosing between them are also presented. Due to the limited word length available on the digital computer, quantization noise occurs, the last section introduces some aspects of these errors.

2.2 Sampled Data Signals

2.2.1 Introduction

The sampling process represents the signal x(t) by its value x(nT) at integral time increments T (called the sampling period) so that the sampled data signal defines values only at certain instants of time. Since it is not possible to feed continuous data into a digital computer, any signal or data input must be represented as a set of samples. A simple model of the sampling process is one which considers that the samples can be acquired by closing a switch at interval times every T seconds for a short time τ seconds as shown in fig.2.1. Referring to fig. 2.1 it is obvious that the switch output is a set of pulses separated by period T with finite width. However, if the pulse width, τ, is negligible compared with interval between successive samples, T, the output of the sampler $x^*(t)$ may be described as a set of impulses with their heights proportional to the values of x(t) at the sampling instants (4,5).

2.2.2 Mathematical Description Using the Dirac "δ" Function

The Dirac function, which is usually referred to as the unit

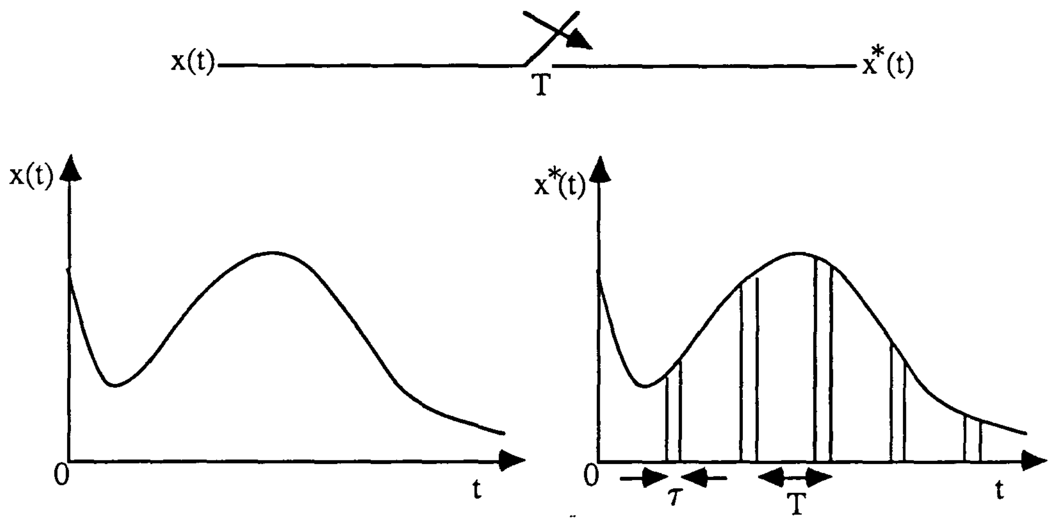


Fig. 2.1 Model of the sampling process

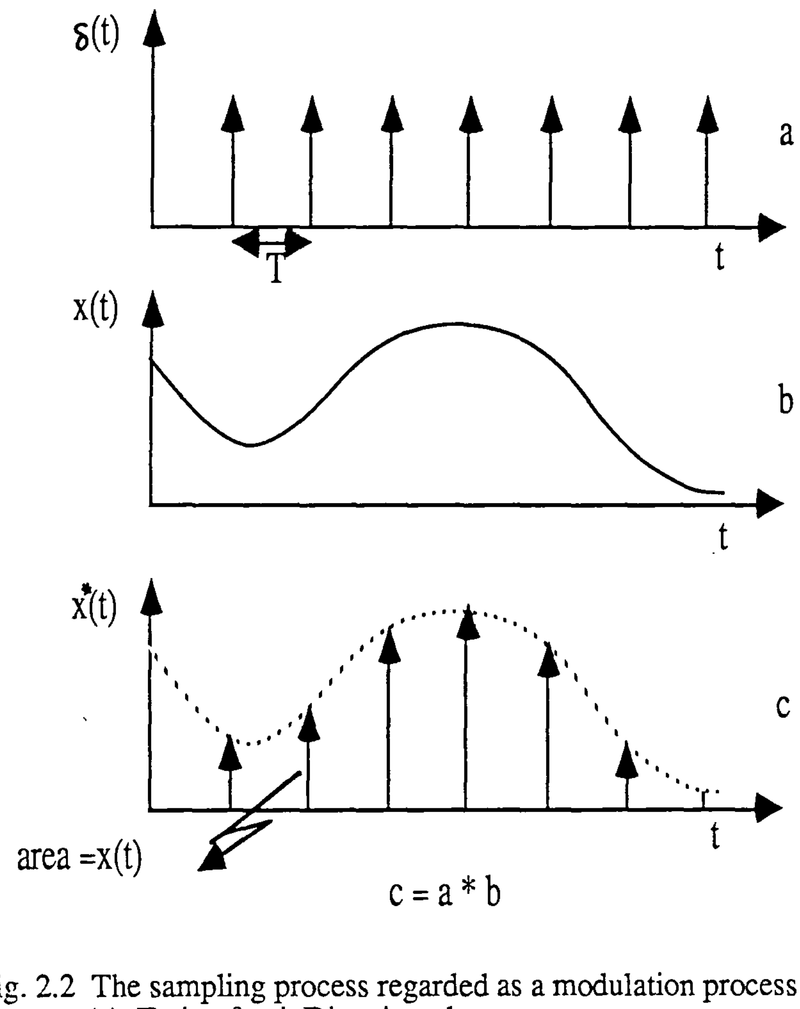


Fig. 2.2 The sampling process regarded as a modulation process
(a) Train of unit Dirac impulses
(b) Continuous signal
(c) Sampled signal

impulse function, is a pulse of extremely small width and unit area. In other words, the product of its width and its mean height is unity, even though its precise shape is undefined.

As shown in fig. 2.2, the ideal sampling function can be represented as a train of unit impulses, and is defined by the equation

$$\delta_T(t)=\sum_{n=-\infty}^{\infty} \delta(t-nT), \qquad 2.2.1$$

where $\delta(t)$ represents a Dirac pulse (unit impulse) occuring at t=0, and $\delta(t-nT)$ is a Dirac function shifted by integral number of T occuring at t=nT. Therefore

$$x^*(t) = x(t)\sum_{n=-\infty}^{\infty} \delta(t-nT) \ , \qquad 2.2.2$$

where x(t) is the original continuous signal and $x^*(t)$ is the sampled signal.

Since the value of x(t) is only known for t=nT, and for a physical system x(t)=0 for t<0, then

$$x^*(t) = \sum_{n=0}^{\infty} x(n)T \ \delta(t-nT) \qquad 2.2.3$$

It is obvious that $x^*(t)$ is a weighted sum of shifted unit impulses, so that x(n)T is the weighting factor of the impulse function $\delta(t-nT)$, as indicated by the value in fig. 2.2.c (4).

Referring to eqn. 2.2.1, $\delta_T(t)$ can be expanded as a Fourier series, that is

$$\delta_T(t) = \sum_{n=-\infty}^{\infty} C_n \ e^{jn\omega_s t}$$

where

$$C_n = (1/T)\int_0^T \delta_T(t) \ e^{-jn\omega_s t} \ dt$$

and ω_s is the sampling frequency equal to $2\pi/T$ rad/sec.

Since the area of the Dirac pulse is unity, then

$$\int_0^T \delta_T(t)\, e^{-jn\omega_s t}\, dt = 1$$

and therefore, $C_n = 1/T$, hence

$$\delta_T(t) = (1/T)\sum_{n=-\infty}^{\infty} e^{jn\omega_s t}$$

and we have seen in fig. 2.2 that for the impulse modulator

$$x^*(t) = \delta_T(t)\, x(t)\ \text{, therfore}$$

$$x^*(t) = 1/T \sum_{n=-\infty}^{\infty} x(t)\, e^{jn\omega_s t} \qquad 2.2.4$$

By taking Laplace transforms and stating the associated shifting theorem, we obtain

$$X^*(S) = \mathcal{L}[x^*(t)] = (1/T)\sum_{n=-\infty}^{\infty} X(S-jn\omega_s)$$

where $S=j\omega$,

therefore

$$X^*(j\omega) = (1/T)\sum_{n=-\infty}^{\infty} X[j(\omega-n\omega_s)] \qquad 2.2.5$$

It is observed from eqn. 2.2.5 that, as a result of impulse-sampling, the frequency spectrum of the signal $x^*(t)$ is the same as the spectrum of the original signal x(t) but is periodic with period ω_s ($2\pi/T$), that is to say, the sampling has introduced a periodicity into the frequency space, which constitutes a fundamental characteristic of the sampled signal shown in fig. 2.3. Referring to fig. 2.3 if the highest frequency component is greater than $\omega_s/2$ (see fig. 2.3.c), a fold-over distortion or aliasing of the frequency response function is introduced, which may be avoided by increasing the sampling frequency. Consequently the original

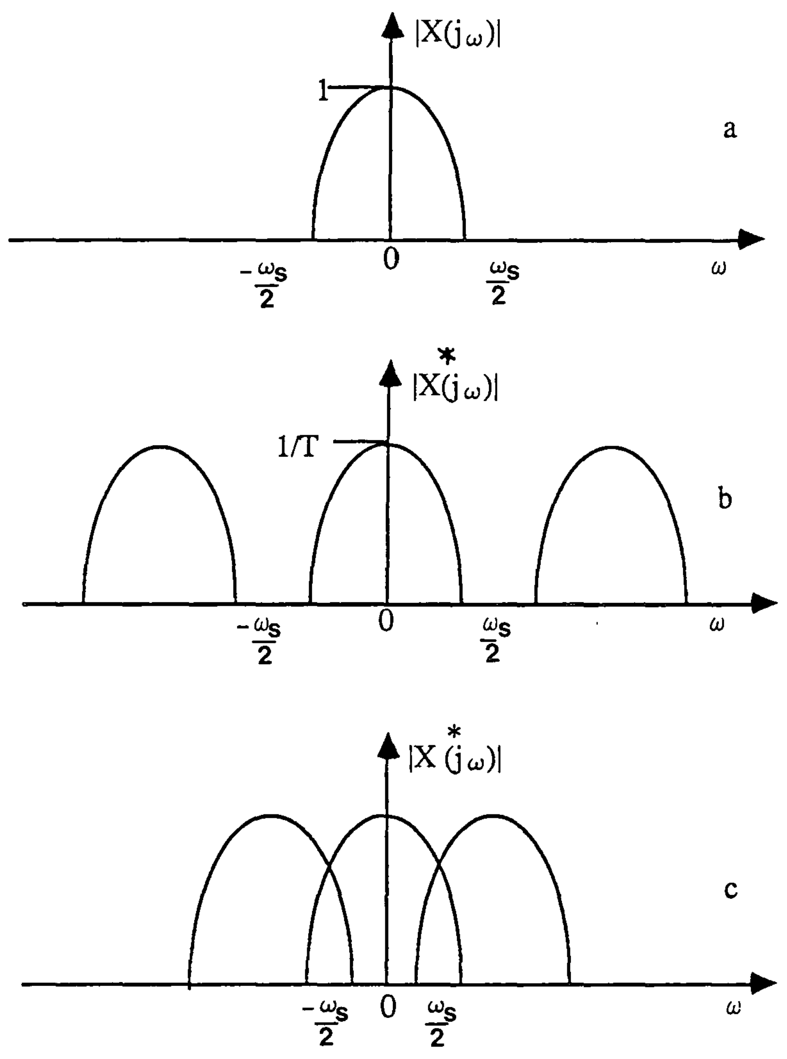


Fig. 2.3 (a) Frequency spectrum of the signal
(b) Frequency spectrum of the sampled signal
(c) Aliasing of frequency specrta

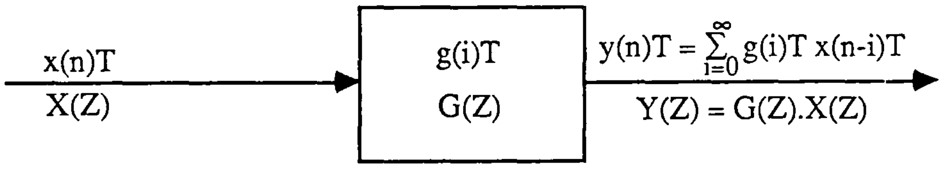


Fig. 2. 4 Digital convolution-summation property

signal cannot be reclaimed from the sampled-data signal. From this, the sampling or Shannon's theorem is derived, in which it specifies the minimum sampling rate for adequate representation of a continuous signal. If the frequency $f=\omega/2\pi$, and to avoid the folding condition, it has been already shown that

$$\omega < \pi/T$$

or

$$f = \omega/2\pi < 1/2T$$

and hence

$$T < \pi/\omega = 1/2f$$

Formally, the sampling theorem may therefore be stated as follows: "A continuous signal which does not contain any component with frequency greater than f Hertz may in principle be recovered from its sampled version, if the sampling interval T is less than 1/2f seconds".

The interval 1/2f is called Nyquist interval and 2f is known as the Nyquist frequency (4,5,7).

2.3 The Z-Transform

In general the analysis and design of linear systems may be carried out by one of two major approaches which relies:

(a) on the use of a transform, such as Laplace and Z-transform and block diagrams.

(b) the state variable technique, which will be discribed in the next section.

As we mentioned before, filters can be classified into analogue filters (continuous-time systems) and digital filters (discrete-time systems).

In continuous-time systems, (42) Laplace transformation is used (**a**) to solve the differential equations, in which the output signal and its derivatives are related to the input signal and its derivatives and also (**b**) to express the behaviour of a filter in terms of a transfer function which describe a signal in terms of a set of poles and zeros in S-plane (S-domain). The Z-transform, on the other hand, is used to describe the properties of a discrete-time systems (sampled data signals) and leads to a useful method of representing the discrete-time systems by either a finite set of poles and zeros in Z-plane (frequency domain representation) or by a linear difference equation (time domain representation).

The Z-transform (4) can be considered as a rule that converts a sequence of numbers into a function of the complex variable Z. The Z-transform of a sampled-data signal may be directly determined from its Laplace transform. Referring to eqn. 2.2.3 we find that

$$x^* = x(0)T\delta(t)+x(1)T\delta(t-T)+x(2)T\delta(t-2T)+\dots$$

The Laplace transform of $x^*(t)$ is given by

$$\mathcal{L}[x^*] = x(0)T+x(1)Te^{-ST}+x(2)Te^{-2ST}+ \dots$$

where S is a complex frequency variable.

We now define the new variable $Z = e^{ST}$ and denote the Z-transform of the signal by X(Z). Hence

$$X(Z) = x(0)T+x(1)TZ^{-1}+x(2)TZ^{-2}+ \dots$$

that is

$$X(Z) = \sum_{n=0}^{\infty} x(n)T\, Z^{-n} \qquad 2.3.1$$

Z is a new complex frequency variable. Thus if $S = \sigma + j\omega$

$$Z = e^{ST} = e^{\sigma T} \cdot e^{j\omega T}$$

The term Z^{-n} implies a time delay of n sampling periods.

Murankami et al. (43) have presented a new complex number-theoretic Z-transform (CNT Z-transform) over a finite ring. It was shown that a digital filter with any desired impulse response can be expressed in terms of the CNT Z-transform. They have found that filters designed on a finite ring have some advantages over the usual Z-transform, i.e. they are errorless and information lossless.

2.4 Z-Transform Properties

1. Linearity :

$$\begin{aligned} X(Z) &= \sum_{n=0}^{\infty} (Ax_1(n)T + Bx_2(n)T)Z^{-n} \\ &= A\sum_{n=0}^{\infty} x_1(n)TZ^{-n} + B\sum_{n=0}^{\infty} x_2(n)Z^{-n} \\ &= A\,X_1(Z) + B\,X_2(Z) \end{aligned}$$

where A and B are constants.

2. Right-Shifting "delay" :

$$\begin{aligned} Y(Z) &= \sum_{n=0}^{\infty} x(n-k)TZ^{-n} = \left[\sum_{n=0}^{\infty} x(n)TZ^{-n}\, Z^{-k}\right] \\ &= X(Z)\, Z^{-k} \end{aligned}$$

3. Left-Shifting :

$$\begin{aligned} Y(Z) &= \sum_{n=0}^{\infty} x(n+k)TZ^{-n} \\ &= Z^k X(Z) - \sum_{n=0}^{k-1} x(n)TZ^{-(n-k)} \end{aligned}$$

4. Convolution-Summation :

The input and output signals of a digital filter are related to each other through the convolution-summation property. Referring to fig. 2.4 and using this property, we have

$$y(n)T = g(0)Tx(n)T+g(1)Tx(n-1)T+g(2)Tx(n-2)T+\ldots$$

where g(i)T is the weighting sequence of the filter. Using eqn. 2.3.1, we obtain

$$Y(Z) = \sum_{n=0}^{\infty} [g(0)Tx(n)T+g(1)Tx(n-1)T +g(2)Tx(n-2)T+ \ldots]\, Z^{-n}$$

therefore

$$Y(Z)=g(0)TX(Z)+g(1)TZ^{-1}X(Z)+g(2)TZ^{-2}X(Z)+\ldots$$
$$= [g(0)T+g(1)TZ^{-1}+g(2)TZ^{-2}+ \ldots]\, X(Z)$$

therefore

$$Y(Z) = G(Z)\, X(Z)$$

$$G(Z) = Y(Z)/X(Z)$$

The ratio Y(Z)/X(Z) (equal to G(Z)), is commonly referred to as the pulse transfer function of the digital filter.

5. Summation :

Suppose that

$$g(n)T = \sum_{i=0}^{n} x(i)T \qquad \text{for} \quad n = 0, 1, 2, \ldots$$

and using eqn. 2.3.1 we obtain

$$X(Z) = \sum_{n=0}^{\infty} [g(n)T-g(n-1)]T\, Z^{-n}$$
$$= G(Z)-Z^{-1}G(Z)$$

$$= G(Z)\ [(Z-1)/Z]$$

therefore

$$G(Z) = [\ Z/(Z-1)\ ]\ X(Z)$$

for further detailes of the Z-transform refer to references (4,6).

2.5 Relation Between the Z-Domain and S-Domain

In order to gain some insight into the relation between S-plane and Z-plane poles and zeros, it is important to investigate what happens to the complex variable Z when S has certain typical values. The process by which a point in one plane is transfered to the other plane is called mapping, and that mapping process is governed by the law :

$$S = \sigma + j\omega \quad \text{and} \quad Z = e^{ST} = e^{(\sigma+j\omega)T}$$

$$= e^{\sigma T}.e^{j\omega T}$$

where T is the period of the sampling process.

It is obvious that for any constant value of σ, Z is a function of ω and is repetitive in form with a period equals $2\pi/T$ radians/sec. The previous value of Z is represented by a vector of length $e^{\sigma T}$ which makes an angle ωT radians with the positive real axis. As shown in fig. 2.5 any point S_x in the S-plane corresponds to just one point Z_x in the Z-plane. Now referring to table 2.1, it is seen that the imaginary axis in the S-plane transforms (maps) to the circumference of the unit circle in the Z-plane.

From fig. 2.5, it can be seen that if :

(1) $\sigma < 0 \longrightarrow |Z| < 1$, the left-hand half of the S-plane transforms into the inside the unit circle in the Z-plane.

(2) $\sigma = 0 \longrightarrow |Z| = 1$, every point on the imaginary axis ($j\omega$) in the

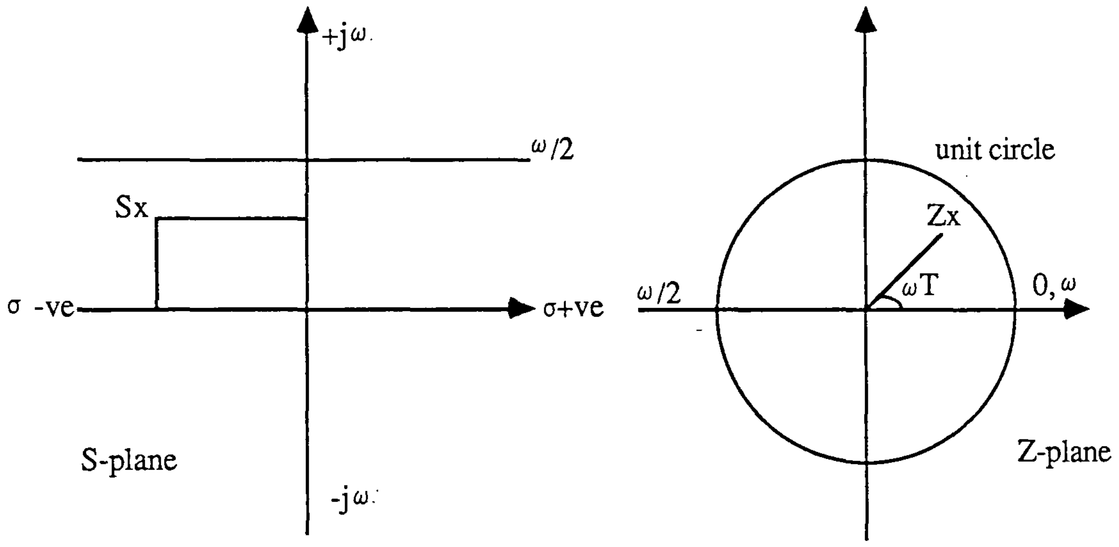


Fig. 2.5 S-plane to Z-plane transformation

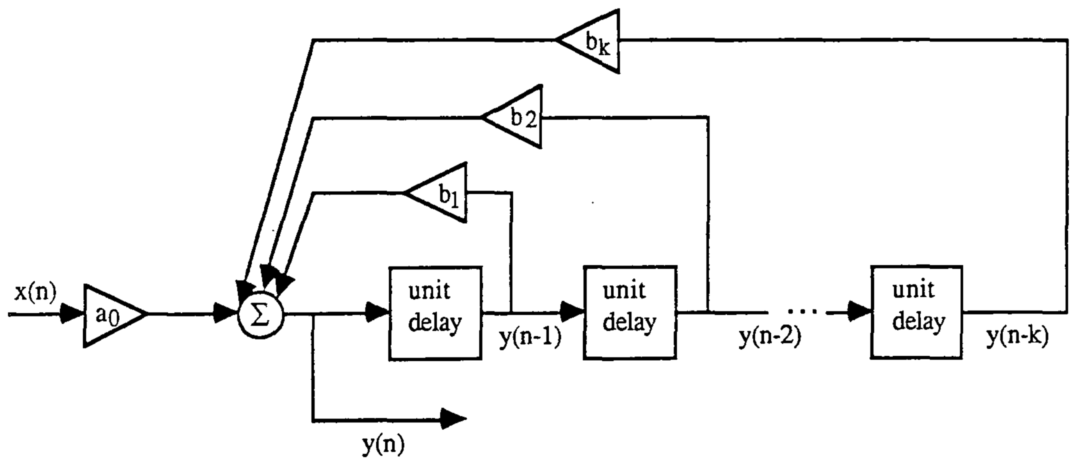


Fig. 2.6 Block diagram representation for the state variable description

$\sigma=0, \quad \omega_s = 2\pi/T$	
$j\omega$	$Z=1 \angle \omega T$
0	$1 \angle 0^\circ$
$\omega_s/8$	$1 \angle 45^\circ$
$\omega_s/4$	$1 \angle 90^\circ$
$3\omega_s/8$	$1 \angle 135^\circ$
$\omega_s/2$	$1 \angle 180^\circ$
$5\omega_s/8$	$1 \angle 225^\circ$
$3\omega_s/4$	$1 \angle 270^\circ$
$7\omega_s/8$	$1 \angle 315^\circ$
ω_s	$1 \angle 360^\circ$

Table 2.1

S-plane is mapped into a point on the unit circle in the Z-plane.

(3) $\sigma > 0 \longrightarrow |Z| > 1$, the right-hand half of the S-plane can be mapped onto the outside of the unit circle in the Z-plane (4,44).

2.6 The Inverse Z-Transform

The inversion of the Z-transform is conducted to determine the time-domain description of the digita filter from the corresponding frequency-domain description (4).

The inverse Z-transform relation is expressed as

$$x(n)T = (1/2\pi j) \oint_c X(Z)Z^{n-1} \, dZ \qquad 2.6.1$$

where c is the counterclockwise contour that encircles the origin. For rational Z-transform contours, integrals of the form of eqn. 2.6.1 are often conveniently evaluated using the residue theorem, i.e.

$$x(n)T = \sum \text{residue of } [X(Z)Z^{n-1}] \text{at the poles incide c} \qquad 2.6.2$$

where

$$\text{residue} = (1/(m-1)!) \lim_{Z \to x} \{(d^{m-1}/dZ^{m-1})[(Z-x)^m X(Z)Z^{n-1})]\} \qquad 2.6.3$$

for a pole of order m at z=x.

Another technique for recovering the sampled time function corresponding to a given Z-transform is simply to expand the Z-transform into a power series by ordinary long division.

A third method of determining the inverse Z-transform is to expand X(Z) into partial fractions, and then refer directly to a table of Z-transform pairs to obtain the corresponding Z-inverse, x(n)T, of each partial fraction (4,6).

2.7 The Difference Equation and the Digital Transfer Function

It has been established in section 2.3, that a digital filter may be represented by a linear difference equation, which is the counterpart of the differential equation in the linear continuous system. Furthermore, it has been established that the pulse transfer function of the filter is a ratio function of Z expressed as Y(Z)/X(Z), where Y(Z) and X(Z) are the Z-transforms of the set y(n)T and x(n)T respectively.

In general, the pulse transfer function of the digital filter G(Z) is given by

$$G(Z) = \frac{a_0+a_1Z^{-1}+a_2Z^{-2}+\ldots+a_NZ^{-N}}{1+b_1Z^{-1}+b_2Z^{-2}+\ldots+b_MZ^{-M}} = \frac{Y(Z)}{X(Z)} \qquad 2.7.1$$

$$G(Z) = \frac{\sum_{i=0}^{N} a_i Z^{-i}}{1+\sum_{i=1}^{M} b_i Z^{-i}} \qquad 2.7.2$$

where a_i and b_i represent the filter coefficients.

The pulse transfer function representation of eqn. 2.7.2 leads to a useful method for determining the filter's response to various input signals. This can be achieved by obtaining X(Z) and G(Z) and multiplying them together to give the Z-transform of the filter output response Y(Z). Finally the inverse Z-transform of Y(Z) is obtained to yield y(n)T. So it is obvious that the first stage in the design of a digital filter is to find the coefficients of the transfer function.

Now from eqn. 2.7.1, we obtain

$$X(Z)\left[a_0+a_1Z^{-1}+a_2Z^{-2}+\ldots+a_NZ^{-N}\right] =$$
$$Y(Z)\left[1+b_1Z^{-1}+b_2Z^{-2}+\ldots b_MZ^{-M}\right]$$

and since Z^{-k} represents a time delay of k sampling periods, then it follows that the input and output sampled-data signals are related by a linear difference equation which can be expressed as

$$\sum_{i=0}^{N} a_i x(n-i)T = \sum_{i=1}^{M} b_i y(n-i)T \qquad 2.7.3$$

since $b_0 = 1$, therefore

$$\begin{aligned} y(n)T &= a_0 x(n)T + a_1 x(n-1)T + \ldots + a_N x(n-N)T \\ &\quad -(b_1 Y(n-1)T + b_2 y(n-2)T + \ldots + b_M y(n-M)T) \qquad 2.7.4 \\ &= \sum_{i=0}^{N} a_i\, x(n-i)T - \sum_{i=1}^{M} b_i y(n-i)T \end{aligned}$$

The linear difference equation is the basic element in the concept of the digital filter, so the implementation of any digital filter should satisfy its difference equation. For further details refer to (4,42,45).

2.8 State Variable Analysis

2.8.1 Introduction

In the previous sections we investigated a way of describing the input-output relation of a system by a linear difference equation and transform domain description (Z-transform of the linear system in a discrete time system). This section discusses another method of describing the input-output relation of systems, namely the state space or state variable description. Generally, the state of a system at time t is that set of variables required at time t so that, given the inputs to the system for $\tau > t$, one can exactly specify the future behaviour of the system for $\tau > t$. So the state of a system may be represented by the values of a number of variables representing the state variables. A system might have an infinite or only a finite number of states and it can be applied to both continuous and discrete time systems. In this section we will discuss the state space description of the behaviour of discrete time system. Many textbooks and articles grant general insight to this powerful technique for system analysis and design (46,47,48,49,50,51).

2.8.2 State Space Description for Discrete Time System

Let the variables $q_1(n)$, $q_2(n)$, ... , $q_r(n)$ represent the state variables of a discrete system. At any instant n, the value of the output of the system, y(n), can be computed from the values of the state variables at that instant and the values of the inputs x(n), x(n+1), .. . For the discrete time system, the normal form of the difference equation may be expressed in matrix form as:

$$q(n+1) = \underline{A}\ q(n) + \underline{B}\ \underline{x}(n) \qquad 2.8.1$$

and the corresponding state output equation is

$$\underline{y}(n) = \underline{C}\ q(n) + \underline{D}\ \underline{x}(n) \qquad 2.8.2$$

where q(n) is the state vector of the system of n state variables

$\underline{x}(n)$ is the input vector.

$\underline{y}(n)$ is the output vector.

$\underline{A}$ is the matrix of coefficients of the state variables.

$\underline{B}$ is the input signals coefficients matrix.

$\underline{C}$ is the matrix of the output signals coefficients.

$\underline{D}$ is the matrix of the input signals terms contained in the output equation.

Eqns. 2.8.1 and 2.8.2 are referred to as the state equations, and a description of a system behaviour by these equations is called the state space description.

Consider, for example, a discrete time system represented by the difference equation

$$y(n) + b_1y(n-1) + \dots + b_ky(n-k) = a_0x(n) \qquad 2.8.3$$

This equation could be represented by a block diagram as illustrated in fig. 2.6. Because eqn. 2.8.3 is an nth order difference eqution, therefore, n state variables should be specified to describe the system (47). Defining the state variables as the outputs of delay elements, we obtain

$$\begin{aligned} q_1(n) &= y(n-k) \\ q_2(n) &= y(n-k+1) \\ &\vdots \\ q_k(n) &= y(n-1) \end{aligned} \qquad 2.8.4$$

referring to fig. 2.6, each state variable at time (n+1) could be written in terms of those at time n as

$$\begin{aligned} q_1(n+1) &= q_2(n) \\ q_2(n+1) &= q_3(n) \\ &\vdots \\ q_{k-1}(n+1) &= q_k(n) \\ q_k(n+1) = y(n) &= a_0x(n) - b_kq_1(n) - \dots - b_1q_k(n) \end{aligned} \qquad 2.8.5$$

To obtain the state variable representation in matrix form. eqn. 2.8.5 is rewritten as

$$\begin{bmatrix} q_1(n+1) \\ q_2(n+1) \\ \vdots \\ \\ q_k(n+1) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & & 0 \\ 0 & 0 & 1 & & 0 \\ & & \cdots & & \\ 0 & 0 & 0 & & 1 \\ -b_k & -b_{k-1} & -b_{k-2} & \cdots & -b_1 \end{bmatrix} \begin{bmatrix} q_1(n) \\ q_2(n) \\ \vdots \\ \\ q_k(n) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ a_0 \end{bmatrix} x(n)$$

and

$$y(n) = \begin{bmatrix} -b_k & -b_{k-1} & \cdots & -b_1 \end{bmatrix} \begin{bmatrix} q_1(n) \\ q_2(n) \\ \vdots \\ qk(n) \end{bmatrix} + \begin{bmatrix} a_0 \end{bmatrix} x(n)$$

The state equation can be solved at any time n by solving for the succeeding values of the state variables in terms of the preceeding values and the input sequence. Starting with the state at time zero, the q(n) can be computed in a step-by-step manner as follows (48):

$$\begin{aligned} q(1) &= \underline{A}q(0) + \underline{B}x(0) \\ q(2) &= \underline{A}q(1) + \underline{B}x(1) \\ &= \underline{A}\left[\underline{A}q(0) + \underline{B}x(0)\right] + \underline{B}x(1) \\ &= \underline{A}^2q(0) + \underline{ABx}(0) + \underline{B}x(1) \\ q(3) &= \underline{A}q(2) + \underline{B}x(2) \\ &= \underline{A}^3q(0) + \underline{A}^2\underline{B}x(0) + \underline{AB}x(1) + \underline{B}x(2) \end{aligned}$$

by continuing this procedure we get a general expression as follows:

$$\underline{q}(n) = \underline{A}^n q(0) + \sum_{m=0}^{n-1} \underline{A}^{n-1-m}\, \underline{Bx}(m)$$

It is obvious from the previous equation that the behaviour of such a state system depends on successive powers of the matrix $\underline{A}$. This equation is referred to as a discrete-transition equation of the system. The system output in that case can be written as

$$\underline{y}(n) = \underline{CA}^n q(0) + \underline{C} \sum_{m=0}^{n-1} \underline{A}^{n-1-m} \underline{Bx}(m) + \underline{Dx}(n)$$

The matrix $\underline{A}^n$ is referred to as the discrete state-transition matrix.

2.9 Frequency Response of Digital Filters

The frequency response can be determined geometrically from values of the transfer function of the filters. However, for discrete-time systems, the frequency response is usually determined by evaluating the transfer function G(z) around the unit circle in the Z-plane, where for continuous-time systems, the frequency response is obtained by evaluating the transfer function along the imaginary axis $j\omega$ (4).

To determine the amplitude and phase characteristics, the poles and zeros of the digital transfer function are plotted in the Z-plane, as shown in fig. 2.7. The amplitude response $|G(e^{j\omega T})|$ may be expressed as

$$|G(e^{j\omega T})| = \frac{\prod_{i=1}^{r} \text{vector magnitudes from the } i^{th} \text{ zero to the point on the } \omega\text{-axis}}{\prod_{k=1}^{m} \text{vector magnitudes from the } k^{th} \text{ pole to the point on the } \omega\text{-axis}}$$

and the phase response $\angle G(e^{j\omega T})$ may be obtained as

$$\angle G(e^{j\omega T}) = \sum_{i=1}^{r} \text{angles from the } i^{th} \text{ zero to the point on the } \omega \text{ axis} - \sum_{k=1}^{m} \text{angles from the } k^{th} \text{ pole to the point on the } \omega \text{ axis}$$

For example, let v_2 , v_3 be the zero vectors and

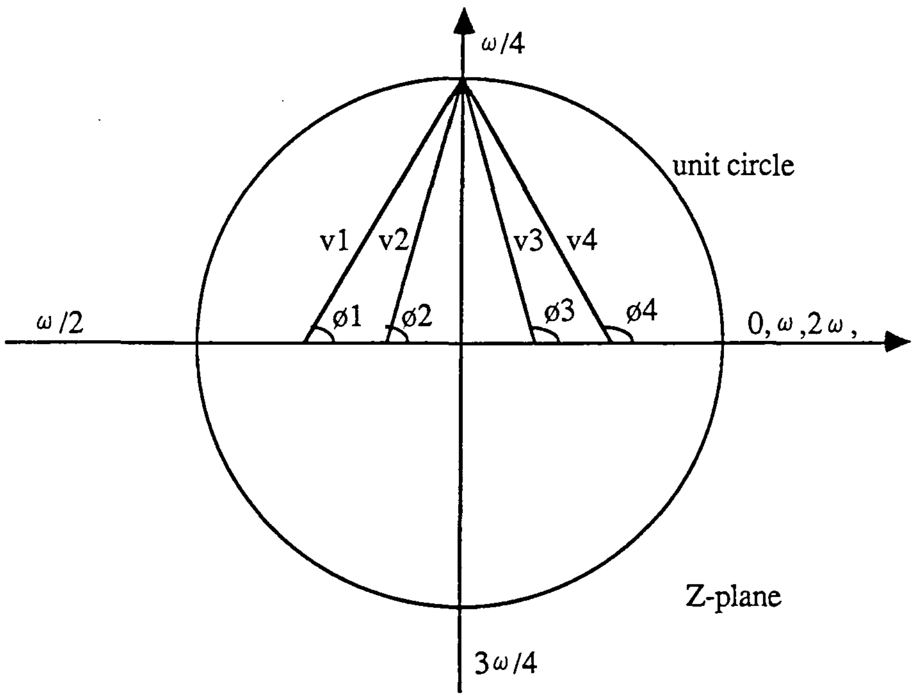


Fig. 2.7 Z-Plane pole-zero configuration

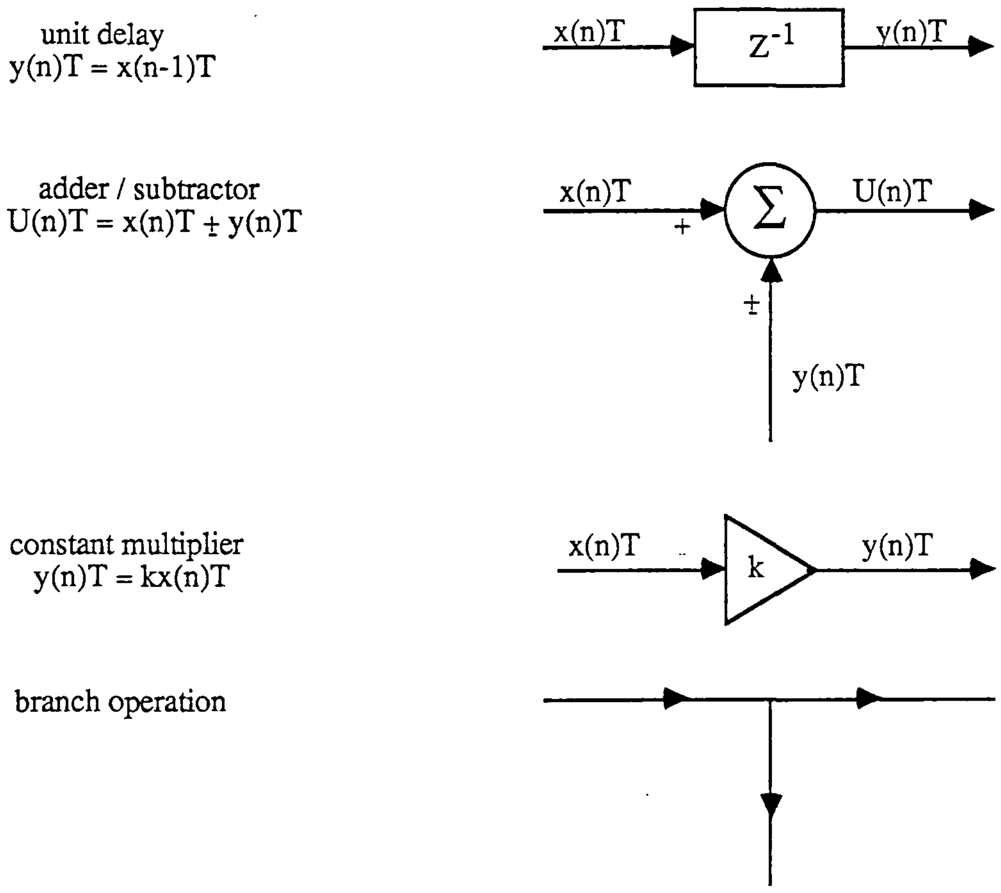


Fig. 2.8 The recommended terminology used in digital filtering

v_1 , v_4 the pole vectors

therefore

$$|G(e^{j\omega T})| = \frac{|v_2| \cdot |v_3|}{|v_1| \cdot |v_4|}$$

let θ_2 , θ_3 be the zero angles and

θ_1 , θ_4 be the pole angles

therefor

$$\angle G(e^{j\omega T}) = (\theta_2 + \theta_3) - (\theta_1 + \theta_4)$$

The above relationships for $|G(e^{j\omega T})|$ and $\angle G(e^{j\omega T})$ are point by point relationships only, in other words, vectors must be drawn on the Z-plane from the zeros and poles to every point on the axis as shown in fig. 2.7, for which the amplitude and phase response is required.

The frequency response can also be determined by substituting $e^{j\omega T}$ for Z in G(Z), and computing directly $|G(e^{j\omega T})|$ and $\angle G(e^{j\omega T})$ (4).

2.10 Digital Filter Design Techniques

Digital filters can be classified into two categories; finite impulse response (FIR) or non-recursive digital filters and infinite impulse response (IIR) or recursive digital filters. The term non-recursive means that the output of the filter is computed using the present and previous inputs only, i.e. $b_i=0$. On the other hand, the term recursive means that the output of the digital filter is computed using the present input and the previous inputs and outputs, i.e. $b_i \neq 0$ (4,45).

It has been mentioned before that the main problem in digital filter design is the determination of the coefficients to obtain its difference equation. These coefficients can be determined by suitable approximation or truncation of the impulse response functions in either the time or frequency domain.

There are at least two different frequency domain approaches in designing the IIR filters (4,52):

(1) the direct approach which is concerned with the Z-plane representation of the digital filter, where the coefficients can be determined using some computational algorithm directly from the filter specifications. This direct approach may be used in the designing of frequency sampling filters and those based on squared magnitude functions.

(2) the indirect approach in which the coefficients of the digital filter are determined by transfering the frequency response of the analogue filter G(S) via an appropriate S-plane mapping to a corresponding digital filter transfer function G(Z). The mapping process can be one of the following processes:

(a) Z-transform (impulse-invariant design method).

(b) Bilinear Z-transform.

(c) Matched Z-transform.

FIR filters (52) have linear phase characteristics and may be designed by a number of methods, of which the following are considered:

(1) frequency sampling, where the coefficients may be determined using the discrete Fourier transform (DFT) which is equivalent to a least-square approximation.

(2) the window methods (weighting functions), used for

functions containing discontinuities. Much work has been devoted to developing suitable window methods. There are many useful window functions which may used in the design of non-recursive digital filters (52) such as:

(a) Hamming window.

(b) Blackman window.

(c) Hanning window, for detail refer to (4,7).

(3) Optimal design methods, these are the most accurate (and complex) ways of designing FIR filters.

Forsythe (53) has developed a new method for computing the coefficients of a digital filter, in which the poles of the transfer function G(S) in the S-plane are mapped directly into the corresponding Z-plane, but the positions of the Z-plane zeros are derived by a more complex process using a Taylor series expansion. This method has the capability of approaching the filter response's theoritical limit unlike any other.

2.11 Realization of Digital Filters

2.11.1 The Realization of Recursive Digital Filters

The terminology shown in fig. 2.8 is suggested by Rabiner (54) for digital filtering. Referring to the linear difference equation (eqn.2.7.4) given in section (2.7). It is obvious that the digital filters are composed of circuits which perform three fundamental operations of storage, multiplication and addition (5). There are many network realizations (45) of IIR filters. One consideration that should be taken into account in the choice between these different realizations is the number of operations to be performed and their precision. That is, networks with the fewest constant multipliers and the fewest delay branches are often the most desirable. The recursive difference equation may be realized in the following forms:

2.11.1.1 Direct Form I

Since the difference equation (eqn. 2.7.4) can be written directly by inspection of the transfer function (in the form of eqn.2.7.1), the network corresponding to eqn. 2.7.4 is called the direct form of the system by (eqn. 2.7.1).

The realization structure shown in fig. 2.9 represents a direct form implementation of eqn. 2.7.4, and it is seen that a k^{th} order filter requires 2k delay operations. The polynomial coefficients are the multiplier values in the feed-forward and feed-back paths (4,6,45).

2.11.1.2 The Direct Form II or Canonic Form

In the canonic form, the feed-forward and feed-back paths share the same delays. In that case the filter is realized more concisely saving a number of delays so k delay operations are required as shown in fig. 2.10 (44,45).

As an example Hwang (55) has presented a new method of digital network realization. A total of 14 regular canonic forms have been obtained.

Hence the total filtering operation is often subdivided into many different processes which are combined to give the required overall transfer function. This subdivision may be achieved in two basic forms (45).

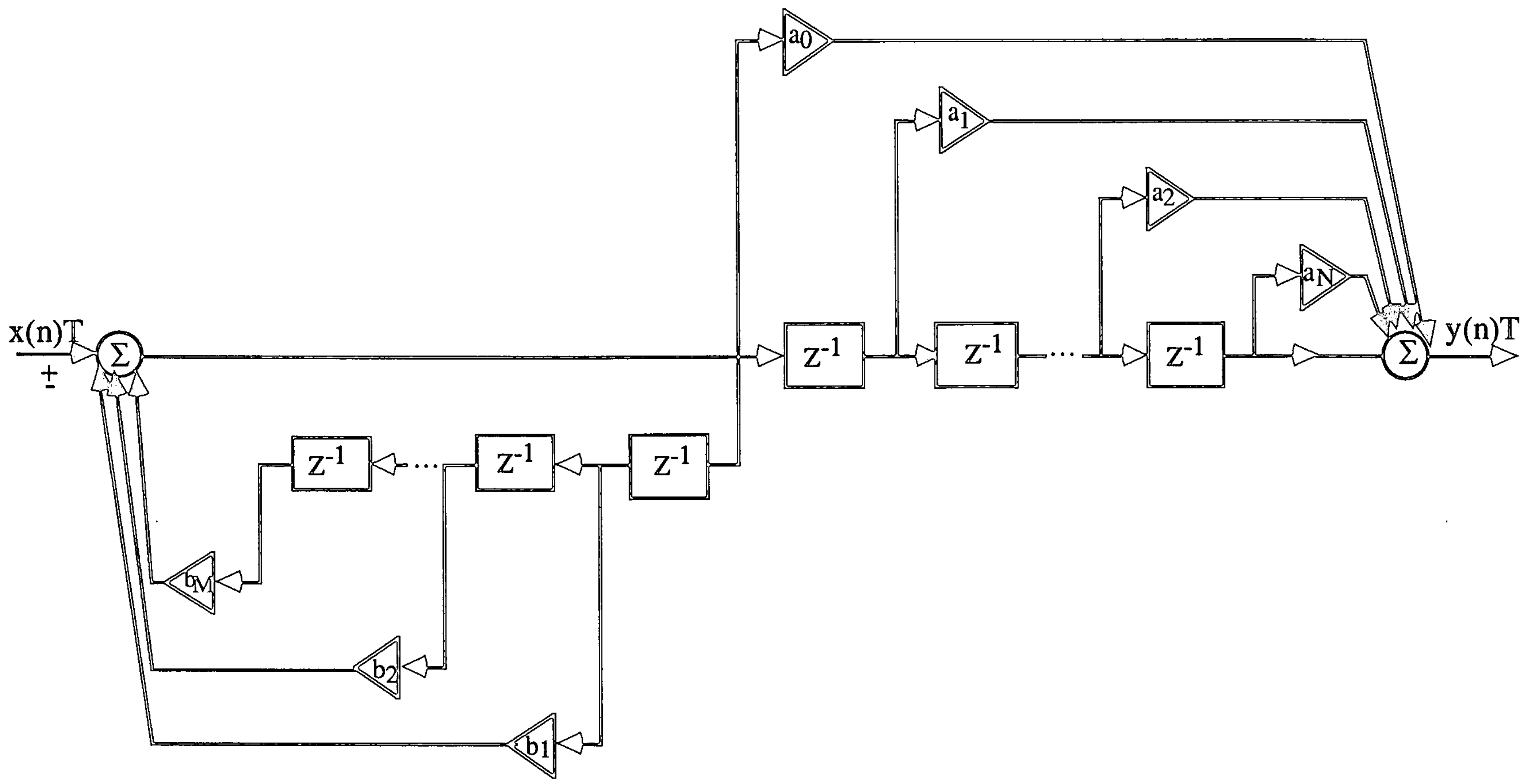


Fig. 2.9 Block diagram representation of the direct form I

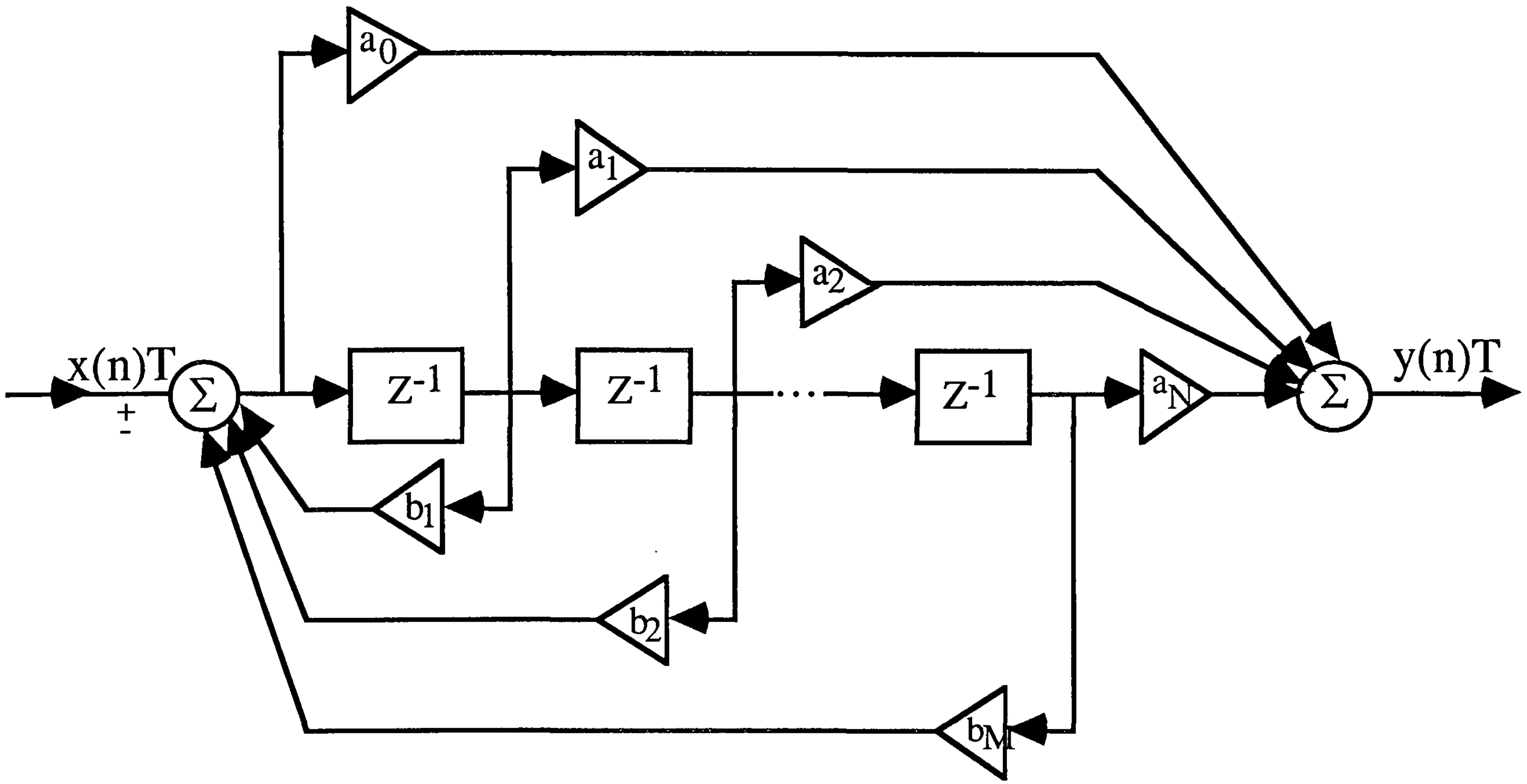


Fig. 2.10 Block diagram representation of direct form II for an nth order filter

2.11.2.3 Parallel Form

To realize a digital filter in parallel form the pulse transfer function G(Z) is expressed as a sum using a partial fraction expansion.

$$G(Z) = C + \sum_{i=1}^{k} G_i(Z)$$

where each $G_i(Z)$ is a first or second order transfer function. The filter can then be realized via a parallel connection of lower order filters in either of the direct forms as depicted in fig.2.11 (8,55).

2.11.2.4 Cascade Form

To realize a digital filter in a cascade form, the pulse transfer function G(Z) is factorised to the form

$$G(Z) = C \prod_{i=1}^{k} G_i(Z)$$

In this form the output y(n)T is the product of the outputs of several subfilters. Similarly it can be realized via the cascaded connection of lower order filters in either of the direct forms as depicted in fig. 2.12 (8,55).

2.11.2 Realization of Non-Recursive Digital Filters

For a non-recursive digital filter, the constant coefficients $b_i=0$, and the transfer function reduces to the polynomial Z^{-1}. Therefore, the current output only depends on the present and previous input samples, and the filter can be realized as in fig. 2.13 (8).

2.12 Frequency transformation for digital filters

Frequency transformation is a method of implementing a desired

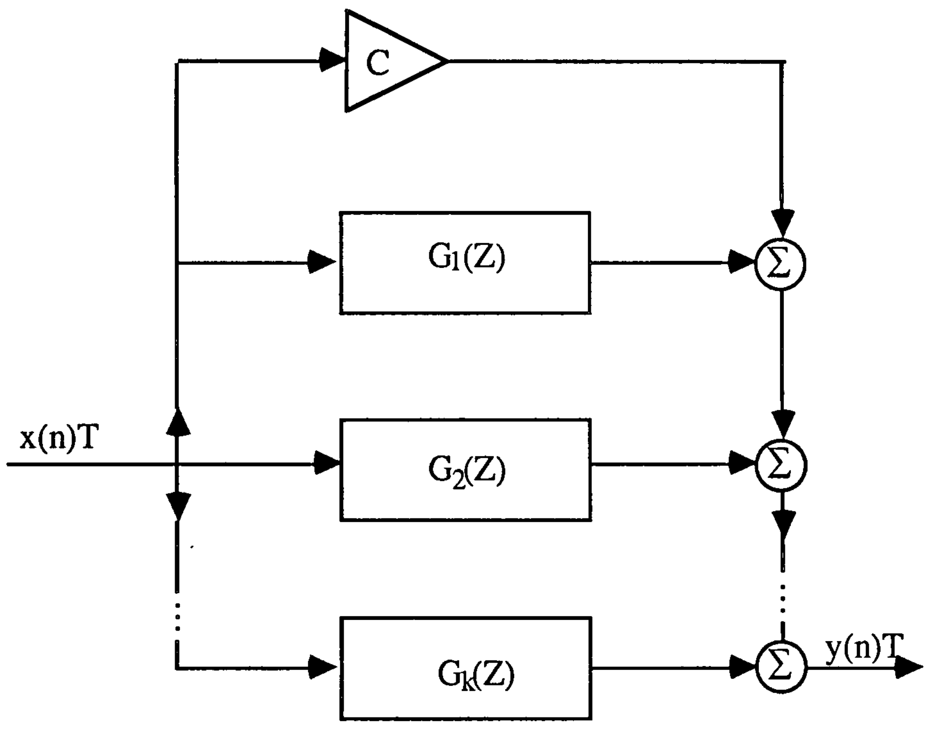


Fig. 2.11 Block diagram representation of the parallel form

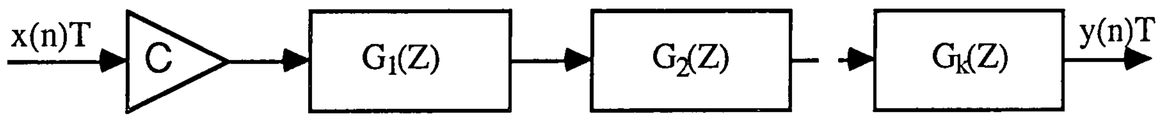


Fig. 2.12 Block diagram representation of the cascade form

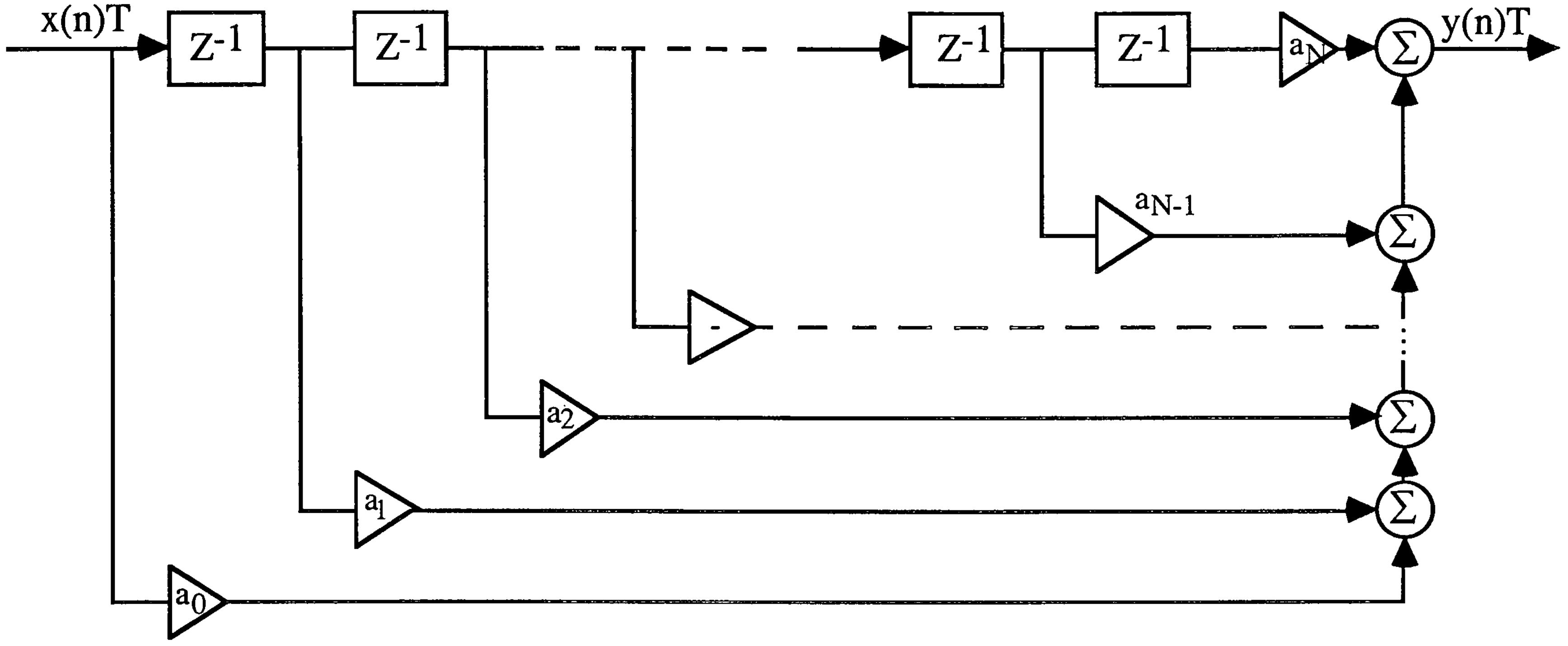


Fig. 2.13 Block diagram representation of non-recursive digital filter

filter from a given normalized filter. A normalized low pass digital filter is one in which the cutoff frequency is a quarter of the sampling frequency. The transformation may be accomplished by changing the transfer function of a given filter to the desired filter. These transformations generally preserve the normalized filter magnitude response (attenuation); other characteristics of it are often retained (8).

Constantinides (56,57) has developed a theory of transforming a given low-pass pulse transfer function into (a) high pass, (b) band-pass and band-stop on the Z-plane without reference to the frequency transformations for an analogue filter.

Table 2.2 below lists the frequency transformation from a normalized low-pass filter to other types of filters at any cutoff frequency :

Filter	Substitute for Z^{-1}	Cutoff frequency	Center frequency
highpass	$-Z^{-1}$	$\omega_{cl}=\omega_s-\omega_{ch}$	$\omega_s/2$
bandpass	$\frac{-Z^{-1}(Z^{-1}-\alpha)}{1-Z^{-1}}$	$\omega_{cl}=\omega_s-\omega_{ch}$	ω_0
bandstop	$\frac{Z^{-1}(Z^{-1}-\alpha)}{1-Z^{-1}}$	$\omega_{cl}=s-(\omega_2-\omega_1)$	ω_0

Table 2.2

where $\alpha = \cos(2\ \omega_0/\omega_s)$, and

ω_{cl} is the cutoff frequency of the normalized low-pass filter.

ω_{ch} is the cutoff frequency of high-pass filter.

ω_0 is the center frequency.

ω_1 and ω_2 are the lower and upper cutoff frequencies respectively.

2.13 Quantization Effects

When a digital filter is implemented on a general or special purpose computer, errors due to the limit of the word length available, which represents the number in the digital computer, become critical and may lead to a filter that does not satisfy its original specifications (58). Here the quantization has been defined as the approximation or the replacement of the signal value by the nearest of a set of quantization levels differing by steps of the size $Q = 1/2^{w-1}$, where w is the word length. The effect of that quantization is to superimpose an error signal e(t), called the quantization noise, on the original signal. So the input to the digital filter is considered to be the sum of two signals, namely, a noiseless input x(n)T and a noise input e(n)T. That is, the quantized input signal is expressed as:

$$x(n)T\big|_q = x(n)T + e(n)T$$

and the amplitude of the error signal is extended for -Q/2 to Q/2 (4,58).

The quantization errors normally take the form :

(I) quantization errors due to round-off and truncation in arithmetic operations.

(II) quantization errors due to the inaccuracy of the input signal since it is represented by a set of discrete values.

(III) quantization errors due to representing the coefficeints by a finite number of bits.

(IV) limit cycle oscillations; and

(V) overflow oscillations.

2.13.1 Round-Off Noise

To implement a digital filter, the signal must be multiplied by a constant coefficients. The result of this multiplication must, in general, be approximated to a number of bits that can be stored properly. This approximation (round-off) can be rounding or truncation (59). The effect of truncation or rounding depends on whether fixed-point or floating-point arithmetic operations are used and how negative numbers are represented. If the quantization step is assumed to be constant for all signal amplitudes, in the other words if noise samples are assumed to be uniformly distributed, then the variance of the input noise is simply $Q^2/12$, $Q^2/3$ for rounding and truncation respectively (60).

Mitra et al. (60) have developed a simple method of calculating a steady state value for the output noise variance of a digital filter as a result of the input quantization noise.

Liu (61) has analysed the round-off noise for each form of digital filter realization showing that the accuracy of a digital filter depends on two important factors; the form of realization and the type of arithmetic used. For fixed point arithmetic, the mean square value of the output noise variance is expressed as

$$(1/2\pi j)\oint \Phi_{ee}(Z)\ dZ/Z$$

where $\Phi_{ee}(Z)$ is the power spectral density of the round-off noise for each form of realization.

According to the studies in (61) for a fixed-point word length, realizing the high-order filter with either parallel or cascade form is considerably more accurate than the direct realization of the same filter. Thus first and second order filters should be used as basic building blocks for higher order filters.

It has been shown that the accuracy of a digital filter design depends on the implementation method, but to achieve still better accuracy, some other advantages need to be sacrified. Barnes has (62) shown that the direct form structures are computationally efficient, but they have high round-off noise gain for narrow-band filters, high coefficient quantization sensitivity, and show some overflow noise. On the other hand optimal state space structures have low round-off gain noise, possess near-minimal coefficient sensitivity, and are free from the overflow or limit cycles independent of the arithmetic convergence. However, the last structure requires three more multipliers per second order section than the direct form. For further details about these structures refer to (62,63).

2.13.2 Input Quantization Noise

In the implementation of a practical digital filter system when a signal is converted from a continuous to a discrete form by an ADC, which normally produces a fixed-point binary number representation of the input samples (4), the digital output has a finite word length which implies a difference between the actual value and the fixed-point representation. This is commonly referred to as white noise (44).

The mean square value used in assessing this noise in the ADC is $\sigma^2 = Q^2/12$ and the mean square value of the error at the output due to input quantization is expressed as

$$= (Q^2/12)(1/2\pi j) \oint G(Z)\, G^*(1/Z)\ dZ$$

2.13.3 Coefficients Rounding Effect

Since the coefficients of the difference equation represent the digital filter, inaccuracy of the coefficients can cause degeneration in the frequency response of the filter. Specifically, changes in the coefficients values give rise to migration of the poles and zeros of the filter, and if any of the poles happen to migrate outside the unit circle in the Z-plane, then the filter, which with accurate coefficients would have been stable, becomes unstable (64).

Knowled and Olcayto (65) have shown that the quantization of a digital filter's coefficients can be represented by a (stray) transfer function in parallel with a corresponding ideal filter. They have also given a measure of the degeneration in filter performance due to coefficient errors by the following statistical mean-squre convergence criterion

$$\sigma_w^2 = (T/2\pi)\int_0^{2\pi/T} \left| G^*(j\omega)-G_\infty^*(j\omega) \right|^2\ d\omega$$

where $G^*(j\omega)$ and $G_\infty^*(j\omega)$ represent the frequency response of the actual and ideal filter, and T is the sampling period.

It is contended that loss of stability in a realization will occur only after the deviation between the actual and ideal frequency response has become intolerable.

To select the minimum word-length, the expression

$$3\sqrt{\sigma_w^2} < \left|\text{Acceptable Gain Fluctuation}\right|$$

must be satisfied, provided that the output noise due to data quantization and multiplicative round off errors is also acceptable with this word length.

Liu (61) has indicated that the effect of coefficients inaccuracy is more pronounced for a high order filter when it is realized in the direct form than when it is realized in the parallel or cascade form.

2.13.4 Limit Cycle Oscillations

Limit cycles are defined to be the autonomous dc or periodic behaviour of a digital filter under zero input conditions, so that with zero input, the output of a recursive digital filter may be non-zero due to the arithmetic rounding or truncation (66).

Munson et al. (67) have used an algorithm to find maximum amplitude limit cycles for many different filters using sign-magnitude and two's complement rounding and truncation with either one or two quantizers.

2.13.5 Overflow Oscillations

The fact that overflow oscillations can exist in digital filters is due to the nonlinearity associated with overflow which occurs due to the fixed point arithmetic addition.

The necessary and sufficient condition for the absence of overflow oscillation "nonlinearity" for digital filter with two's complement arithmetic is that:

$$|\text{The Total Input to the Adder}| \leq 1 \quad (68).$$

2.14 Conclusion

This chapter has presented some methods of describing the input-output relation of the system such as linear difference equation, z-transform and the state space description. The basic concepts of sampled-data signals has also been discussed. We have seen in this chapter that the standard Z-transform is fundamental to a basic understanding of digital filter concepts and its convolution-summation property provides the relationship between the filter's input and output signals. Methods of realizing FIR and IIR digital filters have also been investigated. In fact, IIR filters are generally more economical in execution and computation time and storage requirements compared with FIR filters, but have less stability.

The practical implementation of digital filters is affected by quantization errors due to the finite word length of the input and coefficients. Errors in the coefficients will obviously cause the frequency response of the filter to depart to some extent from that desired, more serious difficulties arise with the IIR filters which may become unstable as a result of its Z-plane poles movement outside the unit circle- in the case of a small change in one or more of its recursive coefficients (or if its recursive multipliers are inaccuratly specified).

CHAPTER -3-

Adaptive Filtering

3.1 Introduction

The basic concepts involved in FIR adaptive filters have been known for many years. They are versatile signal processing elements which have found numerous applications in situations where only a limited knowledge about the signal and the interference is available.

Current real-time adaptive filters are based on Widrow's LMS algorithm, a practical solution to the Widrow-Hoff or (LMS) algorithm. The main advantage of this algorithm is its simple structure which makes it easy to be computed in real time (69).

Adaptive filters can be implemented in the time or frequency domain. Ferrara (70) has shown that the frequency domain implementation of adaptive filters requires less computational time than the time domain implementation. But this is only true for very high order filters, for example the ratio of the number of multipliers required for frequency domain implementation to the time domain implementation is 1.2 for a 32nd order, while it is 0.69 for a 64th order LMS adaptive filter. So for $n \geq 64$, where n is the order of the filter, there is some computational saving gained by implementing an LMS adaptive filter in the frequency domain.

Moschner (11,71) and Deivasigamani (72) have proposed the Clipped-LMS algorithm which operates on clipped input data and has convergence properties somewhat inferior to the conventional LMS algorithm and gives a small loss in the performance of the adaptive filter but is significantly simpler to implement and operates faster.

In this chapter, the basic concept of an adaptive noise canceller that selectively rejects an undesired signal from a composite of signal and noise, and the Wiener solution to the statistical noise cancelling problem are discussed. The LMS algorithm for updating the recursive and non-recursive coefficients will also be described. Quantization noise effects of the variables in the adaptive filter, which minimize the mean square error (MSE) of the filter response, due to the limited-word length and finite precision arithmetic of the digital processor and in particular the coefficient's error due to the value they take when finite precision arithmetic is used, will be investigated.

3.2 The Concept of the ANC

Characteristically an adaptive filter has three main components as shown in fig.3.1:

(a) The processor, a single input multiple output device, which provides memory for the system. The outputs of the processor are distinct linear functions of present as well as past values of the input.

(b) A set of adjustable weights which multiply the processor outputs. The sum of the weighted processor outputs is the output of the adaptive filter.

(c) Some means to compute new weight values according to the adaptive algorithm in use and means by which the weights can be updated (10).

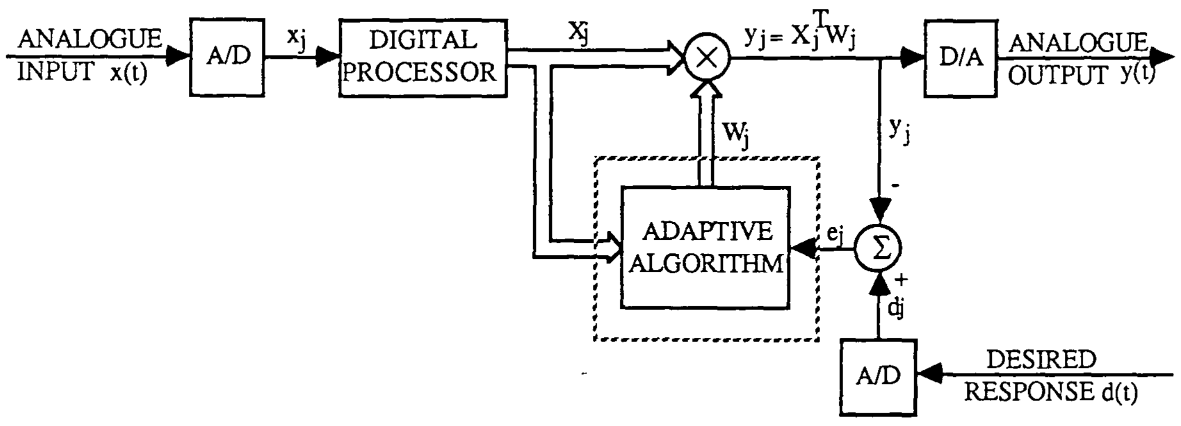


Fig. 3.1 Digital adaptive filter

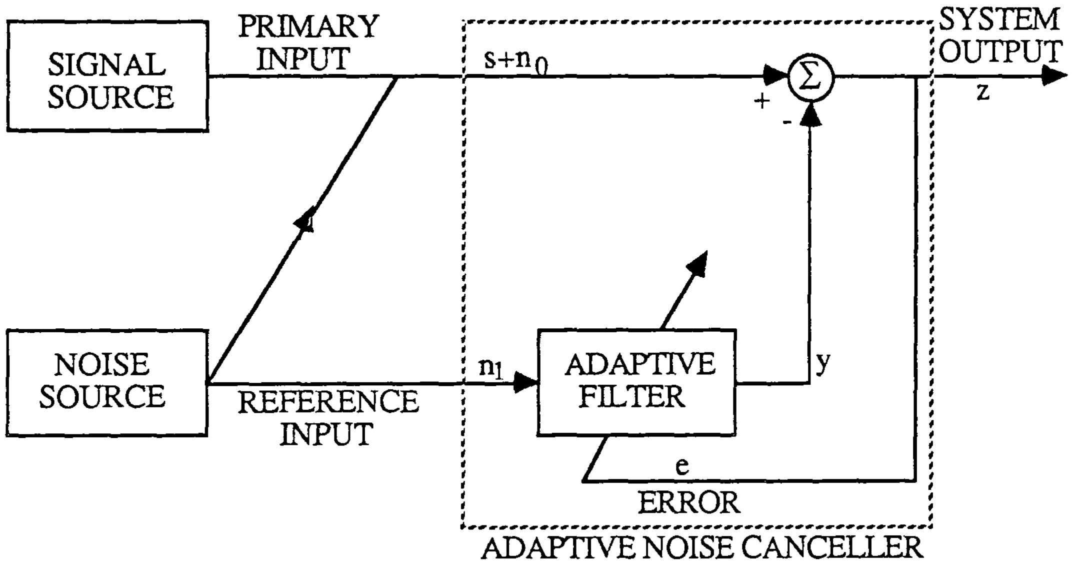


Fig. 3.2 The adaptive noise cancelling concept

Fig. 3.2 illustrates the basic priciples of adaptive noise cancelling. The input to the adaptive filter is a noise signal n_1 that is highly correlated in some unknown way with the additive noise (interference) n_0, but is uncorrelated with the clean signal s. The noise n_1 forms the reference input to the canceller. The combined signal and interference $s+n_0$ form the primary input to the canceller. The noise n_1 is filtered to produce an output y that is subtracted from the primary input to produce the system output z (11,14,73).

The adaptive filter has the ability to automatically adjust its own impulse response (weights) using an algorithm that responds to an error signal dependent on the filter's output. Thus using a proper adjustment algorithm, the filter can be operated in systems whose characteristics develop with time. In noise cancelling systems, the objective is to produce a system output z that is an estimate of the signal s. This objective is accomplished by feeding the system output back to the adaptive filter and adjusting the filter through an algorithm such as the (LMS) adaptive algorithm, (which will be described in detail in the next sections) to minimize the system output power, this will be explained later (11,12,74).

Assuming that the signal s is uncorrelated with both n_0 and n_1 and that s, n_0, n_1 and y are statistically stationary and have zero means, then the system output z is given as:

$$z = s+n_0-y \qquad 3.2.1$$

Squaring eqn. 3.2.1 we obtain

$$z^2 = s^2+(n_0-y)^2+2s(n_0-y) \qquad 3.2.2$$

Taking the expectation of both sides of eqn. 3.2.2, yields:

$$E\left[z^2\right] = E\left[s^2\right] + E\left[(n_0-y)^2\right] + 2E\left[s(n_0-y)\right] \qquad 3.2.3$$

since the signal s is assumed to be uncorrelated with n_0 and y. Since the signal power $E\left[s^2\right]$ is a fixed quantity, minimizing the output power yields:

$$\min E\left[z^2\right] = E\left[s^2\right] + \min E\left[(n_0-y)^2\right] \qquad 3.2.4$$

Thus, when the noise cancelling filter is adjusted so that $E\left[z^2\right]$ is minimized, $E\left[(n_0-y)^2\right]$ is also minimized. The filter output y is then a best square estimate of the primary noise n_0. Moreover, when $E\left[(n_0-y)^2\right]$ is minimized, $E\left[(z-s)^2\right]$ is also minimized, since from eqn. 3.2.1

$$(z-s) = (n_0-y) \qquad 3.2.5$$

Thus, z is a best least-square estimate of the signal s since minimizing the total output power causes the output z to be a best least-square estimate of the signal s. The output z will contain the signal plus noise. From eqn. 3.2.1, the output noise is given by (n_0-y). Minimizing the total output power, $E\left[z^2\right]$, allows one to minimize the output noise power, $E\left[(n_0-y)^2\right]$. Minimizing the total output power maximizes the output signal-to-noise (SNR) ratio as long as the output signal remains constant.

It is seen from eqn. 3.2.3 that the smallest possible output power is

$$E\left[z^2\right] = E\left[s^2\right]$$

When this is achieved,

$$E\left[(n_0-y)^2\right] = 0$$

Therefore,

$$y=n_0 \quad \text{and} \quad z=s$$

In this case, minimizing the output power causes the output signal to be perfectly noise free (11,15).

3.3 The LMS Adaptive Filter

The LMS adaptive filter is the basic element of the adaptive noise cancelling system.

3.3.1 Adaptive Linear Combiner

The adaptive linear combiner is the basic component, or the most significant portion, of most adaptive filtering and signal processing systems. A set of input signals are weighted and summed to form an output signal as shown in fig 3.3. The input signal vector $\underline{X}_j$ is defined as

$$\underline{X}_j \triangleq \begin{bmatrix} x_{0j} \\ x_{1j} \\ \cdot \\ \cdot \\ \cdot \\ x_{nj} \end{bmatrix} \qquad 3.3.1$$

The input signal components are assumed to occur simultaneously on all input lines discretely in time indexed by the subscript j. The weighting coefficients or multiplying factors are not fixed and are adjustable depending on the system. The weight vector $\underline{W}$ is defined by

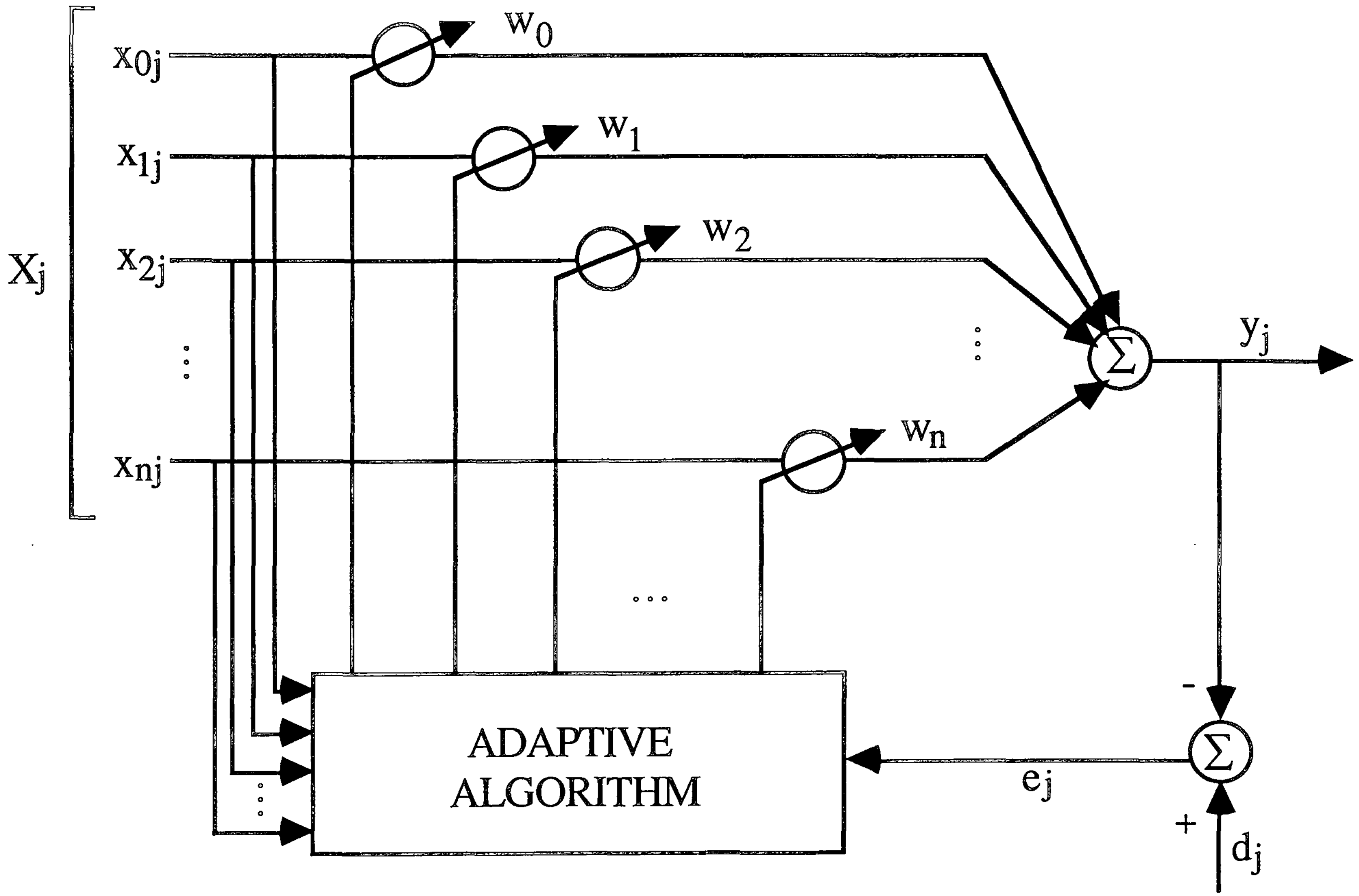


Fig. 3.3 The adaptive linear combiner

$$\underline{W} = \begin{bmatrix} w_0 \\ w_1 \\ \cdot \\ \cdot \\ \cdot \\ w_n \end{bmatrix} \qquad 3.3.2$$

Thus at the jth instant, the output y_j is equal to the inner product of $\underline{X}_j$ and $\underline{W}$

$$y_j = \underline{X}_j^T\underline{W} = \underline{W}^T\underline{X}_j \qquad 3.3.3$$

where the superscript T denotes matrix transposition.
The estimation error output at jth time interval e_j is

$$e_j = d_j-\underline{X}_j^T\underline{W} = d_j-\underline{W}^T\underline{X}_j \qquad 3.3.4$$

where d_j is the desired response input of the adaptive filter (11,12).

3.3.2 The LMS Adaptive Algorithm

The purpose of the adaptive algorithm designated in fig. 3.3 is to adjust the weights of the adaptive linear combiner to minimize the MSE. Squaring eqn. 3.3.4 one obtains

$$e_j^2 = d_j^2-2d_j\underline{X}_j^T\underline{W}+\underline{W}^T\underline{X}_j\ \underline{X}_j^T\underline{W} \qquad 3.3.5$$

Taking the expected value of both sides and assuming that x_j and d_j are zero-mean processes yields

$$E[e_j^2] = E[d_j^2]-2E[d_j\underline{X}_j^T]\underline{W}+\underline{W}^TE[\underline{X}_j\underline{X}_j^T]\underline{W} \qquad 3.3.6$$

Taking the vector $\underline{P}$ as the crosscorrelation function between d_j and the $\underline{X}_j$ vector then yields

$$\underline{P} = E\left[d_j\underline{X}_j\right] = E\begin{bmatrix} d_jx_{0j} \\ d_jx_{1j} \\ \vdots \\ d_jx_{nj} \end{bmatrix} \qquad 3.3.7$$

Defining $\underline{R}$ as the input autocorrelation function matrix of the input signal, we obtain:

$$\underline{R} = E\left[\underline{X}_j\underline{X}_j^T\right] = E\begin{bmatrix} x_{0j}x_{0j} & x_{0j}x_{1j} & x_{0j}x_{2j} \cdots & \\ x_{1j}x_{0j} & x_{1j}x_{1j} & x_{1j}x_{2j} \cdots & \\ \vdots & & & \\ \cdots & & \cdots & x_{nj}x_{nj} \end{bmatrix} \qquad 3.3.8$$

The MSE can then be expressed as

$$E\left[e_j^2\right] = E\left[d_j^2\right] - 2\underline{P}^T\underline{W} + \underline{W}^T\underline{R}\underline{W} \qquad 3.3.9$$

Note that the error is a quadratic function of the weights which can be represented as a concave hyperparaboloidal surface. The gradient methods widely used to minimize the error by optimally adjusting the weights by descending along this surface to seek its minimum (the bottom of the bowl) (11,69). The optimal weight vector $\underline{W}^*$, generally called the Wiener vector, which yields the minimum MSE (MMSE), is obtained by setting the gradient of the MSE function to zero.

The gradient of the MSE function is determined by differentiating eqn. 3.3.9 with respect to $\underline{W}^T$

$$\nabla[e_j^2] = \begin{bmatrix} \partial E[e_j^2]/\partial w_0 \\ \vdots \\ \partial E[e_j^2]/\partial w_n \end{bmatrix} = -2\underline{P}+2\underline{R}\,\underline{W} \qquad 3.3.10$$

thus :

$$\underline{W}^* = \underline{R}^{-1}\underline{P} \qquad 3.3.11$$

This equation is the Weiner-Hopf equation in the matrix form (11,75,76).

In practice, it is difficult to obtain $\underline{W}^*$ because we do not know the exact statistics of $\underline{R}$ and $\underline{P}$, but an estimate of $\underline{W}^*$ could be found, by estimating $\underline{R}$ and $\underline{P}$ for the given input and given desired response.

Many methods are used to adjust adaptive parameters, the most common method used is the stochastic gradient search technique by steepest descent which converges towards the optimum solution. In this method the weight vector is changed along the direction of the negative gradient.

Since the LMS algorithm is an implementation of the method of steepest descent, the next weight vector $\underline{W}_{j+1}$ is equal to the present weight vector $\underline{W}_j$ plus the negative gradient vector multiplied by a constant proportional to the negative gradient:

$$\underline{W}_{j+1} = \underline{W}_j - \mu \nabla_j \qquad 3.3.12$$

The parameter μ is the convergence factor which determines the rate of convergence, accuracy of the weight vector and stability.

The LMS algorithm estimates an instantaneous gradient by assumming that e_j^2 , representing a single squared error sample, is an estimate of the mean-square error and by differentiating e_j^2 with respect to $\underline{W}$. The relationship between the true and estimated gradients are expressed as follows (11,69):

$$\nabla = \begin{bmatrix} \partial E[e_j^2]/\partial w_0 \\ \cdot \\ \vdots \\ \partial E[e_j^2]/\partial w_n \end{bmatrix}_{\underline{W}=\underline{W}_j}, \hat{\nabla} = \begin{bmatrix} \partial e_j^2/\partial w_0 \\ \vdots \\ \cdot \\ \partial e_j^2/\partial w_n \end{bmatrix}_{\underline{W}=\underline{W}_j} = 2e_j \begin{bmatrix} \partial e_j/\partial w_0 \\ \cdot \\ \cdot \\ \partial e_j/\partial w_n \end{bmatrix}_{\underline{W}=\underline{W}_j} \qquad 3.3.13$$

The gradient estimate used by the LMS algorithm takes the gradient of the square of a single error sample thus:

$$\hat{\nabla}_j = -2e_j \underline{X}_j$$

Replacing the true gradient in eqn.3.3.12 by this estimate yields the so called Widrow-Hoff algorithm

$$\underline{W}_{j+1} = \underline{W}_j + 2\mu e_j \underline{X}_j \qquad 3.3.14$$

By using any arbitrary value for the weight vector as an initial value, the algorithm will converge in the mean and will remain stable as long as the parameter μ is greater than 0 and less than the reciprocal of the largest eigenvalue λ_{max} of the matrix $\underline{R}$ (11,75).

$$1/\lambda_{max} > \mu > 0 \ . \qquad 3.3.15$$

3.3.3 The Convergence Factor "μ"

It has been mentioned in the previous section that μ controls the rate of convergence, stability and the accuracy of the adaptive filter.

Stability, and the relation between the speed of adaptation and performance of the adaptive system have been extensively studied by a number of authors, (23) and there is no general agreement on conditions for the filter stability. However, in general, the large values lead to faster convergence or adaptation, but add significant noise to the weight values producing a more noisy adaptive process. It has been observed from many studies (11,23,77) that the filter stability or convergence is guaranteed within the range of values

$$1/\lambda_{max} > \mu > 0$$

Tanik et al. (78) have investigated the convergence behaviour of the LMS algorithm with regard to the assumption that the filter inputs x_j and d_j are Gaussian and independent over time. They have shown that the sufficient convergence condition for that case is:

$$0 < \mu < 1/3 \ (2/\Sigma\lambda_i)$$

Conventional adaptive filters use a fixed "μ" so it is the same for all parameters of the filter. Recently a few studies have been published about using a variable μ for the filter parameters (79). Mikhael et al. (35,80) have proposed using and adjusting individual convergence factors in real time for different filter parameters, so that their values are kept optimum for a new set of input variables. They have also shown from computer simulation results that the individual adaptation approach gives a much better performance than the conventional fixed group adaptation approach. The convergence factor values for the non-recursive and recursive coefficients are given as:

$$\mu_a(n) = 0.5/\sum_{i=0}^{N} x^2(n-i)$$

and

$$\mu_b(n) = 0.5/\sum_{i=1}^{M} y^2(n-i)$$

respectively.

3.3.4 The LMS Adaptive Filter

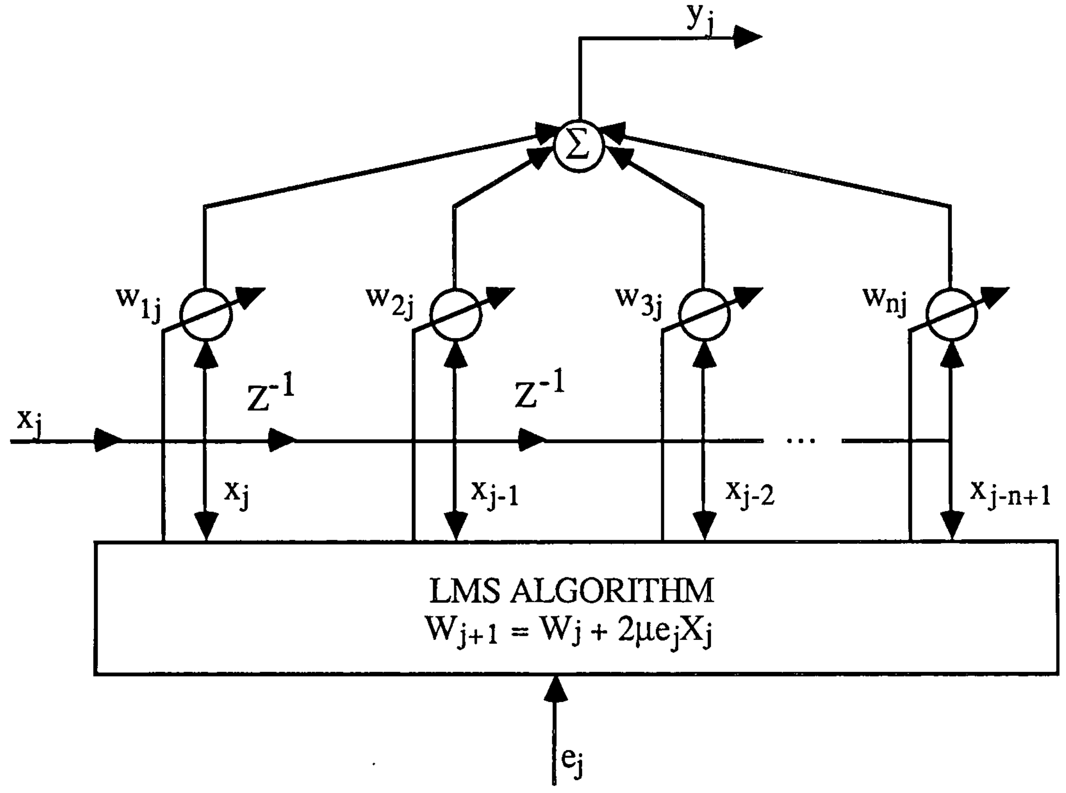
The adaptive filter may be formed by implementing the adaptive linear combiner in conjunction with a tapped delay line as shown in fig. 3.4.a. Because of the structure of the delay line, the input signal vector is

$$\underline{X}_j = \begin{bmatrix} x_j \\ x_{j+1} \\ \cdot \\ \cdot \\ \cdot \\ x_{j-1+n} \end{bmatrix}$$

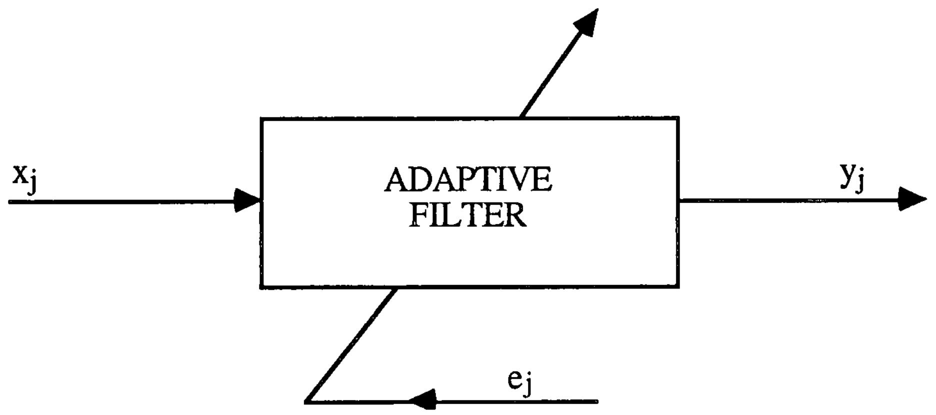
It is obvious that the components of this vector are delayed versions of the input signal x_j. Fig. 3.4.b represents a simplification of the adaptive tapped-delay line filter. This kind of filter permits the adjustment of gain and phase at many different frequencies simultaneously (11,76).

3.4 Wiener Solution to the Statistical Noise Cancelling Problem

This section presents the derivation of the optimal unconstrained Wiener solution to certain statistical noise cancelling problems. The purpose of this is to demonstrate analytically some advantages of the noise cancelling techniques such as the increase in signal-to-noise (SNR) ratio.



(a)



(b)

Fig. 3.4 The LMS adaptive filter
(a) Block diagram
(b) Symbolic representation

Fig 3.5 illustrates a classic single-input, single-output Wiener filter where x_j represents the input signal and y_j the output signal, which are assumed to be discrete in time and d_j is the desired response input. The input signal and the desired response input are assumed to be statistically stationary. The filter is linear, discrete and designed to be optimal in the minimum mean-square-error sense. The optimal impulse response $\underline{W}^*(k)$ of this filter can be obtained from the discrete Wiener-Hopf convolution summation equation:

$$\sum_{l=-\infty}^{\infty} \underline{W}^*(l)\Phi_{xx}(k-l) = \Phi_{xd}(k) \qquad 3.4.1$$

where

$$\Phi_{xx}(k) = E[x(j)x(j+k)] \quad \text{and}$$

$$\Phi_{xd}(k) = E[x(j)d(j+k)]$$

In this form the impulse response $\underline{W}^*(k)$ may be causal or noncausal and extendable finitely and infinitely to the left or right of the time origin, i.e. this is the unconstrained form.

The transfer function of the Wiener filter is

$$\mathcal{W}^*(Z) = \sum \underline{W}^*(k)\, Z^{-k} \qquad 3.4.2$$

Taking the Z-transform of egn. 3.4.1, then yields the optimal unconstrained Wiener transfer function:

$$\mathcal{W}^*(Z) = \frac{\delta_{xd}(Z)}{\delta_{xx}(Z)} \qquad 3.4.3$$

where $\delta_{xx}(Z)$ is the power-density spectrum of the input signal, which is the Z-transform of $\Phi_{xx}(k)$, and $\delta_{xd}(Z)$ is the cross power spectrum between the input signal and the desired response input,

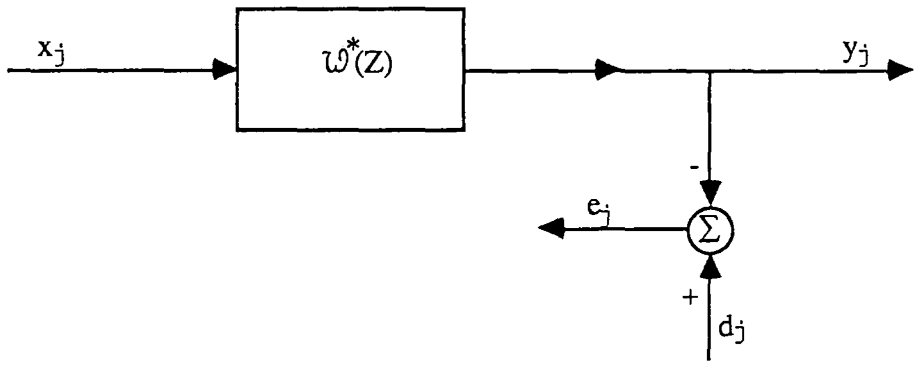


Fig. 3.5 Single channel Wiener filter

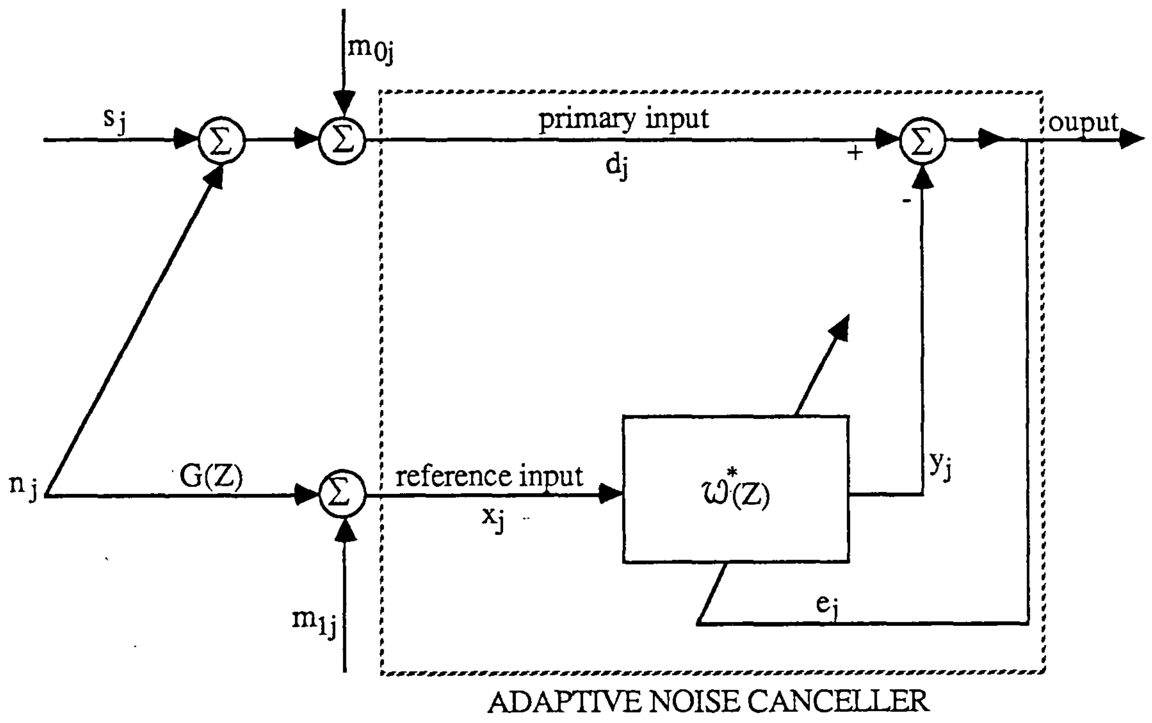


Fig. 3.6 Single channel adaptive noise caceller with correlated and uncorrelated noises in the primary and reference inputs

which is the Z-transform of $\Phi_{xd}(k)$ (11,81).

We now show the application of Wiener filter theory to adaptive noise cancelling. Fig. 3.6 shows a single channel adaptive noise canceller including an adaptive filter whose primary input consists of a signal s_j plus a sum of two noises m_{0j} and n_j. The reference input is a sum of two other noises m_{1j} and $n_j*g(j)$, where g(j) is the impulse response of the channel whose transfer function is G(Z). The noises n_j and $n_j*g(j)$ are correlated with each other and uncorrelated with the signal s_j and they are assumed to have a finite power spectrum at all frequencies, while m_{0j} and m_{1j} are uncorrelated with each other, with s_j and with n_j and $n_j*g(j)$. If one assumes that the adaptive process has converged and the minimum MSE solution has been found, then the adaptive filter is equivalent to a Wiener filter.

The optimal unconstrained transfer function of the adaptive filter is thus given by eqn. 3.4.3 and can be expressed as follows: The filter's input spectrum is

$$\delta_{xx}(Z) = \delta_{m1m1}(Z)+\delta_{nn}(Z)\left|G(Z)\right|^2 \qquad 3.4.4$$

where $\delta_{m1m1}(Z)$ is the spectrum of the noise m1 and $\delta_{nn}(Z)|G(Z)|^2$ is the spectrum of the noise n arriving via G(Z), and the cross power spectrum between the filter's input and the desired response input is

$$\delta_{xd}(Z) = \delta_{nn}(Z)\; G(Z^{-1}) \qquad 3.4.5$$

The Wiener transfer function is thus

$$\omega^*(Z) = \frac{\delta_{nn}(Z)\; G(Z^{-1})}{\delta_{m1m1}(Z)+\delta_{nn}(Z)|G(Z)|^2} \qquad 3.4.6$$

From eqn 3.4.6, it would appear that $\mathcal{W}^*(Z)$ is independent of the primary signal spectrum$\delta_{ss}(Z)$ and of the primary uncorrelated noise spectrum $\delta_{m0m0}(Z)$ (11,77,82).

An interesting special case is when m1 in the reference input is zero. Then $\delta_{m1m1}(Z)$ is zero and the optimal transfer function 3.4.6 becomes.

$$\mathcal{W}^*(Z) = 1/G(Z) \qquad 3.4.7$$

The performance of the single-channel noise canceller can be evaluated more generally by obtaining the ratio of the signal-to-noise density at the output $\rho_{out}(Z)$ to the signal-to noise density ratio at the primary input $\rho_{pri}(Z)$, so that

$$\frac{\rho_{out}(Z)}{\rho_{pri}(Z)} = \frac{\text{Primary noise power epectrum}}{\text{Output noise power spectrum}}$$

$$= \frac{\delta_{nn}(Z)+\delta_{m0m0}(Z)}{\delta_{\text{output noise}}(Z)} \qquad 3.4.8$$

and as seen from fig. 3.6

$$\delta_{\text{output noise}}(Z) = \delta_{m0m0}(Z)+\delta_{m1m1}(Z)\left|\quad {}^*(Z)\right|^2 + \delta_{nn}(Z)\left|\left[1-G(Z)\quad {}^*(Z)\right]\right|^2$$

If A(Z) and B(Z) are defined as the ratios of the spectra of the uncorrelated to the spectra of the correlated noises at the primary and the reference inputs, then

$$A(Z) = \frac{\delta_{m0m0}(Z)}{\delta_{nn}(Z)}$$

$$B(Z) = \frac{\delta_{m1m1}(Z)}{\delta_{nn}(Z)\left|G(Z)\right|^2}$$

respectively, then the transfer function 3.4.6 can be written as

$$\mathcal{W}^* = \frac{1}{G(Z)\left[B(Z)+1\right]} \qquad 3.4.9$$

Substituting the value of * in equation 3.4.8, it yields:

$$\frac{\rho_{out}(Z)}{\rho_{pri}(Z)} = \frac{\left[A(Z)+1\right]\left[B(Z)+1\right]}{A(Z)+A(Z)B(Z)+B(Z)} \qquad 3.4.10$$

This equation represents the ideal noise canceller performance which has single primary and reference inputs and stationary signals and noises. This expression is a good method for estimating the level of noise reduction to be expected in the case of using an ideal noise cancelling system.

It is obvious from 3.4.10 that the ability of a noise cancelling system to reduce noise is limited by the uncorrelated-to-correlated noise density ratios at the primary and reference inputs.

1) small A(Z)

$$\frac{\rho_{out}(Z)}{\rho_{pri}(Z)} = \frac{1+B(Z)}{B(Z)}$$

2) small B(Z)

$$\frac{\rho_{out}(Z)}{\rho_{pri}(Z)} = \frac{1+A(Z)}{A(Z)}$$

3) small A(Z) and B(Z)

$$\frac{\rho_{out}(Z)}{\rho_{pri}(Z)} = \frac{1}{A(Z)+B(Z)}$$

When A(Z) and B(Z) approach zero, $\rho_{out}(Z)/\rho_{pri}(Z) \longrightarrow \infty$, and in this case there will be a complete removal of the noise at the system output. When A(Z) and B(Z) are small, however, other factors limit the performance of the system. These factors include the finite length of the adaptive filter in practical systems, and misadjustment caused by gradient estimation noise in the adaptive process, discussed in section 3.7 (11).

3.5 The Recursive LMS Adaptive Filter

The IIR adaptive filtering problem has been studied for many years. In 1975 White (83) developed the MMSE gradient algorithm for application to a speech analyzer and synthesizer and to an equalizer in data transmission. Stearn and Elliot (84) have suggested an approach using the method of steepest descent. However, these algorithms involved a reasonably large amount of computations per iteration. In 1976 Fentuch (33) proposed a simplified algorithm similar to that of the FIR filter.

Since the non-recursive (transversal) adaptive filter has a finite impulse response, i.e., they can produce only zeros with no poles in the transfer function, this limits the capability of the transversal adaptive filter in many applications. To overcome this limitation, a recursive adaptive filter, which has the capability of producing poles as well as zeros in the filter transfer function, is described and is easily implemented using two LMS transversal adaptive filters as shown in fig. 3.7 (23,33).

Assuming that the filter, under static conditions, is described by its transfer function then (81):

$$G(Z) = \frac{Y(Z)}{X(Z)} = \frac{\sum_{i=0}^{N} a_i Z^{-i}}{1+\sum_{i=1}^{M} b_i Z^{-i}}$$

In the time domain, the input-output relation is:

$$y(n)T = \sum_{i=0}^{N} a_i x(n-i)T - \sum_{i=1}^{M} b_i y(n-i)T$$

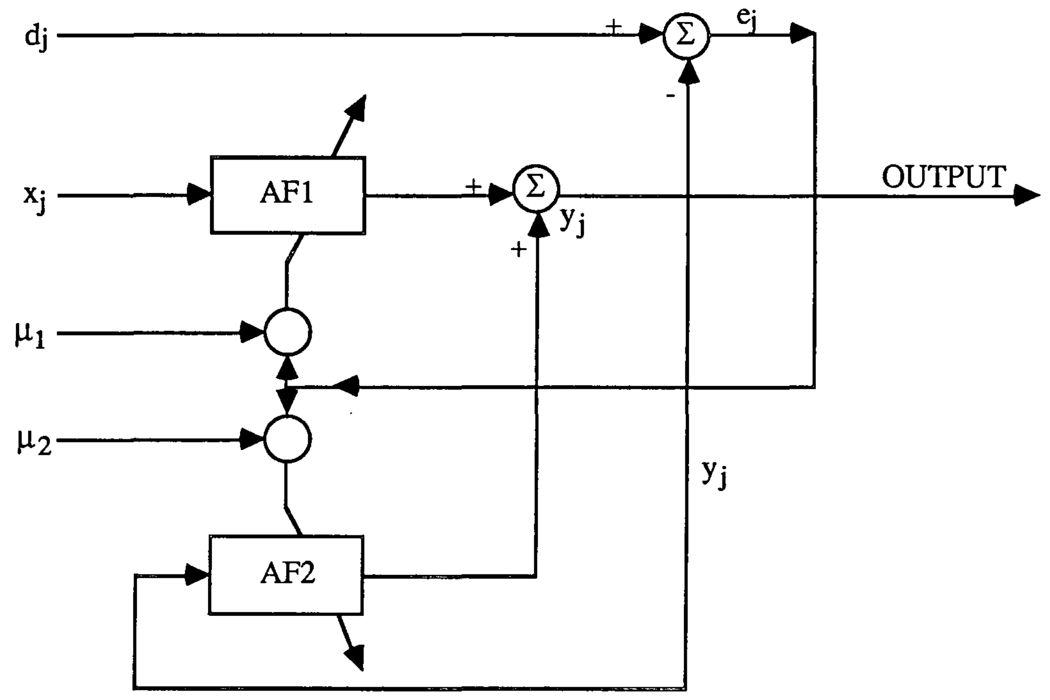


Fig. 3.7 Adaptive recursive filter constructed using two LMS transversal adaptive filters

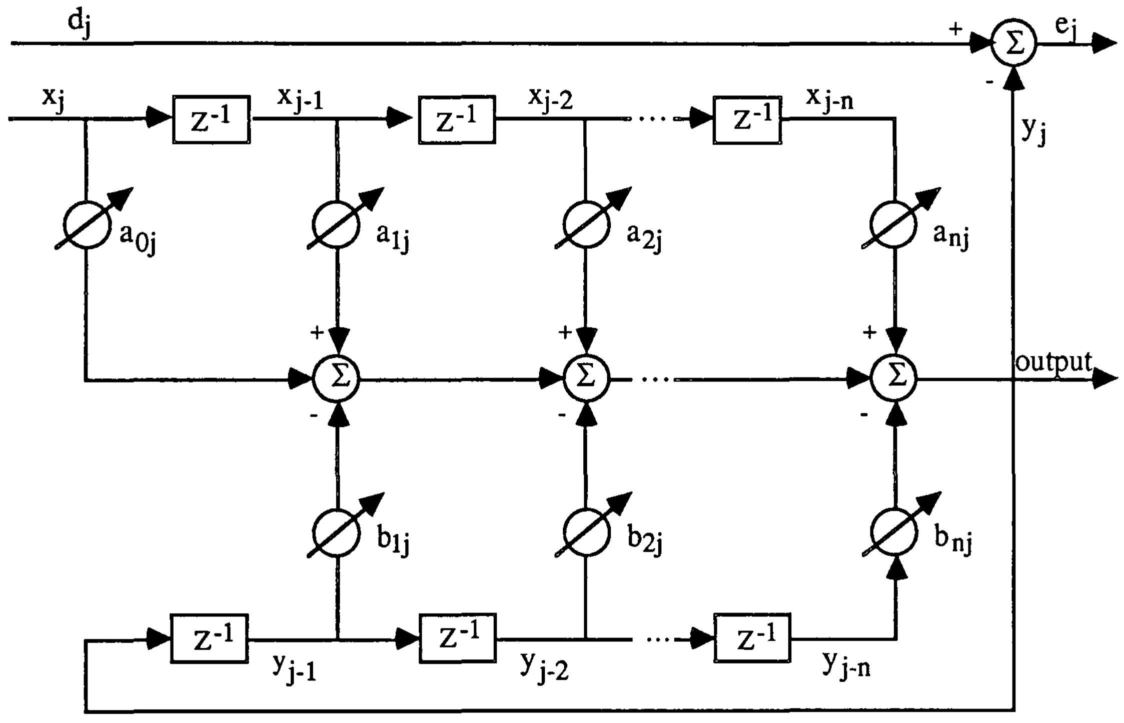


Fig. 3.8 The recursive adaptive noise canceller

Fig. 3.8 shows the block diagram of a recursive adaptive noise canceller where the set $\{a_i\}$ is referred to as the set of feed-forward coefficients and the set $\{b_i\}$ represents the feed-back coefficients. The LMS algorirthm is an implementation of the method of steepest descent and according to this method the next feed-forward and feed-back coefficients are:

$$\underline{a}_{j+1} = \underline{a}_j + 2\,\mu_1 e_j \underline{X}_j \qquad 3.5.1$$

and

$$\underline{b}_{j+1} = \underline{b}_j + 2\,\mu_2 e_j \underline{Y}_j \qquad 3.5.2$$

respectively (33,73).

3.6 Quantization Effects

The steady state output error of the LMS adaptive algorithm due to the limited word-length and the finite precision arithmetic of a digital processor consists of three terms (85):

(I) errors due to quantizing the input data.
(II) errors as a result of rounding the arithmetic operations used to calculate the filter's output; and
(III) the error due to the deviation of the filter's coefficients from the values they take when finite precision arithmetic is used. This last term is of particular interest.

It has been pointed out that in a digital adaptive filter, the weighting factors are controlled using the method of steepest descent employing gradient estimates.

As a result of the random nature of the input signal and the quantization noises arise due to the limited word-length used in the filtering operation and the coefficients updating, which consists essentially of additions and multiplications. The measured estimates of the gradient vector $\hat{\nabla}_j$ differs from the true value of the gradient ∇_j and contains an additive noise component $\underline{N}_j$. These differences are referred to as gradient measurement noise, with zero mathematical expectation (75,77):

$$\begin{aligned} \nabla_j &= -2e_j\underline{X}_j \\ \hat{\nabla}_j &= \nabla_j+\underline{N}_j \end{aligned} \qquad 3.6.1$$

where ∇_j is the true gradient, $\underline{N}_j$ is a zero mean-gradient estimation noise vector, and $\underline{X}_j$ is the input signal vector as mentioned before.

The adaptation process will be affected with gradient noise during both the initial transients and in steady state adaptation, the latter term is of particular interest. The gradient noise for steady state adaptation is:

$$\underline{N}_j = -2\left[e_j+\eta_d-\eta_y-\eta_w\sum_{i=1}^{n}\underline{X}_{i\,j}\right]\underline{X}_j \qquad 3.6.2$$

where η_d, η_y and η_w are the quantization noises for d, y and w respectively. They are stationary and independent of the adaptation stage i.e. of j (86,87).

According to the Wiener theory, when $\underline{W}_j = \underline{W}_j^*$, $\underline{X}_j$ and e_j are uncorrelated under steady state adaptation conditions, taking into account that the noises η_d, η_y and η_w are stationary and uncorrelated with each other and with $\underline{X}_j$ and e_j. Thus the covariance of function of the gradient noise $\underline{N}_j$, if all the components of eqn. 3.6.2 have zero mathematical expectations, is given by:

$$\mathrm{cov}\left[\underline{N}_j\right] = E\left[\underline{N}_j\underline{N}_j^T\right] = 4\left[\xi_{min}+\delta_d^2+\delta_y^2+\delta_w^2\ \Phi(\underline{R})\right]\underline{R} \qquad 3.6.3$$

where T is the sign of transposition, $\xi_{min} = E\left[e_j^2\right]$ is the MSE at the output of the adaptive filter when $\underline{W}_j = \underline{W}^*$; $\delta^2 = E\left[\eta^2\right]$ is the variance (expected value) of the quantization noise, $\underline{R} = E\left[\underline{X}_j\ \underline{X}_j^T\right]$ is the input correlation matrix, and for a Gaussian process X :

$$\Phi(\underline{R}) = 3\sum_{k,i=1}^{n} E\left[\underline{X}_{ij}\ \underline{X}_{kj}\right]^*$$

The input correlation matrix $\underline{R}$, may be reexpressed as:

$$\underline{R} = \underline{Q}\Lambda\underline{Q}^T \qquad 3.6.4$$

where $\underline{Q}$ is the orthomodal matrix of $\underline{R}$ and

$= \mathrm{diag}\left[\lambda_1, \lambda_2, \quad \dots, \quad \lambda\right]$ is the diagonal matrix of $\underline{R}$'s eigenvalues, we obtain for the gradient noise

$$\underline{N}_j' = \underline{Q}^T\underline{N}_j \qquad 3.6.5$$

and correspondingly from eqn. 3.6.5 considering eqns. 3.6.3 and 3.6.4. Its covariance becomes

$$\mathrm{cov}\left[\underline{N}_j'\right] = E\left[\underline{Q}^T\underline{N}_j\underline{N}_j^T\underline{Q}\right]$$

$$= 4\left[\xi_{min}+\delta_d^2+\delta_y^2+\delta_w^2\,\Phi(R)\right]\underline{\Lambda} \qquad 3.6.6$$

The components of $\underline{N}_j'$ are uncorrelated with each other since the matrix $\underline{\Lambda}$ is diagonal and they can be handled easily, while those for $\underline{N}_j$ are correlated.

This gradient noise causes noise in the weight vector and according to this noise, the LMS algorithm for updating the coefficients can be written as:

$$\underline{W}_{j+1}' = \underline{W}_j'+\mu(-\hat{\nabla}_j)$$

$$= \underline{W}_j' + \mu(-\nabla_j' + \underline{N}_j') \qquad 3.6.7$$

This equation can be expressed in term of $\underline{V}_j'$ as

$$\underline{V}_{j+1}' = (I - 2\mu_q\Lambda)\underline{V}_j' - \mu_q\underline{N}_j' + H_w' \qquad 3.6.8$$

where $\underline{V}$ is defined as the difference between $\underline{W}_j$ and the Wiener solution $\underline{W}^*$

$$\underline{V} = (\underline{W}_j - \underline{W}^*)$$

$\underline{V}_j' = \underline{Q}^T\underline{V}_j$; $\mu_q = \mu + \eta_\mu$; η_μ is the quantization noise of μ,

$\eta_w = \eta_v$; $H_w' = \underline{Q}^T H_w$; $H_w^T = [\eta_{1w}, \eta_{2w}, \ldots, \eta_{nw}]$.

For the steady state adaptation conditions, when the input signal $\underline{X}_j$ is stationary, the process $\underline{V}'$ is stationary and, consequently,

$$\text{cov}[\underline{V}_{j+1}'] = \text{cov}[\underline{V}_j'] \quad ,$$

and taking eqn. 3.6.6 into account, the covariance of the weight-vector noise is

$$\text{cov}[\underline{V}_j'] = [\xi_{min} + \delta_d^2 + \delta_y^2 + \delta_w^2\Phi(R)]\,\mu(1+\mu_g^2)$$
$$(I - \mu\underline{\Lambda})^{-1} + \delta_w^2(1/4)\underline{\Lambda}^{-1}(I - \mu\underline{\Lambda})^{-1} \ , \qquad 3.6.9$$

and $$\text{cov}[H_w'] = \delta_w^2 I$$

where $\mu_g = \delta_\mu / \mu$, the components of the weight-vector have equal $\underline{V}_j'$ variance and are matually uncorrelated. It has been found, however, that in this case the quantization noise of μ has the most effect on the gradient noise (77,87).

In the steady state adaptation, due to the quantization, random noise occurs in the weight vector and causes an excess MSE.

The excess between the MSE and the minimum MSE is the sum of variance of the weights calculated by the respective eigenvalues.

The mathematical expectation of the excess MSE of the adaptive filter may be written as $E\left[\Delta\xi(m)\right] = \sum_{i=1}^{n} \lambda_j E\left[\upsilon_i^2(m)\right] + \delta_y^2$ where υ_{ij}^2 is the variance of the weighting vector $\underline{\upsilon}_{ij}$.

The average excess MSE of the adaptive filter is an important quantity and can be obtained by assuming that $\lambda_i = \lambda_{med} = n^{-1}\mathrm{tr}\underline{R}$, for the matrix $\underline{R}$ where

$$\mathrm{tr}\underline{R} = \sum_{i=1}^{n} \lambda_i = \sum_{i=1}^{n} \left(E\left[x_i^2\right] + \left[\delta_x^2\right]\right)$$

is the trace of the matrix $\underline{R}$. In this case

$$\text{diagonal}\ (\underline{R}) = 3\ \mathrm{tr}\underline{R} \quad , \text{ and}$$

$$E\left[\nabla\xi(j)\right] = (\xi_{min} + \delta_d^2)(1+\mu_g^2)\frac{n\,\mu\,\mathrm{tr}\underline{R}}{n-\mu\,\mathrm{tr}\underline{R}} + \delta_y^2\left[1+(1+\mu_g^2)\right]\frac{n\,\mu\,\mathrm{tr}\underline{R}}{n-\mu\,\mathrm{tr}\underline{R}} + \delta_w^2(1/\mu)\ 3(1+\mu_g^2)\frac{n\,\mu\,\mathrm{tr}^2\underline{R}}{n-\mu\,\mathrm{tr}\underline{R}} + \frac{n^2}{4(n-\mu\,\mathrm{tr}\underline{R})} \qquad 3.6.9$$

Hence, for slow convergence of the minimum MSE algorithm $\mathrm{tr}\underline{R} \gg 1$, we then have:

$$E\left[\nabla\xi(j)\right] = (\xi_{min} + \delta_d^2)(1+\mu_g^2)\,\mu\,\mathrm{tr}\underline{R} + \delta_y^2\left[1+(1+\mu_g^2)\right]\mathrm{tr}\underline{R} + \delta_w^2(1/\mu)\left[3(1+\mu_g^2)\,\mu^2\mathrm{tr}^2\underline{R} + (n/4)\right] \qquad 3.6.10$$

and for the rapid convergence, $\mathrm{tr}\underline{R} = 1$ we have:

$$E[\nabla \xi(j)] = (\xi_{min}+\delta_d^2)(1+\mu_g^2)\frac{n}{n-1} + \delta_y^2\left[1+(1+\mu_g^2)\right]\frac{n}{n-1} + \delta_w^2 \, tr\underline{R} \left[3(1+\mu_g^2)+(n/4)\right]\frac{n}{n-1} \qquad 3.6.11$$

It follows from eqn.s 3.6.10 and 3.6.11 that the contribution to the output effect of filtering the quantization noise y, the value of which cannot be less than δ_y^2, increases with increased rate of convergence, i.e., as $\mu tr\underline{R}$ increases, and in the mode where $tr\underline{R}=1$ it approaches a value of $2\delta_y^2$. The contribution of the quantization noise of the weighting factors is proportional to the input power and to the number of weighting units in the adaptive filter, and consists of two components, one of which acts through the gradient noise, and the second by direct formation of y (87).

Caraiscos and Liu (85) have derived expressions for the steady state mean square quantization error when fixed and floating point arithmetic are used, and found them to be similar.

3.7 Misadjustment Due to Gradient Noise

It is obvious that the purpose of adaptation is the minimization of the MSE. However, according to the gradient noise, an excess between the measured estimate MSE and the MMSE exists. This excess MSE governs the quality and the performance of the adaptive filter and it is a very important factor. The dimensionless ratio of the average excess MSE in an adaptive solution to the minimum possible MSE is referred as the misadjustment, thus:

$$M = \frac{\text{average excess MSE}}{\lambda min}$$

For the LMS algorithm

$$M = \mu \, \mathrm{tr}\underline{R} \qquad 3.7.1$$

As can be seen, the misadjustment depends on the rate of convergence, so it is related to the speed of adaptation and it is also related to the number of weights, because it depends on tr$\underline{R}$.

Since tr$\underline{R}$ is the sum of eigenvalues, then:

$$M = \mu \sum_{i=1}^{n} \lambda_i = \mu n \lambda_{ave} \qquad 3.7.2$$

where λ_{ave} represents the average value of the eigenvalues

$$\text{If} \quad \lambda_{ave} = 1/4\mu \, (1/\ \tau_{iMSE}) \qquad 3.7.3$$

where τ_i is the adaptation time constant.
Substituting value of λ_{ave} into the eqn. (3.7.2) yields:

$$M = n/4 \ (1/\tau_{iMSE}) \qquad 3.7.4$$

The important special case is when all eigenvalues are similar, so in other words, all time constants are equal. Thus the misadjustment is given by:

$$M = n/\ \tau_{MSE} \qquad 3.7.5$$

From eqn. 3.7.5 it is seen that M increases linearly with number of weights and is inversely proportional to the time constant of the adaptation, i.e. decreases with the decrease of μ (75,77,88).

3.8 Conclusion

Adaptive filters generally consist of two distinct parts: a filter, whose structure is designed to perform a desired processing function, and an algorithm for adjusting the coefficients of that filter. The LMS algorithm, as one of the simplest and easiest algorithm to implement, is used to update these coefficients. It has been shown that this algorithm is based on the gradient technique and that its convergence or divergence is governed by the proper choice of μ. It is noticed that the LMS algorithm presented for the IIR filter is similar to that for the FIR LMS adaptive filter. The quantization effects due to the limited word length available on the digital computer and the effect of the misadjustment by the rate of convergence has also been investigated.

CHAPTER -4-

Computer Simulation

4.1 Intoduction

The principal motivation in the study of adaptive IIR filters is the significant computational and therefore hardware saving possible versus FIR filters. The structure of that filter in the time domain may be described, as has been pointed out, by the input-output relation

$$y(n)T = \sum_{i=0}^{N} a_i x(n-i)T - \sum_{i=1}^{M} b_i y(n-i)T \qquad 4.1.1$$

The sets $\{a_i\}$ and $\{b_i\}$ can be updated using the LMS algorithm. This algorithm has gained considerable popularity since the early 1960s. Its simplicity and ease of implementation make it an attractive solution for many practical problems.

As has been established in chapter 3, the LMS algorithm for updating the feed-forward and feed-back coefficients are given by the expressions

$$\underline{a}_{j+1} = \underline{a}_j + 2\mu_1 e_j \underline{X}_j$$
$$4.1.2$$
$$\underline{b}_{j+1} = \underline{b}_j + 2\mu_2 e_j \underline{Y}_j$$

respectively.

In this chapter, a number of computer simulation programs were developed to investigate the behaviour of the IIR adaptive filter, implemented as an ANC, under different conditions and with various parameters. So to be compatible with the hardware implementation of the algorithm, computer simulations for both floating point and fixed point arithmetic representations of the IIR adaptive filter were also discussed.

4.2 Computing IIR Filter Transfer Function

It has been mentioned in chapter 2 that the frequency response (transfer function) of an IIR filter may be expressed as

$$G(Z) = \frac{a_0 + a_1 Z^{-1} + a_2 Z^{-2} + \dots + a_{N-1} Z^{-(N-1)}}{1 + b_1 Z^{-1} + b_2 Z^{-2} + \dots + b_{M-1} Z^{-(M-1)}} \qquad 4.1.3$$

It can be easily obtained by substituting $e^{j\omega T}$ for Z in G(Z), and computing directly $|G(e^{j\omega T})|$, by considering the parameter (ωT) varies from 0 to π. A computer simulation was developed to satisfy this requirement and is listed in appendix A-1.

4.3 IIR Adaptive Filter Performance with Different N and μ

In the hardware implementation of the IIR adaptive filter, the speed of the processor used become a hindrance limiting the execution of the coefficients updating operation for each sampling instant. Various simulation programs, listed in appendix A-2 through A-5, were developed to investigate the performance of the filter when the coefficients updating process takes place only once for N sampling points. Four different types of implementing the IIR adaptive filter including the conventional type have been discussed. The last three types vary in the time interval between the input samples as well as the output samples involved in the coefficients updating operation, but for all these types one new input and ouput sample is applied whenever this operation is accomplished.

In these programs, two different types of signals are applied to the IIR adaptive filter.

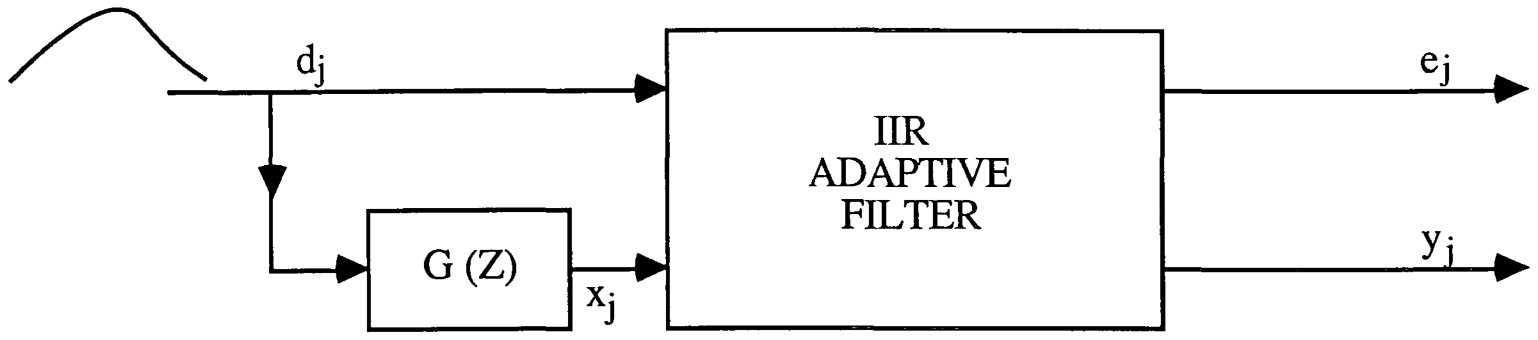
(a) White noise (or random numbers) was supplied to the desired response input d_j, and the filter input x_j was d_j waveform passed through a system with transfer function $G(Z) = 1+Z^{-2}$ as illustrated in fig. 4.1.a

(b) x_j and d_j were supplied from two different sources. The d_j was obtained by summing two equal amplitude sine waves, the required fundamental and interference second harmonic. The filter input was the interference second harmonic signal passed through the filter with $G(Z) = 1+Z^{-2}$ as depicted in fig. 4.1.b. This simple second order filter produces a zero at $\pi/2$ in its steady state transfer function.

Both the filter input and the desired response input are simulated in both previous systems. A common flowchart of the computer simulation program for implementing the filter is shown in fig. 4.2. The performance of the filter is evaluated by plotting its transfer function (magnitude of the transfer function) with 16 feed-forward and 15 feed-back coefficients and the types of implementation are as follows:

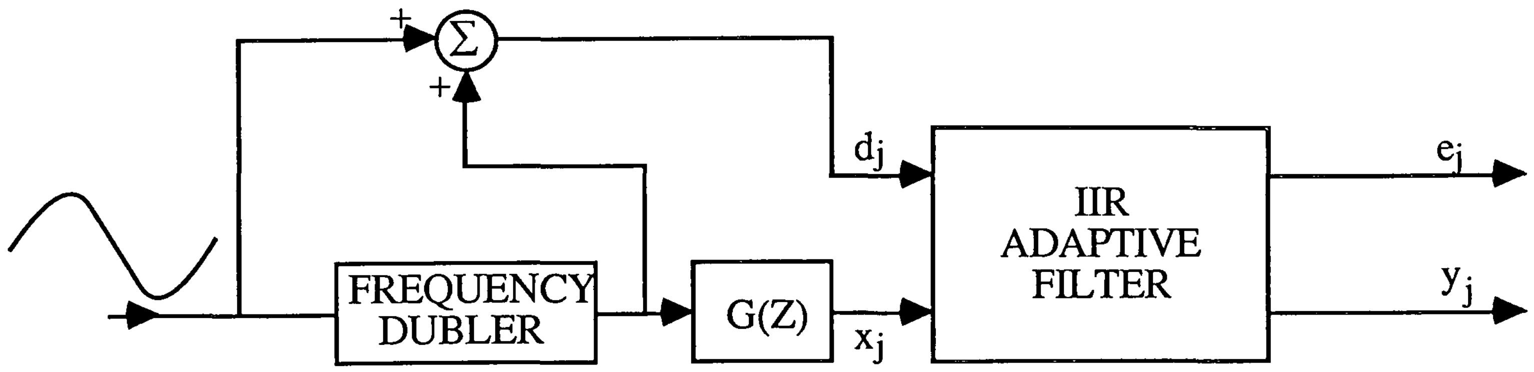
4.3.1 **Conventional Type**

The simulation program (appendix A-2) of this type is very straightforward, so referring to the flowchart in fig. 4.2, there is no need for C, K, S and H parameters and N=1 since the coefficients are updated every sampling instant. In this type the input and output samples are expressed as x_j, x_{j-1}, ..., x_{j-15}, y_{j-1}, y_{j-2}, y_{j-15}. Fig. 4.3 illustrates the steady- state transfer function of the IIR adaptive filter implemented according to this type, the filter inputs follow the system (a) (fig.4.1.a). While



d_j
G (Z)
x_j
IIR ADAPTIVE FILTER
e_j
y_j

(a)



+
Σ
+
FREQUENCY DUBLER
G(Z)
x_j
d_j
IIR ADAPTIVE FILTER
e_j
y_j

(b)

Fig. 4.1

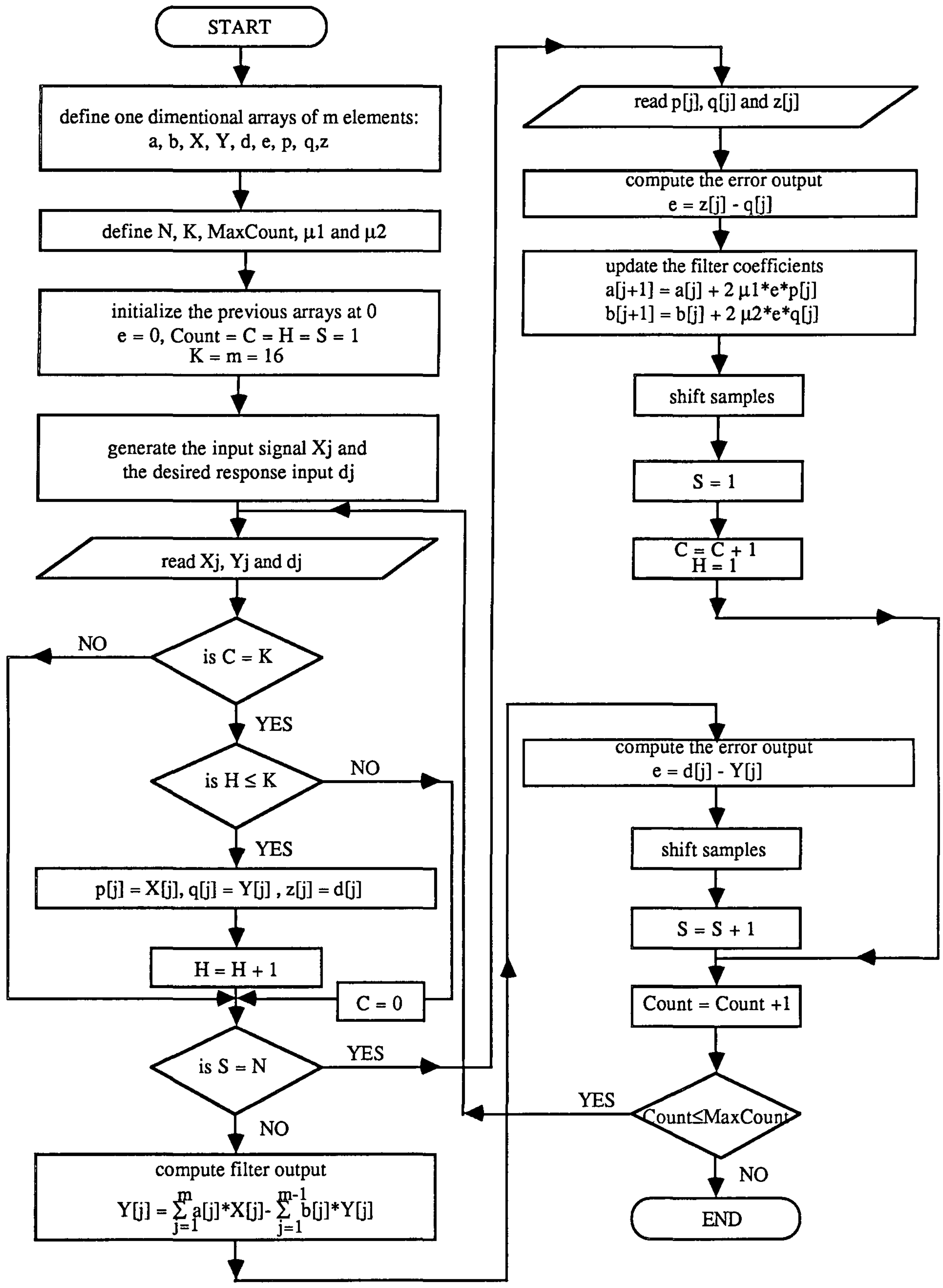
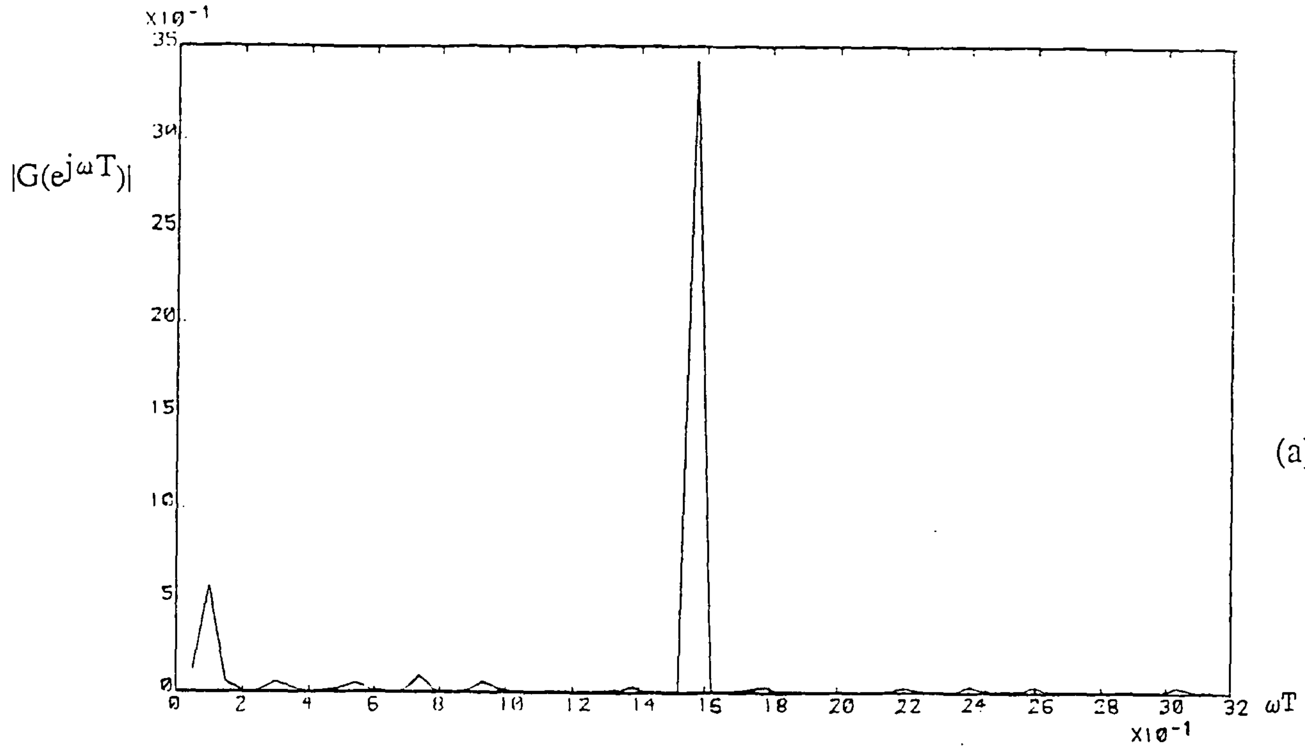
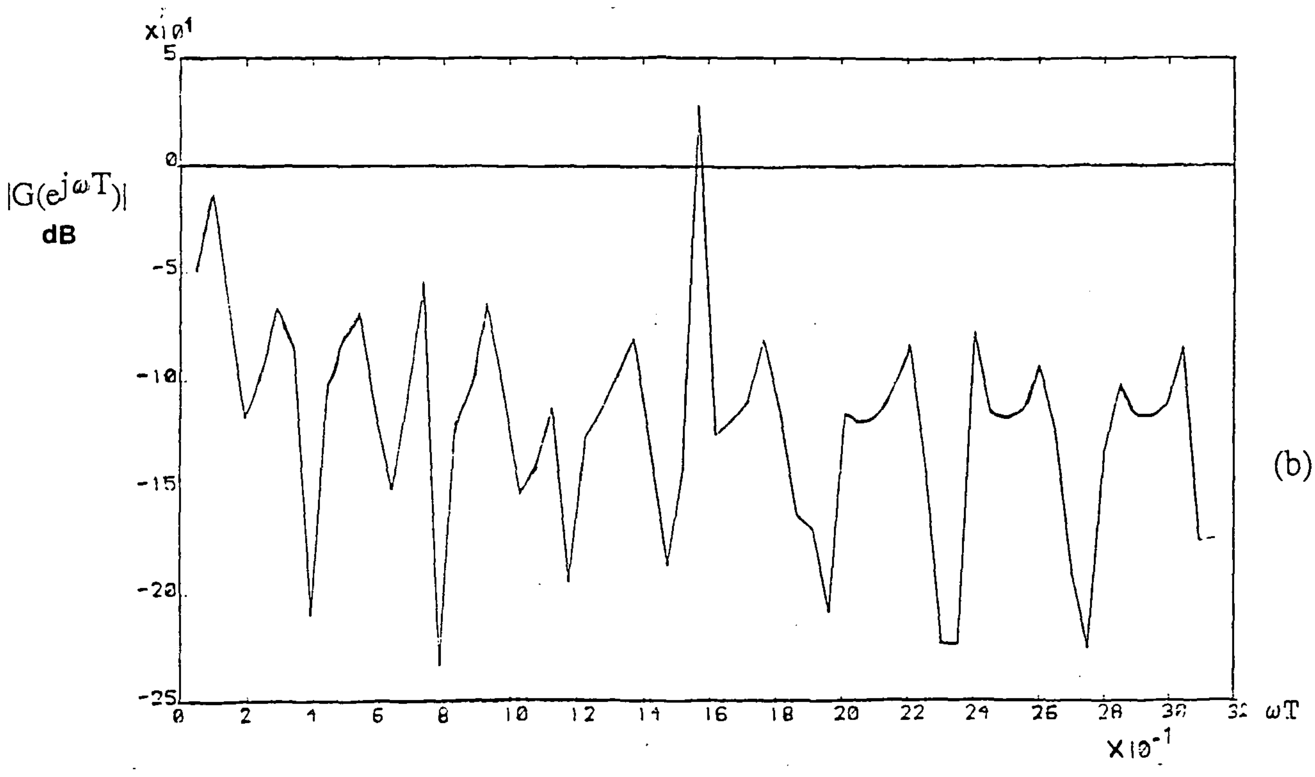

START
define one dimentional arrays of m elements:
a, b, X, Y, d, e, p, q,z
define N, K, MaxCount, µ1 and µ2
initialize the previous arrays at 0
e = 0, Count = C = H = S = 1
K = m = 16
generate the input signal Xj and
the desired response input dj
read Xj, Yj and dj
is C = K
NO
YES
is H ≤ K
NO
YES
p[j] = X[j], q[j] = Y[j] , z[j] = d[j]
H = H + 1
C = 0
is S = N
YES
NO
compute filter output
Y[j] = Σ_{j=1}^{m} a[j]*X[j]- Σ_{j=1}^{m-1} b[j]*Y[j]
read p[j], q[j] and z[j]
compute the error output
e = z[j] - q[j]
update the filter coefficients
a[j+1] = a[j] + 2 µ1*e*p[j]
b[j+1] = b[j] + 2 µ2*e*q[j]
shift samples
S = 1
C = C + 1
H = 1
compute the error output
e = d[j] - Y[j]
shift samples
S = S + 1
Count = Count +1
Count≤MaxCount
YES
NO
END

Fig. 4.2 The IIR LMS adaptive filter flowchart

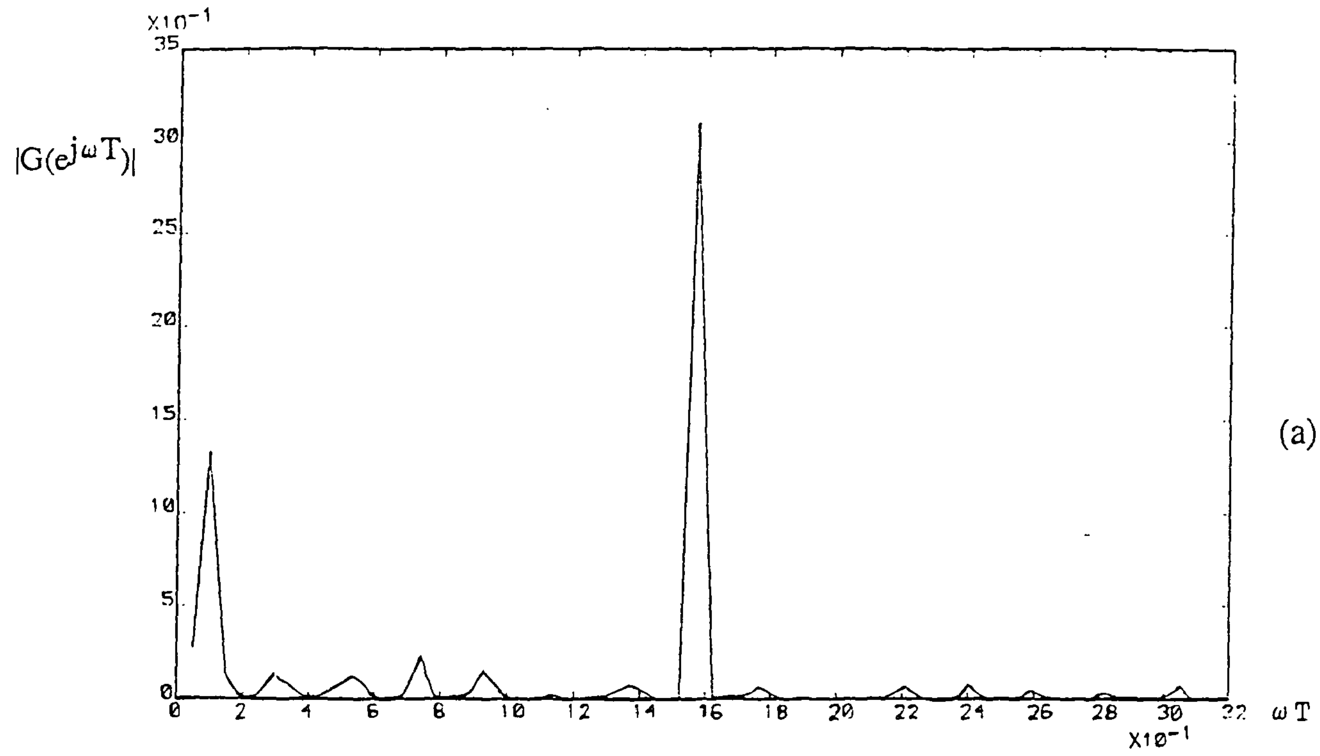


(a)



(b)

Fig. 4.3 The Adaptive filter transfer function
$\mu 1 = 7.2\text{x}10^{-6}$, $\mu 2 = 3.6\text{x}10^{-6}$
(a) and (b) N = 1



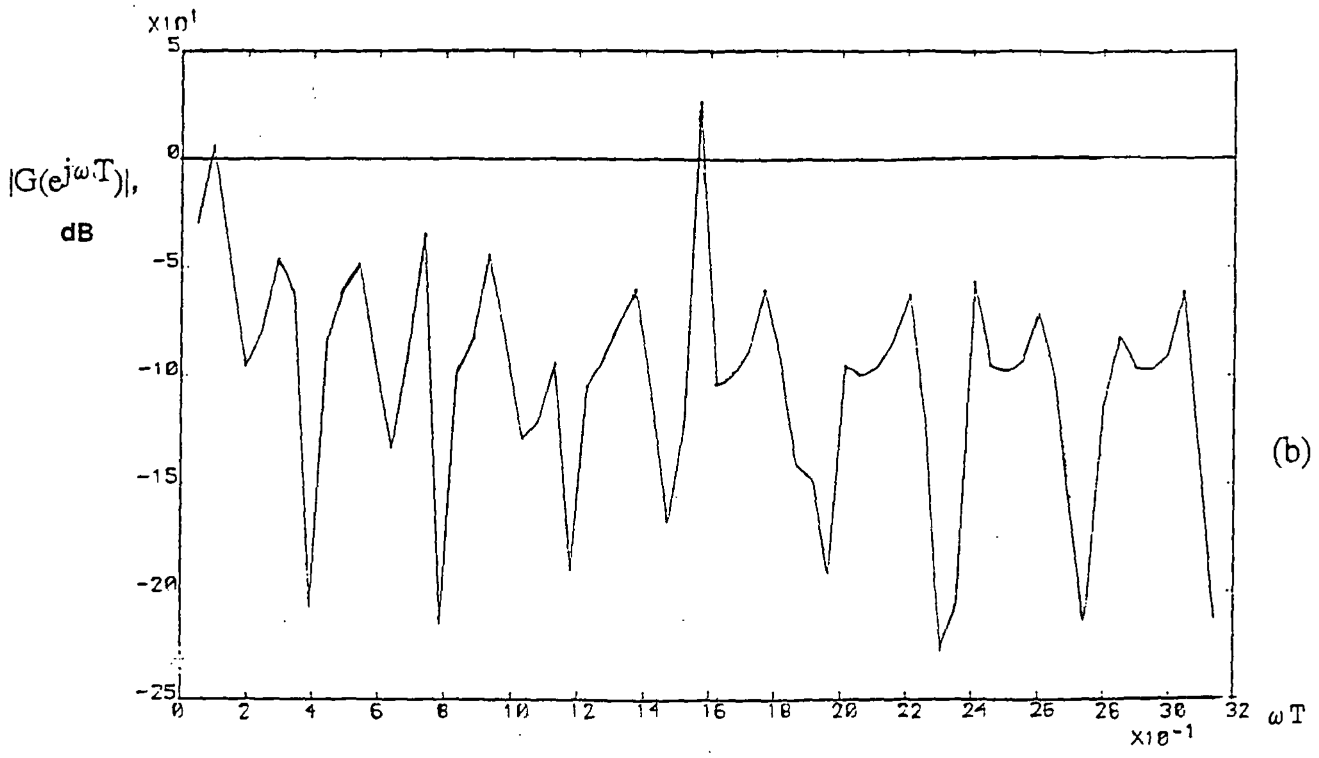


Fig. 4.4 The Adaptive filter transfer function
$\mu 1 = 7.2\times10^{-6}$, $\mu 2 = 3.6\times10^{-6}$
(a) and (b) N = 1

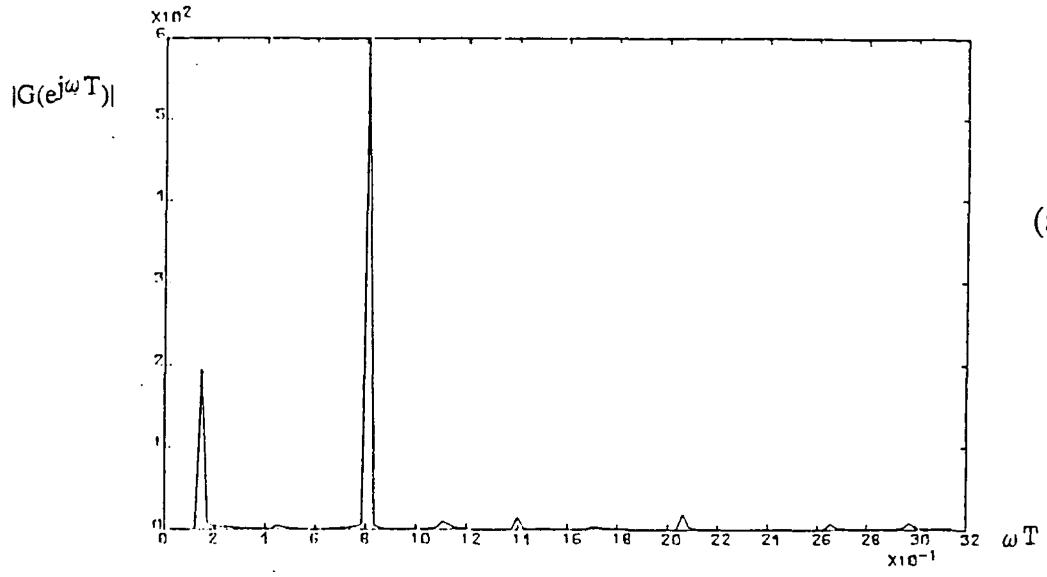
fig. 4.4 shows the filter transfer function by supplying inputs of system (b) to the filter. The convergence factors μ_1 and μ_2 are set to the values $7.2\text{x}10^{-6}$ and = $3.6\text{x}10^{-6}$ respectively.

4.3.2 N Time Interval Type

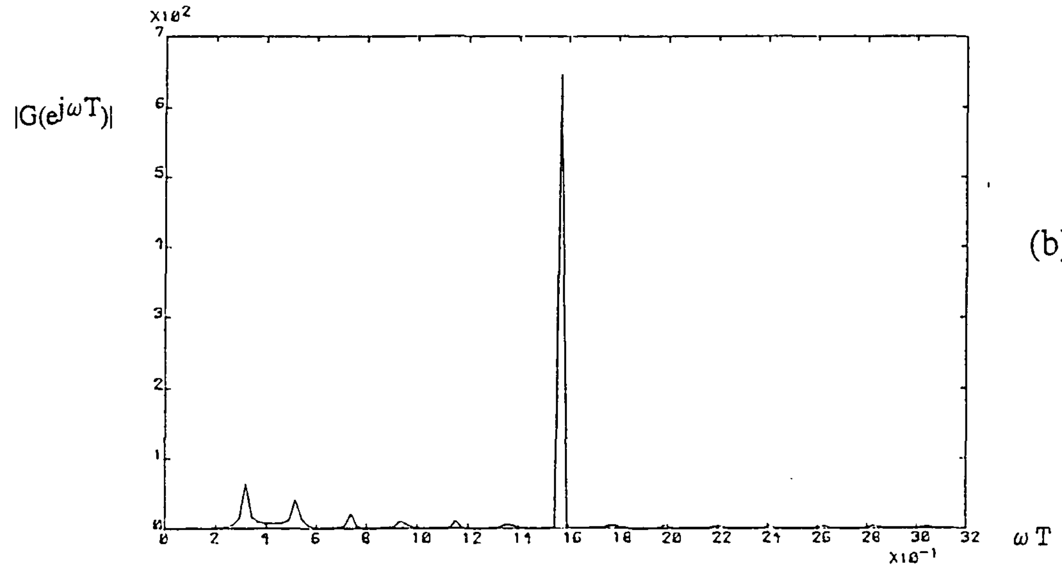
In this type the inputs applied to the ANC follow the system (b) and the times between the input as well as the output samples involved in the updating operation are N. The sampled data are expressed in the form x_j, x_{j-N}, x_{j-2N}, ..., x_{j-15N}, y_{j-1}, y_{j-1-N}, y_{j-1-2N}, ..., $y_{j-1-14N}$. In this program, listed in appendix A-3, there is no need for K ,C and H has only one value, two parameters are varied individually, and N. Fig. 4.5 illustrates the steady-state transfer function of the filter for a constantμ_1= $7.2\text{x}10^{-4}$ andμ_2= $3.4\text{x}10^{-4}$, while N takes different values, for example 16, 32 and 128. Fig. 4.6 displays the corresponding diagrams at different convergence factors μ_1 =$7.2\text{x}10^{-6}$ and μ_2 =$3.6\text{x}10^{-6}$.

4.3.3 Successive Input & Output Samples Type

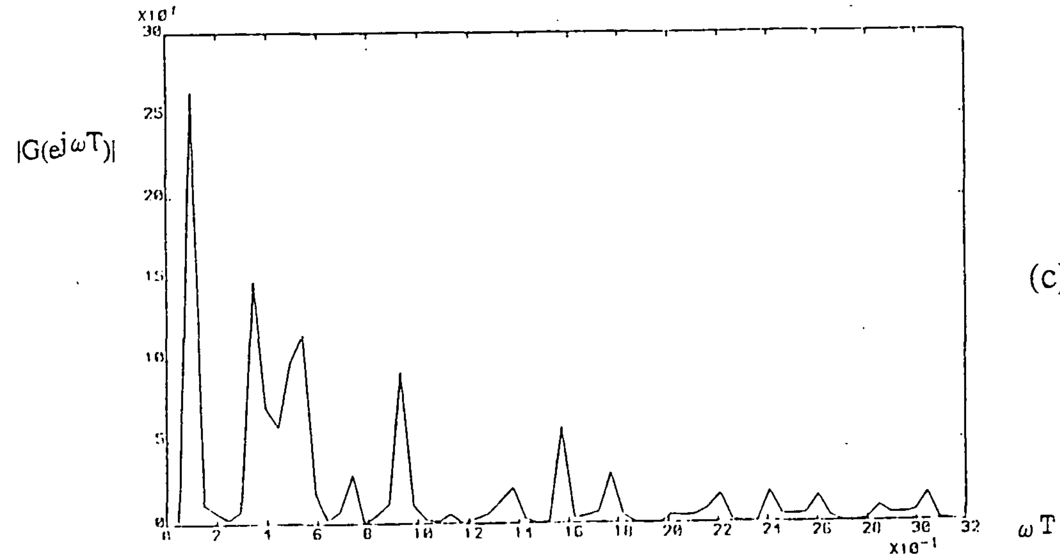
In this type the input and output samples are expressed as x_j, x_{j-1}, x_{j-2}, ..., x_{j-15}, y_{j-1}, y_{j-2}, y_{j-3}, ..., y_{j-15}. Referring to the flowchart in fig. 4.2, the computer simulation of this program follows the flowchart with the assumption that K and C are always 16, H is extended from 1 to 16. So the 16-input and the 16-output samples (only 15 output samples are used) are completely changed and replaced by a new group of successive samples whenever the coefficients updating operation takes place. Fig. 4.7 illustrates the steady-state transfer function of the filter for the convergence factors μ_1 = $7.2\text{x}10^{-4}$ and μ_2 = $3.4\text{x}10^{-4}$ at different values of N (16, 32 and 128), while fig. 4.8 shows the transfer function diagrams at the same values of N, but the filter



(a)

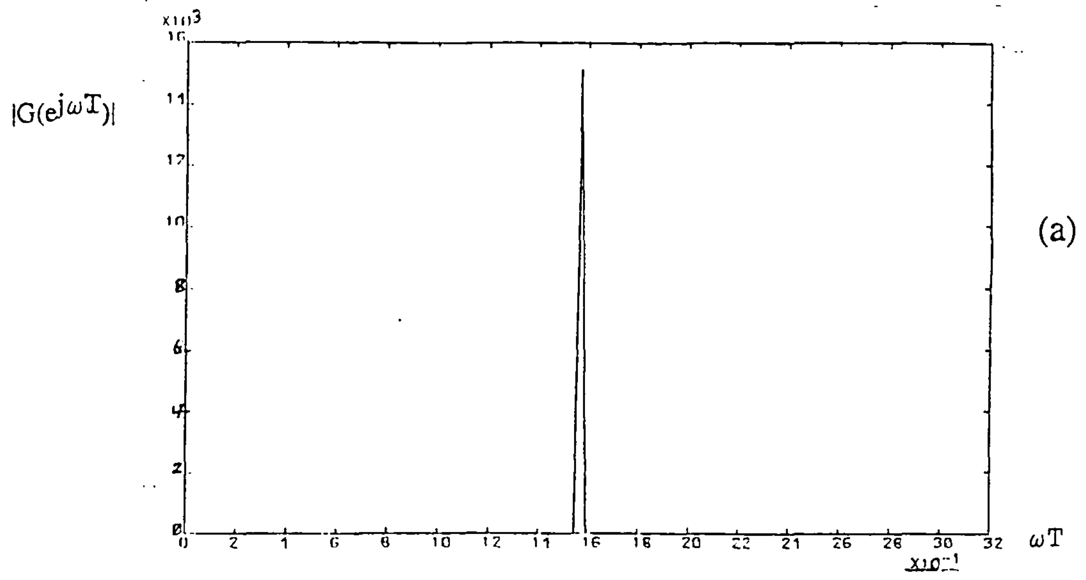


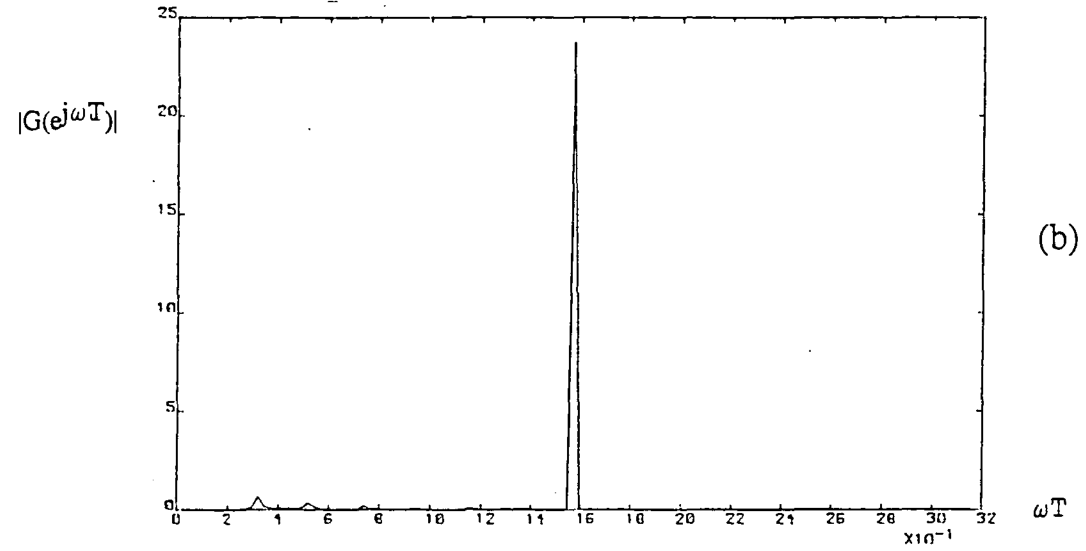
(b)



(c)

Fig. 4.5 The Adaptive filter transfer function
$\mu_1 = 7.2\text{x}10^{-4}$, $\mu_2 = 3.4\text{x}10^{-4}$
(a) N = 16, (b) N = 32, (c) N = 128





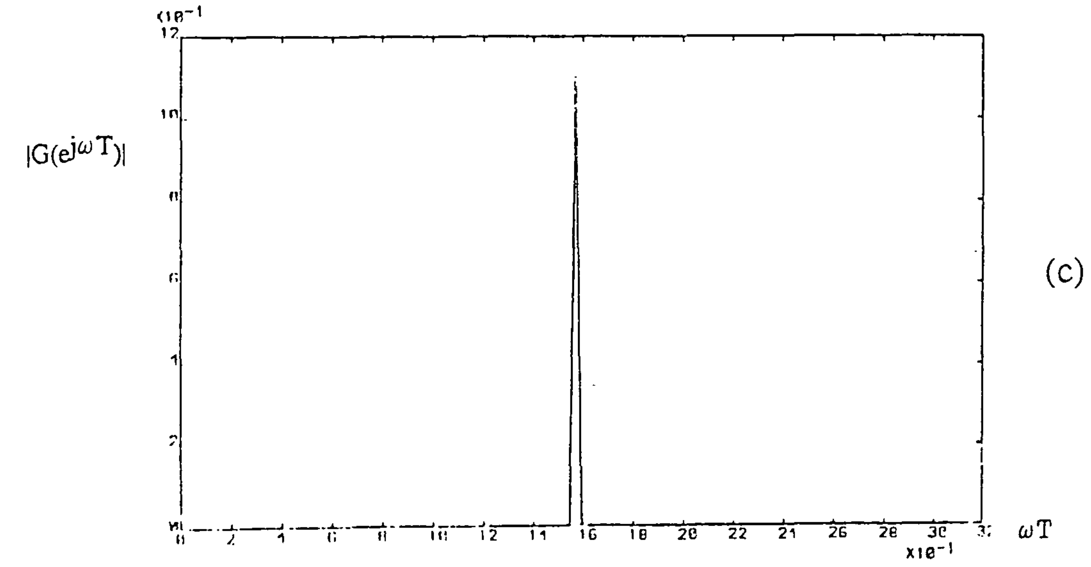
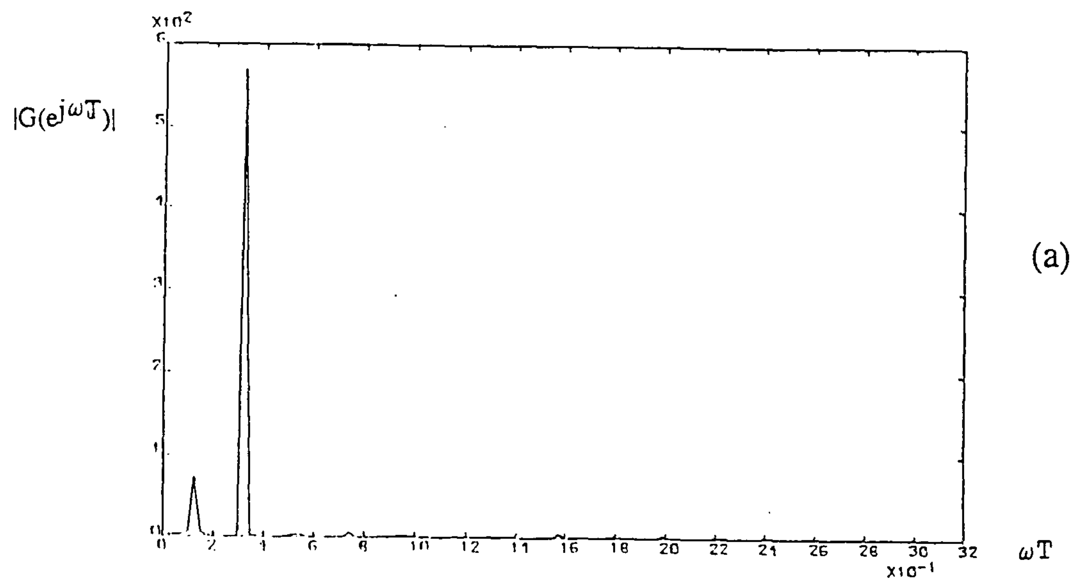
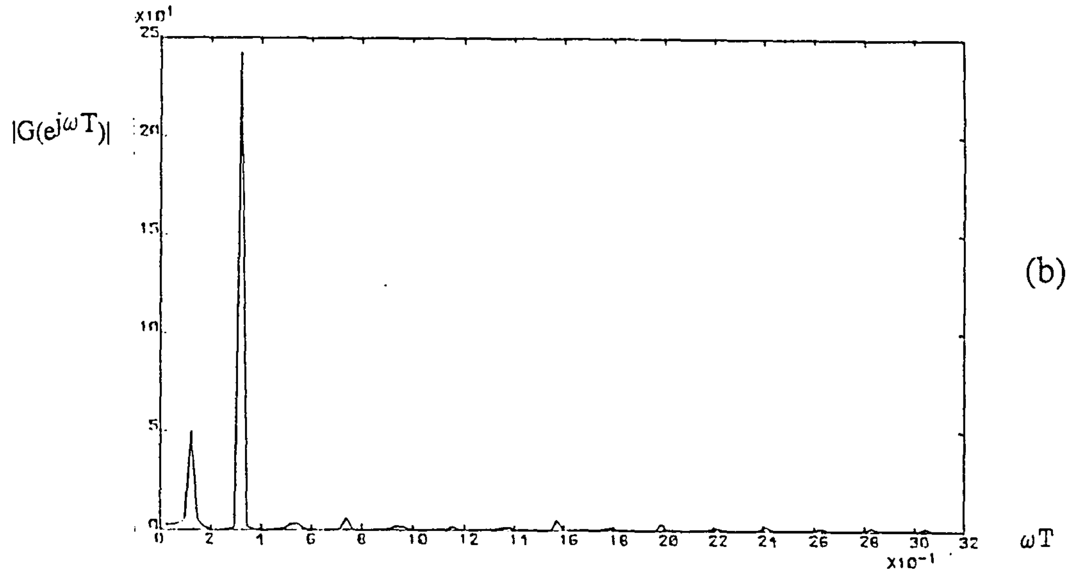


Fig. 4.6 The Adaptive filter transfer function
$\mu_1 = 7.2\text{x}10^{-6}$, $\mu_2 = 3.6\text{x}10^{-6}$
(a) N = 16 , (b) N = 32 , (c) N = 128





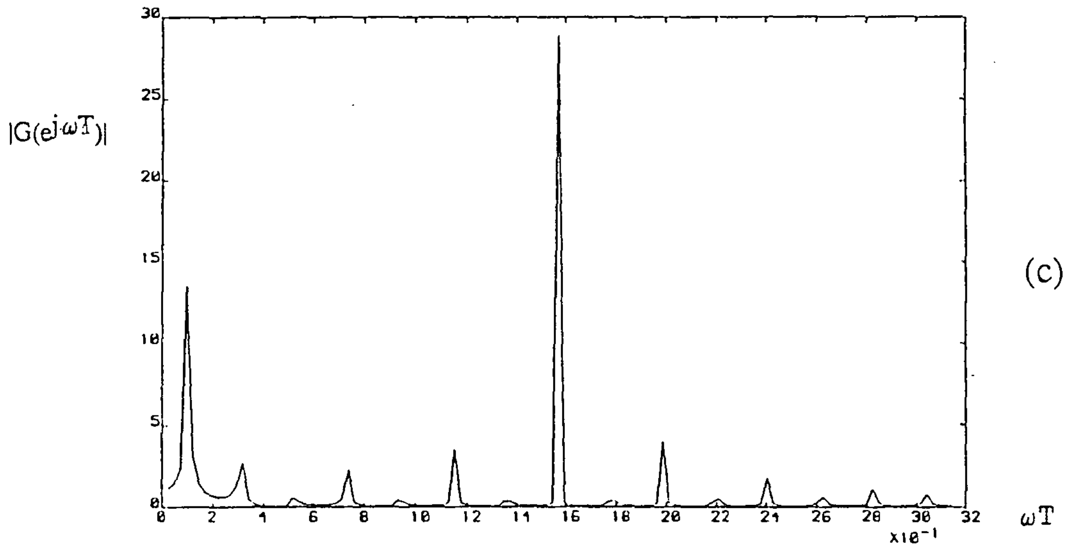
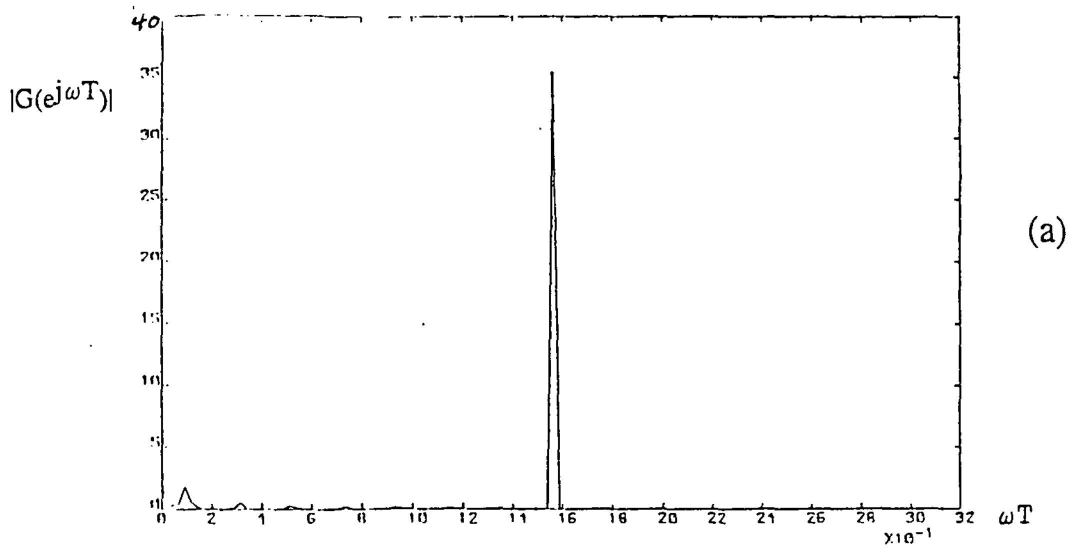
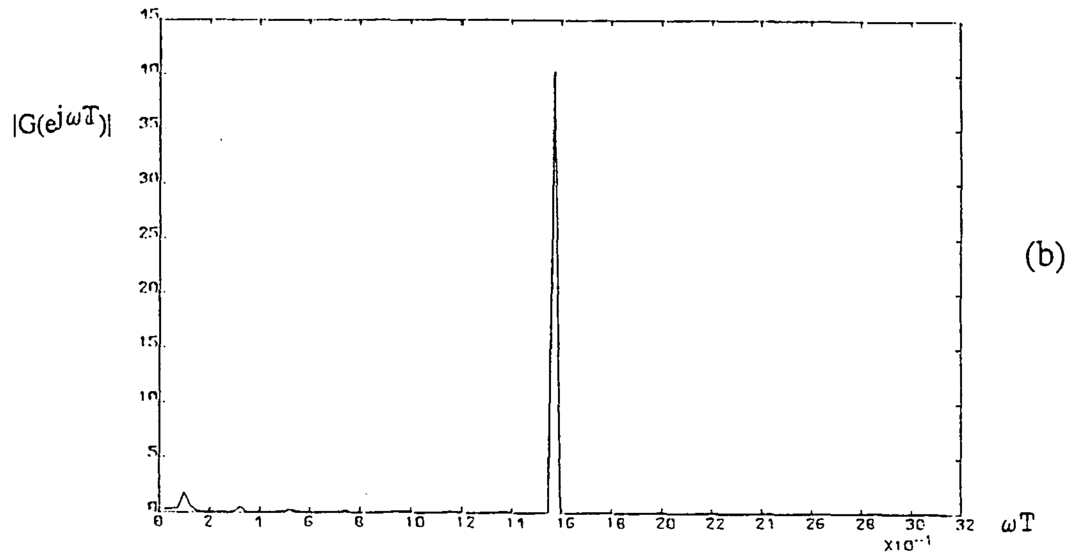


Fig. 4.7 The Adaptive filter transfer function
$\mu_1 = 7.2\times10^{-4}$, $\mu_2 = 3.4\times10^{-4}$
(a) N = 16, (b) N = 32, (c) N = 128





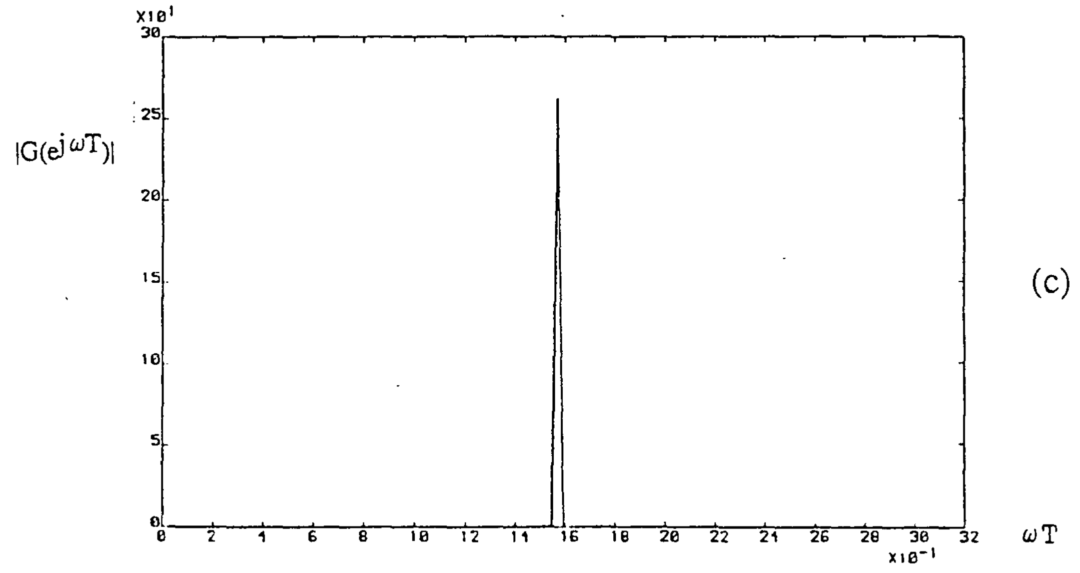


Fig. 4.8 The Adaptive filter transfer function
$\mu_1 = 7.2\times10^{-6}$, $\mu_2 = 3.6\times10^{-6}$
(a) N = 16, (b) N = 32, (c) N = 128

convergers with different values, $\mu_1 = 7.2\text{x}10^{-6}$ and $\mu_2 = 3.6\text{x}10^{-6}$.

4.3.4 Blocks Type

Two blocks (groups) of C (16) successive samples with time interval N*16 between the blocks are defined in this program, listed in appendix A-5. In this case, one new sample of the second block is applied to the first block, in which the samples are shifted one location and ripple through the first block for every coefficients updating operation, this process will be discussed in detail in chapter 6. The input and output samples are represented in the form x_j, x_{j-1}, x_{j-2}, ..., x_{j-15}, $x_{j-(N*16)}$, $x_{j-(N*16)-1}$, ..., $x_{j-(N*16)-15}$, y_{j-1}, y_{j-2}, ..., y_{j-15}, $y_{j-(N*16)-1}$, $y_{j-(N*16)-2}$, ..., $y_{j-(N*16)-15}$. The simulation program follows the flowchart in fig. 4.2, with K=16, and the maximum value of C is 16. The transfer function of the filter following this process is shown in fig. 4.9 and fig. 4.10, for N=16, 32 and 128 at two different values of the convegence factors $\mu_1 = 7.2\text{x}10^{-4}$, $\mu_2 = 3.4\text{x}10^{-4}$ and $\mu_1 = 7.2\text{x}10^{-6}$, $\mu_2 = 3.6\text{x}10^{-6}$ respectively

4.4 Comparisons of the Filter Implementation Types

The aim of previous simulations is to gain the optimal performance in the adaptive filter implementation. As a first step in evaluating the performance, various types of implementation have been simulated. The simulations helped to determine the optimal implementation of processor function and parameters as well as to expose certain methods of implementing the algorithm. On the other hand, it is better to avoid methods which could significantly degrade the IIR adaptive filter performance.

Gaining some insights into the previous results (figures), the

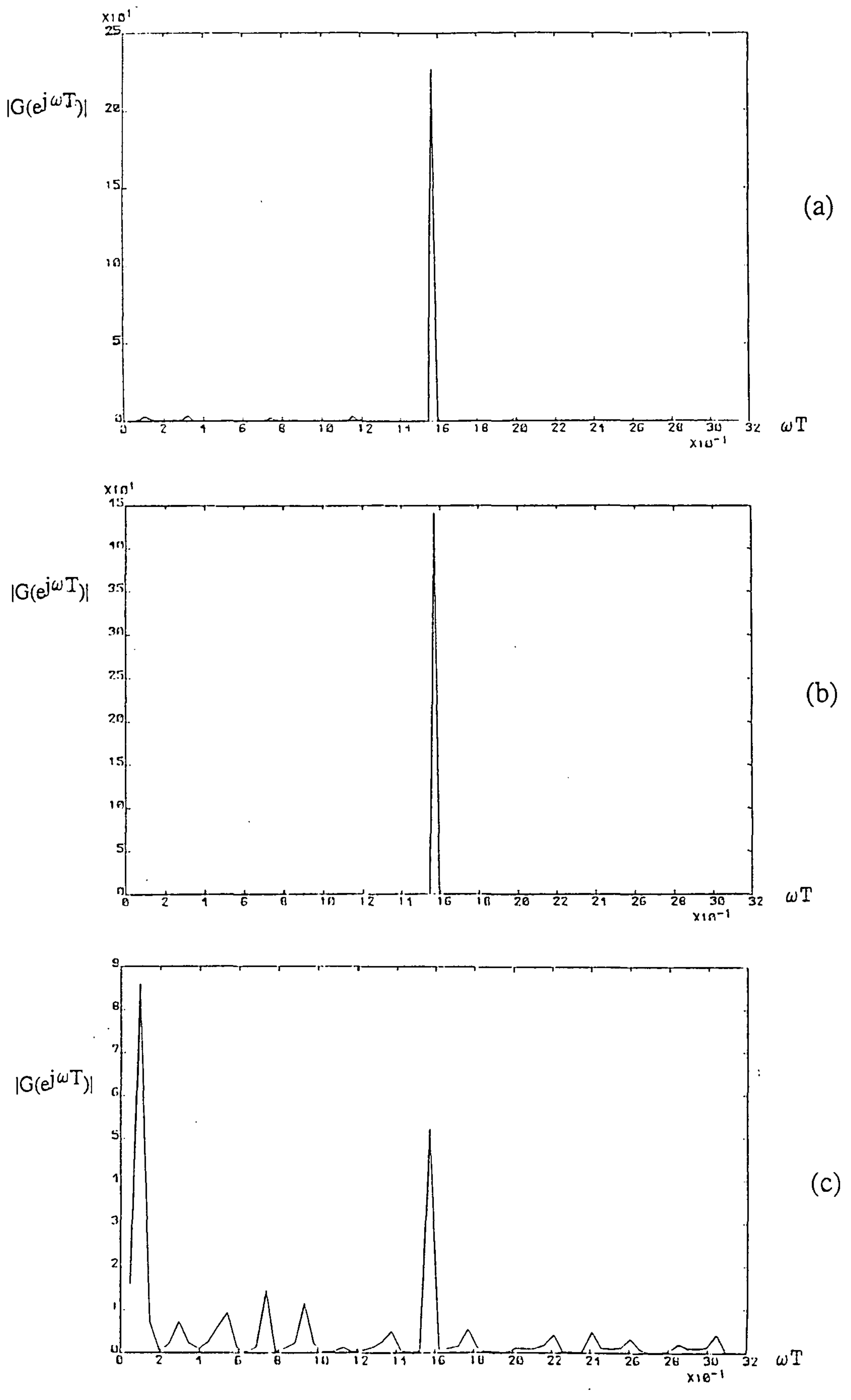
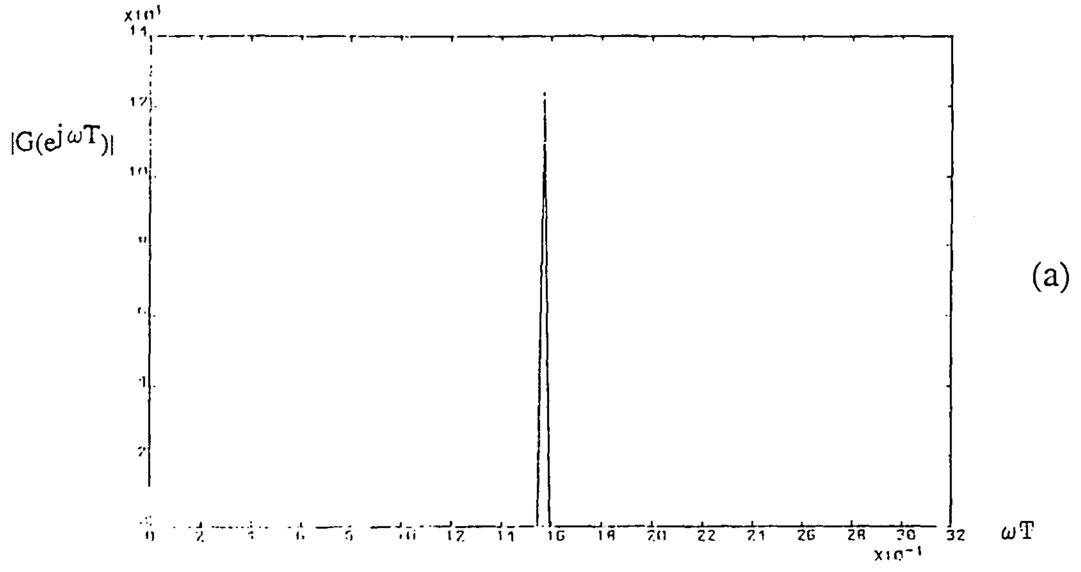
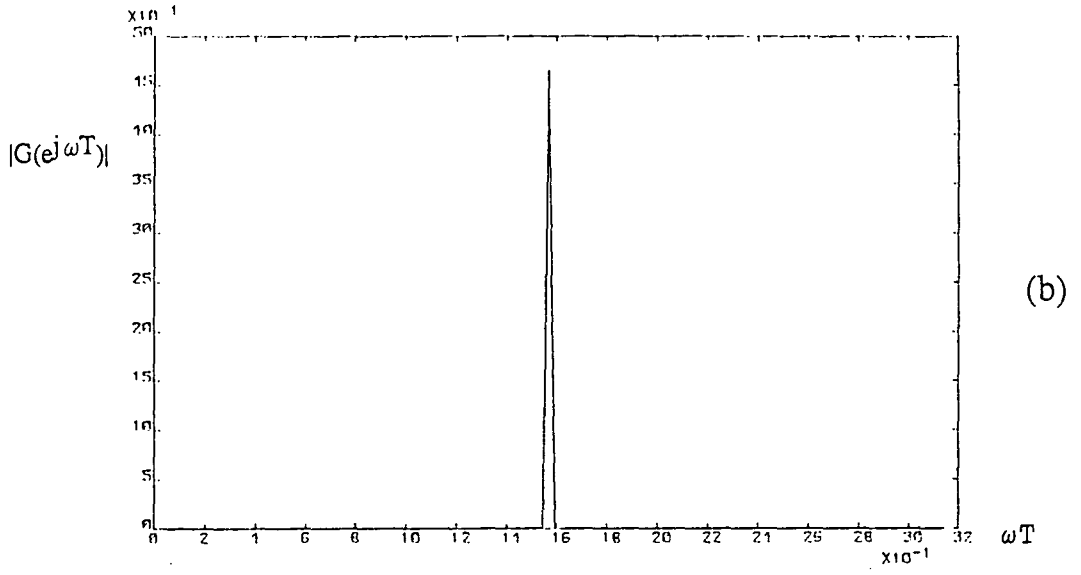


Fig. 4.9 The Adaptive filter transfer function
$\mu_1 = 7.2\text{x}10^{-4}$, $\mu_2 = 3.4\text{x}10^{-4}$
(a) N = 16 , (b) N = 32 , (c) N = 128





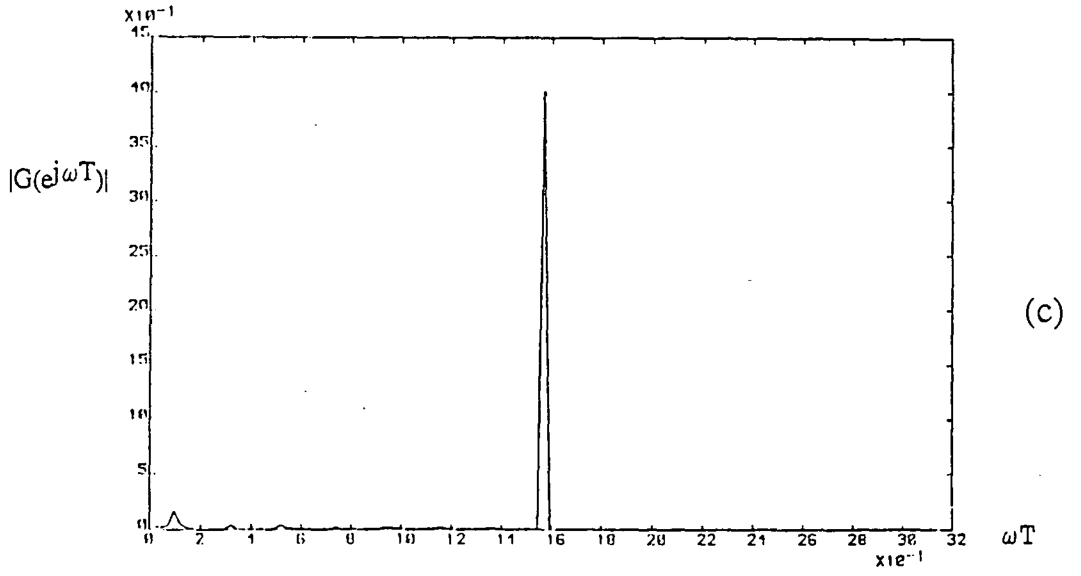


Fig. 4.10 The Adaptive filter transfer function
$\mu_1 = 7.2\text{x}10^{-6}$, $\mu_2 = 3.6\text{x}10^{-6}$
(a) N = 16 , (b) N = 32 , (c) N = 128

following aspects can be observed.

(a) There is no doubt that the LMS IIR adaptive filter can be efficiently employed as an ANC (fig.4.3&4.4 ,N=1).

(b) When the filter inputs are statistically stationary, the best steady-state performance results from slow adaptation. In other words, small convergence factors lead to a better performance.

(c) At small convergence factors (for example $\mu_1 = 7.2\times10^{-6}$ and $\mu_2 = 3.6\times10^{-6}$) in the last three types of implementation, the IIR adaptive filter developed a sharp peak or a pole at the correct frequency, which is $\pi/2$, and its transfer function is the inverse of the input transfer function, which produces a zero at this position ($\pi/2$), at various values of N.

These results illustrate and prove the ability and efficiency of the IIR adaptive filter to produce the desired performance when the coefficients updating operation takes place only once in N sampling points. It is obvious that the steady-state transfer function of the filter is improved and refined (lower side lobes) by increasing the value of N. So this operation is equivalent to the convergence of the filter using small μs, which leads to a more stable system.

(d) Comparing the performance of the filter in the last three types, there is no doubt that the IIR LMS adaptive filter exhibits better performance utilizing the type discussed in section 4.3.4.

This type of implementation is suitable for a wide range of convergence factors and for various values of N. Thus, for these reasons, in addition to the fact that in this type the input and output samples represent the periodic signal more efficiently, the microprocessor software is developed to update the coefficients of the filter implemented in hardware according to this type, as will be investigated in the next two chapters.

4.5 The ANC Performance

A sine wave was used as a test signal to enable any distortion due to noise contamination to be measured. This type of signal has been used to illustrate the efficiency and the ability of the IIR adaptive filter to cancel the interference corrupting the desired signal. A simulation program was developed to implement a 16th order LMS IIR adaptive filter in accordance with section 4.3.4.

Most adaptive filter simulations produce a set of floating-point coefficients and the input and output samples are also represented in floating-point arithmetic. For hardware implementation, all these data are represented by a number of bits depending on the ADC and DAC resolutions and the register length of the microprocessor used to perform the algorithm that updates the coefficients. So the simulation programs were executed using either floating-point arithmetic or fixed-point arithmetic representations of the data involved in the filter implementation. The desired response signal was generated by adding two equal amplitude sine waves, a desired fundamental of f_1 and the second harmonic of f_2 (where $f_2 = 2f_1$) representing the interference (noise). The filter input signal was the interference at f_2 altered in phase and amplitude in both modes of representation. The canceller comprised an IIR filter with 16 non-recursive (feed-forward) and 16 recursive (feed-back) coefficients taking into account that $a[1] = 1$ ($a_0 = 1$,eqn. 4.1), and N was chosen to be 128.

4.5.1 Floating-Point Simulation

This simulation program, listed in appendix A-6, followed the flowchart illustrated in fig.4.2, with some modification as regards the type discussed in section 4.3.4. So K and the maximum value of C were set to 16 and the updating operation accomplished once for every 128 input and output samples involved in the operation. An example of the performance of the filter is depicted in fig. 4.11. It shows typical desired response and the interference inputs and the corresponding noise canceller outputs. The adaptation constants μ_1 = 7.2e-06 andμ_2 = 7.2e-06.

4.5.2 Fixed Point Simulation

This section concerns the implementation of the IIR adaptive filter with fixed-length input-output and coefficients. The simulation program followes the flowchart in fig.4.2, as listed in appendix A-7 but with an important consideration taken into account ie. that the x_j, d_j, y_j, and e_j took the values between $2^{n-1}-1$ and -2^{n-1} according to the ADC and DAC resolutions (12-bits (n=12)) used in the hardware implementation of the ANC.

The coefficients were considered as a fractions of 12-bit size. This could be achieved by scaling down the filter output y_j by 2^{-12} at the end of the filtering operation for each sampling instant (period). The ANC system used was identical to the one employed in the floating point representation. An example of the behaviour of the ANC in fixed point representation is illustrated in fig. 4.12. It shows the typical primary and reference inputs and the ANC outputs at the end of the convergence (adaptation) with convergence factors $\mu_1 \simeq 1020033$ (2^{-19}) and $\mu_2 \simeq 1020033$ (2^{-19}).

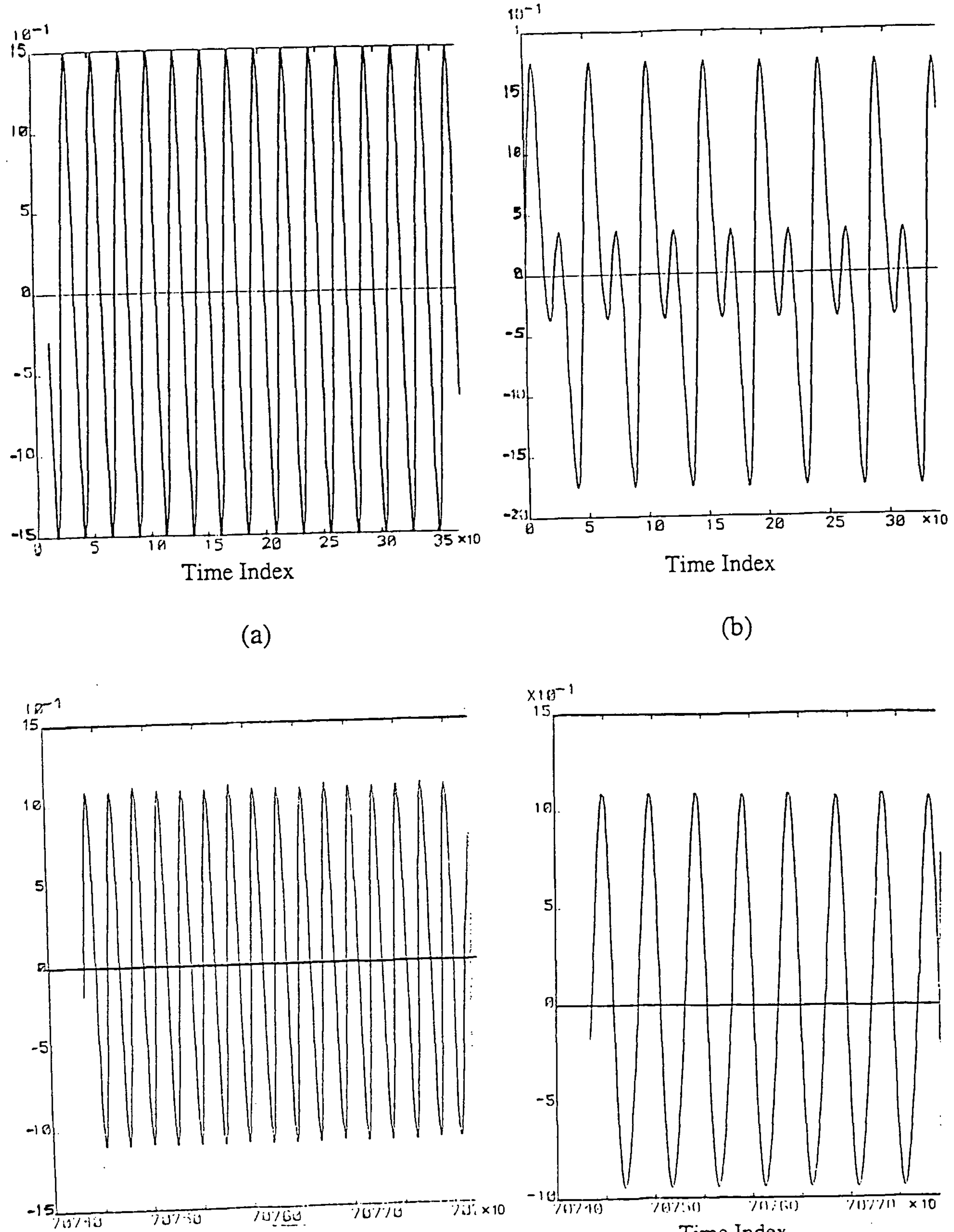


Fig. 4.11 Simulation result showing the cancellation performance of the adaptive filter
(a) signal input
(b) desired response input
(c) filter output
(d) error output

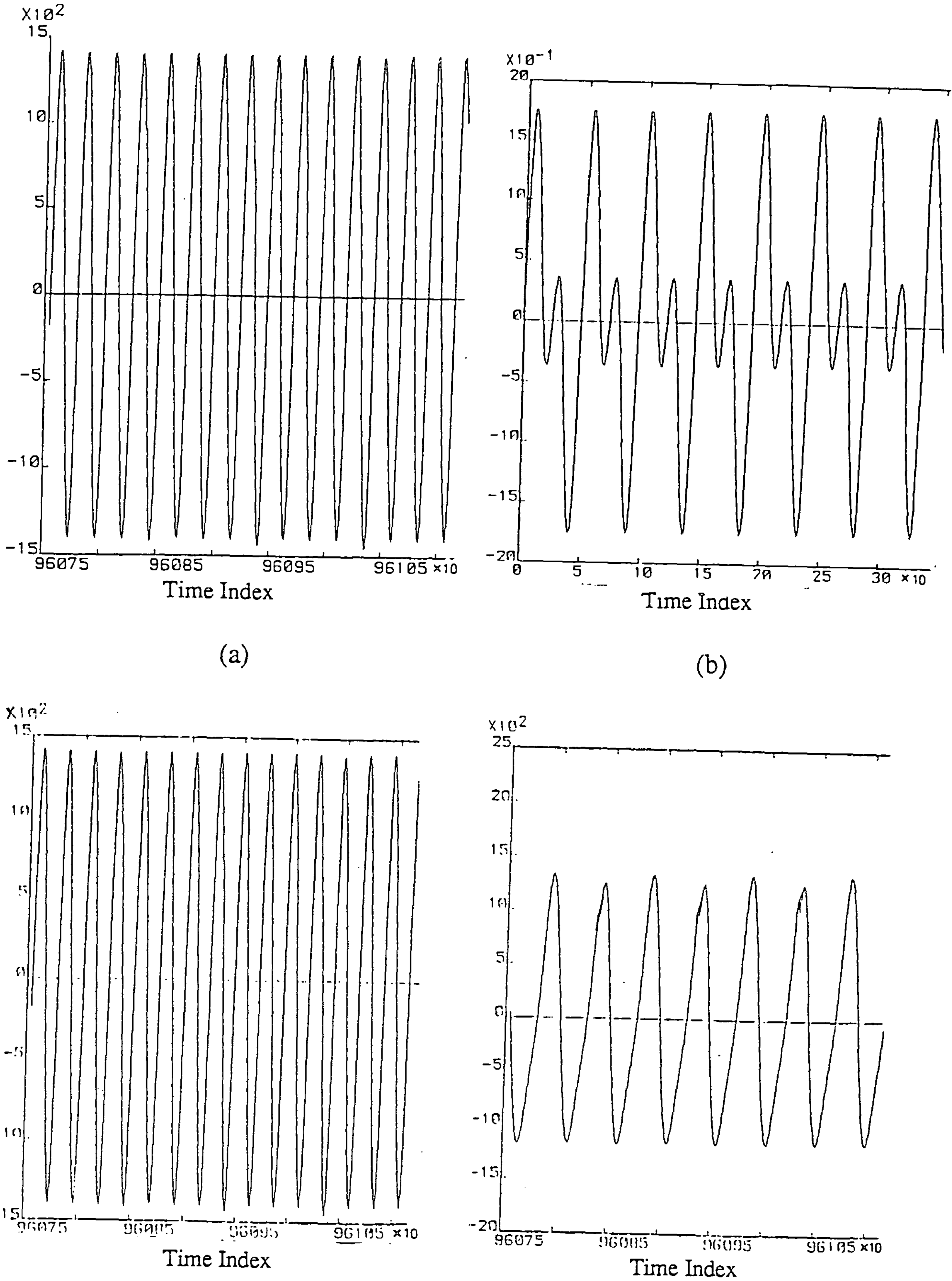


Fig. 4.12 Simulation result showing the cancellation performance of the LMS adaptive filter
(a) signal input
(b) desired response input
(c) filter output
(d) error output

4.6 Conclusion

Several computer simulation programs with different characteristics have been successfully developed verifying the optimality and efficiency of the ANC when the coefficients updating operation is achieved only once in N sampling instants yielding a more stable system. These simulation results greatly demonstrate the competency of the noise canceller to adaptively filter the additive periodic noise in both floating point and fixed point representations of data involved in the design of the ANC.

CHAPTER -5-

DESIGN ANALYSIS

5.1 Introduction

In recent years, attention has been focused on digital hardware in communications equipment. As progress has continued in the digital environment, a whole new range of signal processing problems have required digital solutions; many of them can be expressed in their simple form as either filtering or a level detection function.

The advent of LSI made it convenient to carry out many filtering operations digitally rather than with analogue circuits, so that it is possible to perform all the arithmetic processing and shifting operations on one chip.

The FAD IC was designed at the British Telecom Research Laboratories to perform the role of a general purpose programmable digital filter and detector suitable for the flexible processing of signals in the audio frequency band.

It has been mentioned in the introductory chapter that the aim of this work is the design and implementation of a high order adaptive filter using the FAD IC acting as a noise cancelling system with the aid of a fast microprocessor.

This chapter presents the analysis and design, in hardware, of the noise cancelling system which is controlled by a 68000 microprocessor as illustrated by the block diagram in fig 5.1.

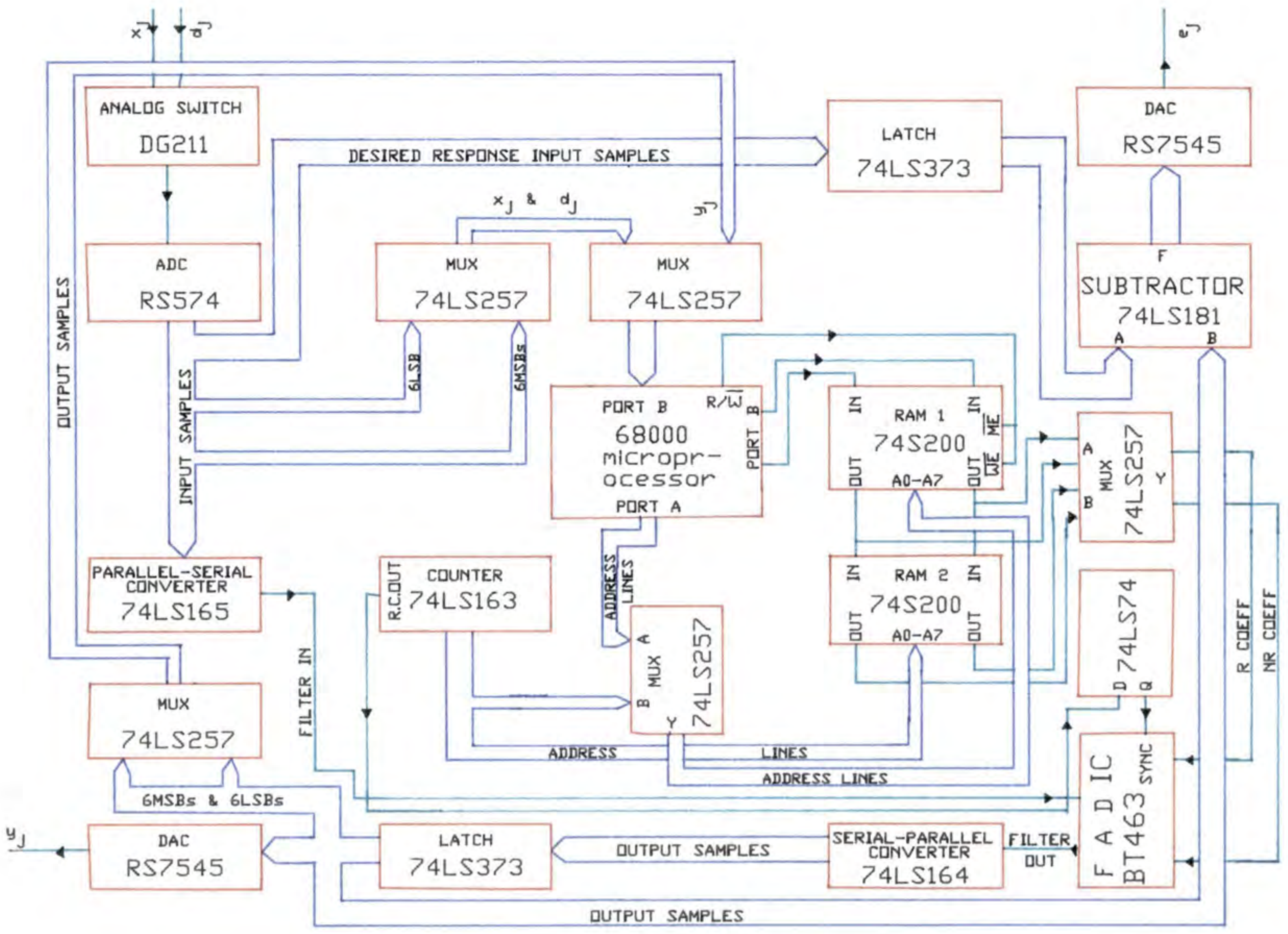


Fig.5·1 The Adaptive Noise Canceller Block Diagram

It introduces the hardware configuration of the 68000 microprocessor and also the FAD IC which represents the heart of the noise cancelling system. Interfacing considerations and problems associated with the efficient hardware implementation of the filtering algorithms are also discussed. Fig. 5.2 shows the ANC system interfaced with the 68000 microcomputer board, while fig. 5.3 (a) and (b) show the top and bottom views of the ANC system.

5.2 Structure of the FAD Chip

The FAD chip is realized in 5μNMOS and can be clocked in the range of 0.5-3.00 MHz. It is assembled in a 24 pin DIL package and has serial I/O in order to minimize the number of pins. There are two independent sections of the FAD IC, the filter section and the level detection section. The first one is of particular interest (41).

The filter section is realized as a second order recursive canonic form (which uses a minimum number of delays, multipliers and adders) which consists of four 16*13 serial/parallel multipliers, shift registers as delay elements and a number of adders as shown in fig. 5.4.

The transfer function of such a filter is governed by

$$G(Z) = S \frac{1+AZ^{-1}+BZ^{-2}}{1-aZ^{-1}-bZ^{-2}}$$

where the multiplier coefficients A and B define a pair of complex zeros, while a and b define a pair of complex poles, and S is the input scaling factor.

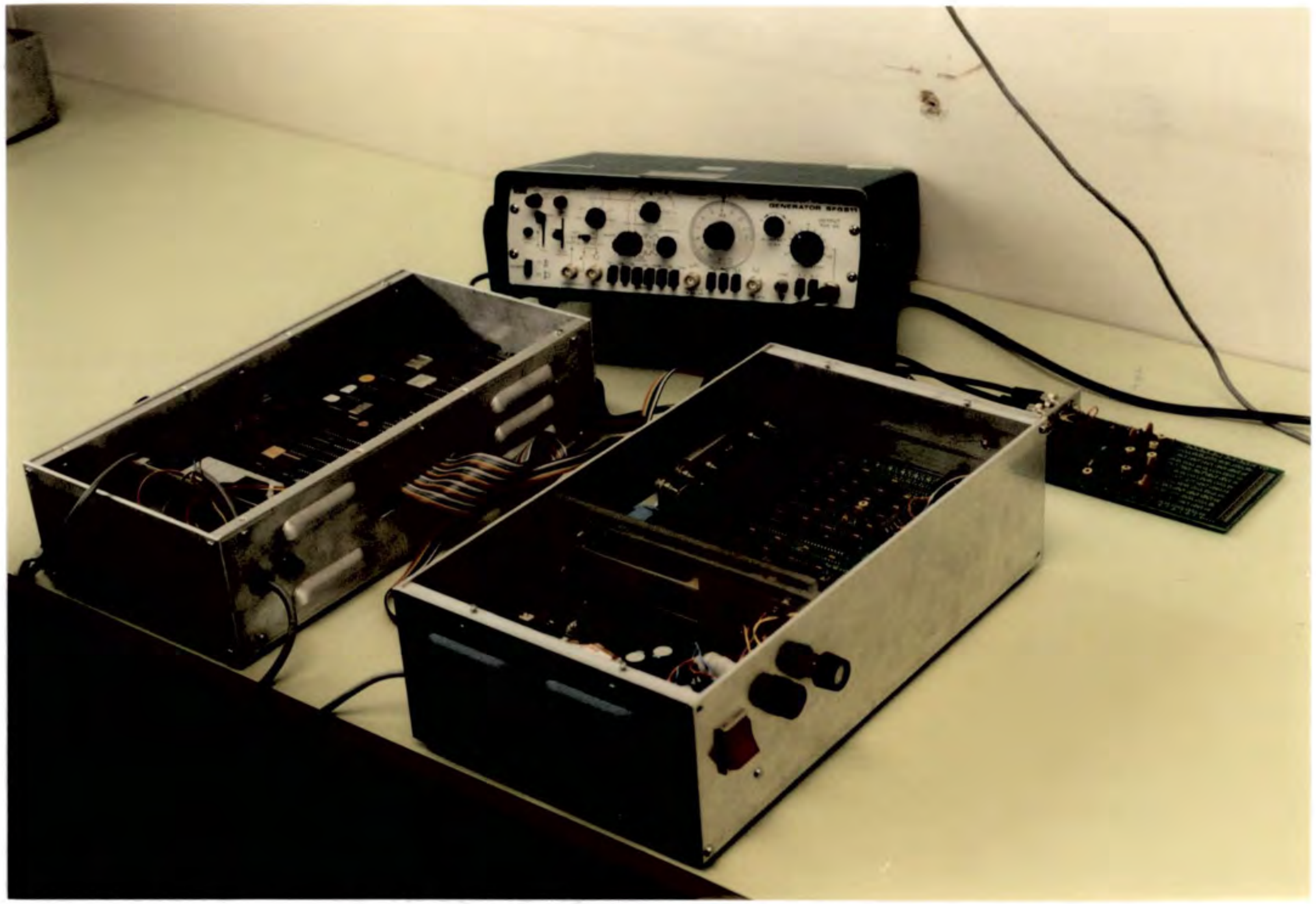


Fig. 5.2 The ANC system interfaced with 68000 microcomputer board

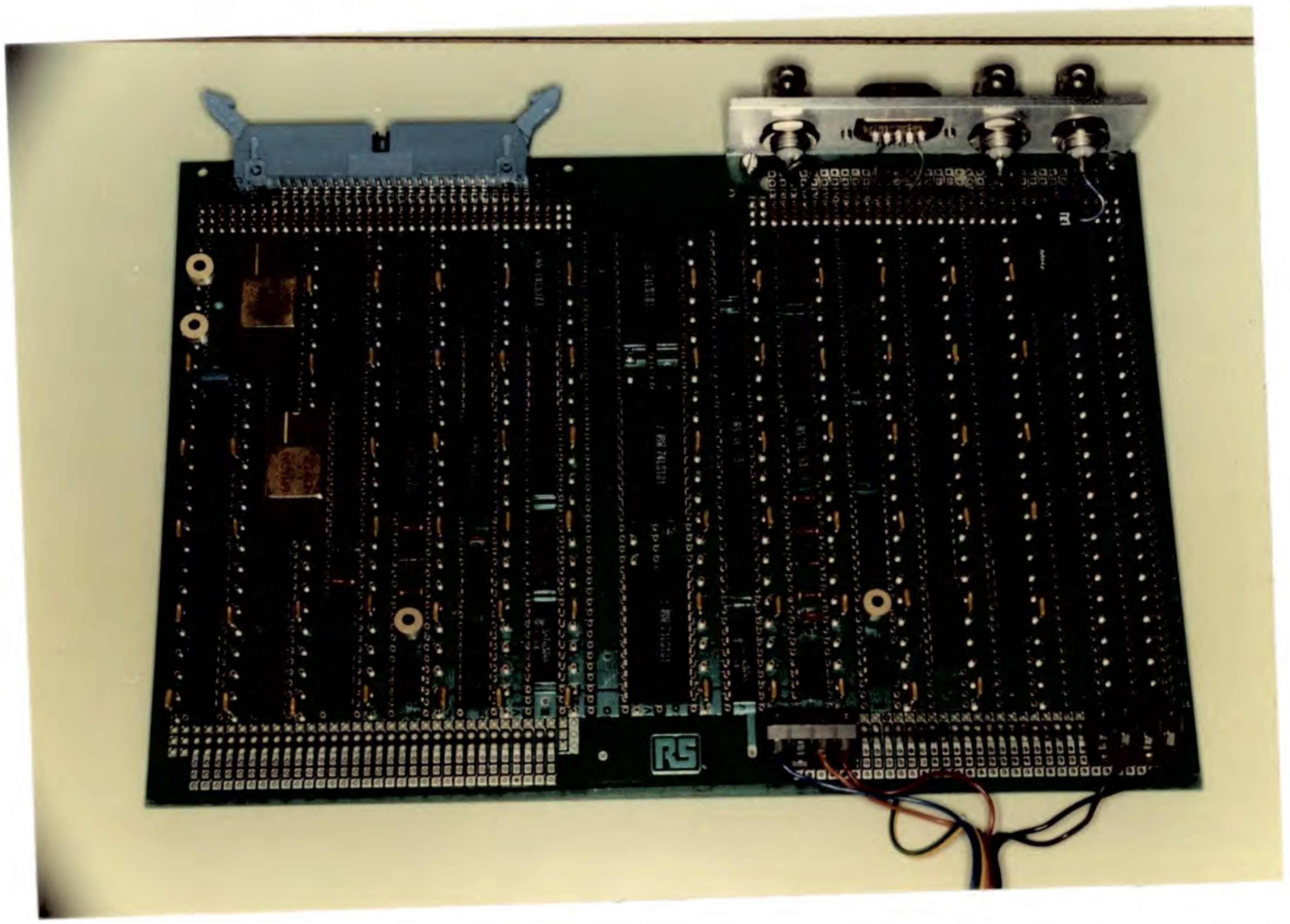


Fig. 5.3.a The ANC system (top view)

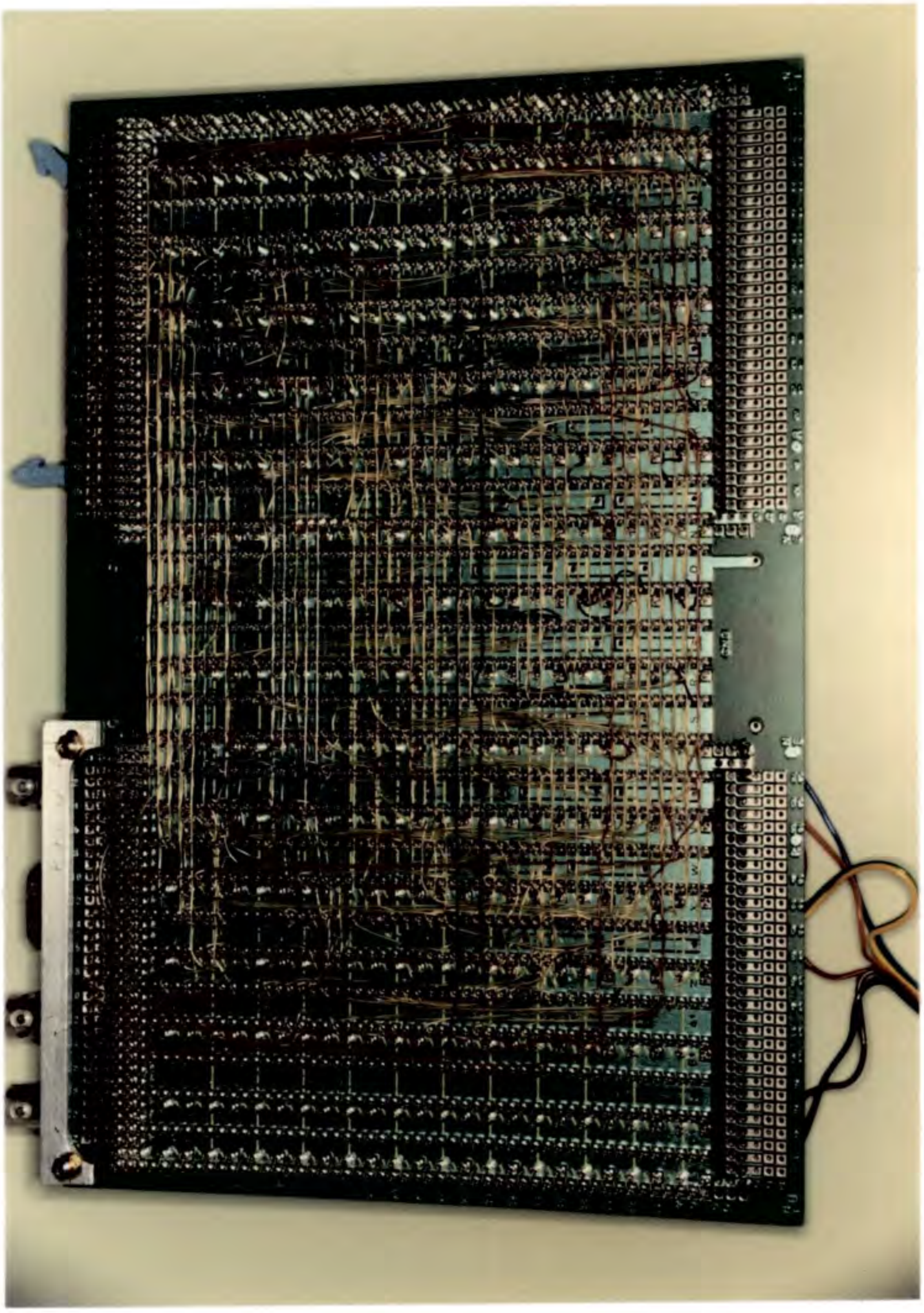


Fig. 5.3.b The ANC system (bottom view)

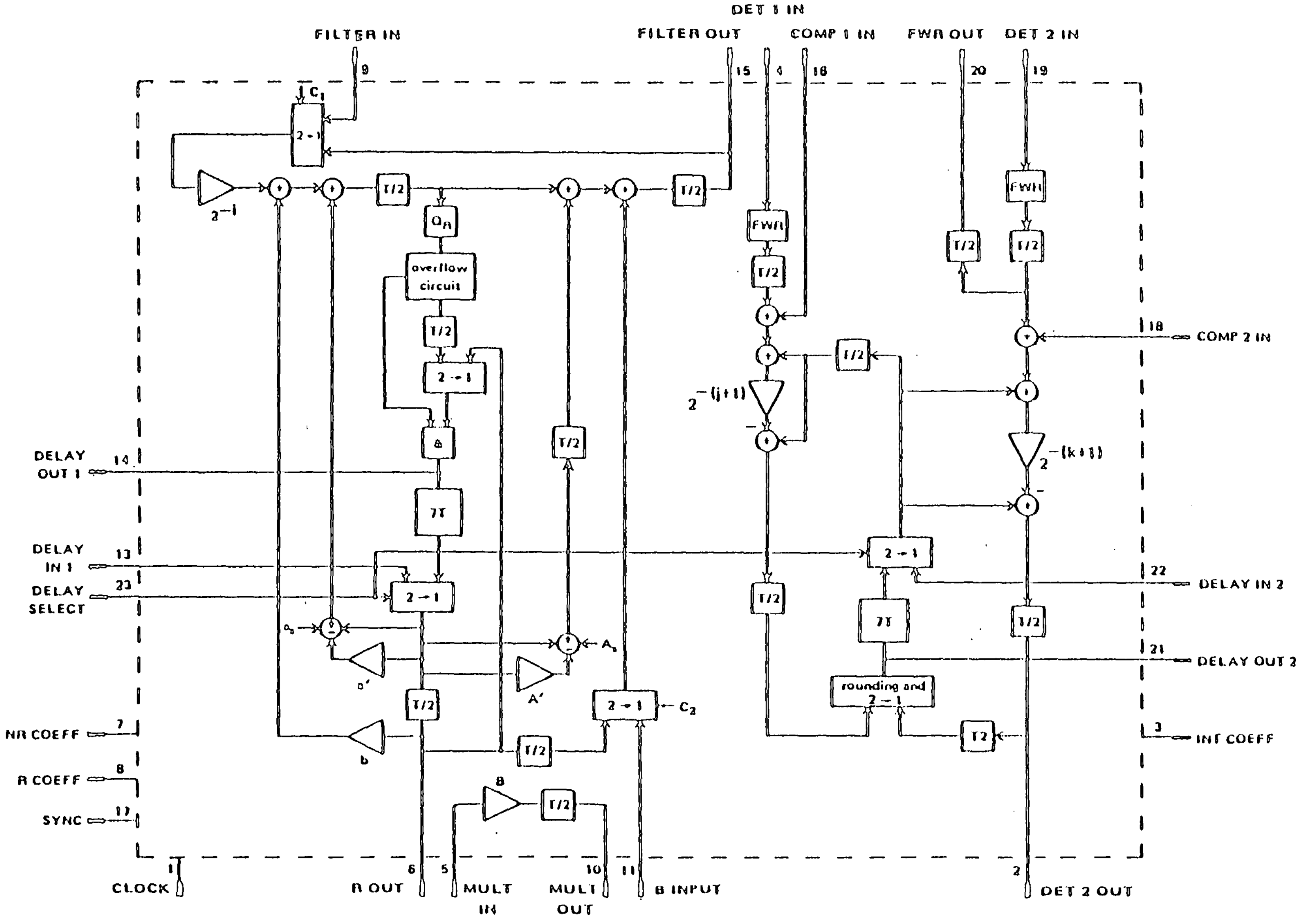


Fig 5·4 Complete functional diagram of the FAD IC

The components of the second order digital filter are illustrated in fig. 5.4 and the definition of some useful terms are briefly described below.

5.2.1 The FAD IC Components

5.2.1.1 Coefficient Multipliers

The multipliers form the heart of the filter and govern the stability and accuracy of the FAD chip. They have a serial/parallel structure and provide a 12-bit accuracy for the fractional part of the coefficients. The range of coefficients are chosen so that the poles and zeros can be implemented inside or on the unit circle in the Z-plane and are as follows:

$$2 > A \geq -2$$

$$2 > a \geq -2$$

$$1 \geq B \geq -1$$

$$1 > b \geq -1$$

In principle, the coefficients A and a are separated into an integer and a fractional part of 13-bit (including the sign bit) accuracy,namely A' and a', where the multipliers handle the fractionional part of the coefficients only. For the A and a coefficients an extra bit is required, namely A_s and a_s respectively and are used to control an adder/subtractor following the multipliers. A and a can be separated into the form:

$$(-1)^{A_s} + A'$$

and $$(-1)^{a_s} + a'$$

respectively.

Multiplier coefficients enter the FAD chip in serial form to keep the pin count to a reasonable number and are converted to parallel form for the multiplier with the least significant bit (LSB) first. They are entered in one computation cycle (timeslot) -32 clock cycles- and used in the next one (41,89,90).

5.2.1.2 The Quantizer

As a result of multiplying a 16-bit data word with a 13-bit coefficient, a 29-bit output word is produced. A quantizer is then used to restrict the word length of the data, resulting from the multiplication process, to 16 bits. Quantization can be performed in various ways (e.g. truncation and rounding), as has been mentioned in section 2.13. The FAD uses rounding because the errors introduced then have zero mean and the smallest variance (thus minimizing the mean-square value of the quantization noise). Rounding is achieved by adding 2^{-16} to the data entering the quantizer and truncating the result to 16 bits before further operations can be performed (41,89).

5.2.1.3 The Input Scaler and Overflow Circuit

As a result of two's complement number additions, overflow may occur. In order to avoid it, the input data words should be scaled down or attenuated by a factor such that even the most unfavourable sequence of inputs cannot cause overflow at either the internal node or the output node of the second-order section. In the FAD this input scaler factor, S, performs multiplication of each input data word by $S=2^{-i}$ by delaying the input data by (13-i) bits where i is an integer in the range $13 \geq i \geq 0$ which is controlled by four bits S_4, S_3, S_2, S_1, where S_4 is the LSB, S_1 is the most significant bit (MSB), encoded in binary form. Within the overflow circuit there is a detection circuit which sets all multiplier inputs to zero for one complete sample period, T_s, whenever overflow is detected. This part of the circuit is not necessary for a correctly scaled filter.

5.2.1.4 Data Selectors (Multiplexers)

DS1 Selects the filter order, so that the data enters the second order section from the input pin or is recycled for higher order. The selection is determined by the data bit C1 which controls DS1, when C1 is 0 data is recycled.

DS2 Depends on the value of the B coefficient, when B is unity C2=0, otherwise C2=1.

DS3 Delay selector. Certain delay units, notably a delay of 8T (which will be defined in the next section) are provided.

For a second order section or 16th order section no extra delay is required. For a 16th order section an internal delay of 7T is provided on the chip by applying a logical 1 to the DELAY SELECT pin, which causes DS3 to transmit the output from the internal delay to the coefficient multipliers (41,89).

5.2.2 Timing

The various time parameters of the FAD are listed and defined below.

(a) Clock period t_c: is defined as the reciprocal of the clock frequency f_c. The range of values which the FAD is guaranteed to operate at is

$$500 \text{ KHz} \leq f_c \leq 3.000 \text{ MHz}$$

and the typical value is 2.048 MHz, and consequently, the corresponding range of t_c for this range of frequencies is

$$0.333\ \mu\text{sec} \leq t_c \leq 2\ \mu\text{sec}$$

and the typical value is 0.448 μsec.

(b) Computation cycle T (Timeslot). This is defined as the time required for the serial/parallel coefficient multipliers to operate on their input words and is 32 t_c. However input data is only applied at the first half of this period (16 t_c).

(c) Sampling Period T_S: this corresponds to the unit delay of the input and output samples and is represented by Z^{-1} in the block diagram of the filter. The value of T_S must be an integer number y of computation cycles, y represents the number of second order digital filters used.

For a 16th order filter

$$\begin{aligned} T_S &= y * T \\ &= 8 * 32\ t_c \\ &= 256\ t_c \end{aligned}$$

The other consideration to be taken into account is the synchronization pulse on the FAD IC, which should coincide with the first bit of the coefficient and signal word entering the FAD.

Fig. 5.5 illustrates the relative positions in time of the input sample data, coefficients and control data, and the output data. It is important to realize that the input samples enter the FAD during the first half of timeslot 1 with LSB first and in two's complement form. In principle, in a practical application the coefficients should be applied for all timeslots. The coefficients enter the FAD during one computation cycle and are used during the next one. The output data will be available after the filter operation is completed (i.e. after y computation cycles, which coincides with the first half of the next timeslot). The layout of the individual bits within the stream of the coefficient and

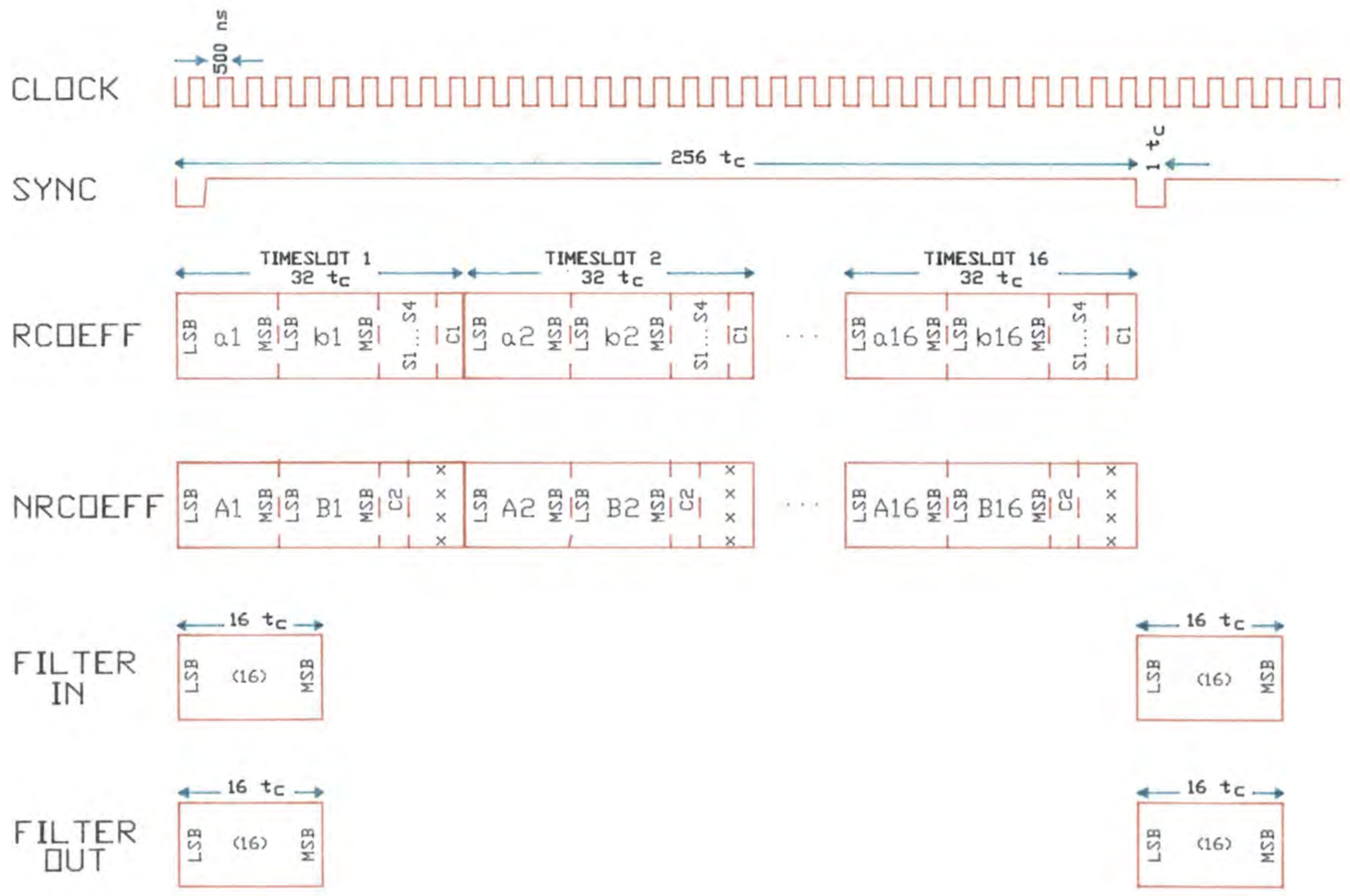

CLOCK
500 ns
SYNC
256 t_c
1 t_c
RCOEFF
TIMESLOT 1 32 t_c
TIMESLOT 2 32 t_c
TIMESLOT 16 32 t_c
LSB a1 MSB LSB b1 MSB S1 … S4 C1
LSB a2 MSB LSB b2 MSB S1 … S4 C1
LSB a16 MSB LSB b16 MSB S1 … S4 C1
NRCOEFF
LSB A1 MSB LSB B1 MSB C2 × × × ×
LSB A2 MSB LSB B2 MSB C2 × × × ×
LSB A16 MSB LSB B16 MSB C2 × × × ×
FILTER IN
FILTER OUT
16 t_c
LSB (16) MSB

Fig. 5.5 Timing Diagram of the FAD IC × denotes 'don't care'

control data is also indicated in the timing diagram. Table 5.1 gives the coefficient and control data bit definitions and clock pulse positions corresponding to them, for further details about the FAD IC refer to (41,89).

5.3 68000 Microprocessor

5.3.1 Introduction

In the last thirty years, computer technology has progressed from colossal mainframe computers to the microprocessors.

A microprocessor is the central processing unit (CPU) of a microcomputer and consists of one or more LSI circuits designed to perform most of digital processing tasks by an appropriate choice of a set of instructions 'software' defined by the user. The microprocessor works as a sequential computational or control unit by executing these sets of instructions.

The early 1970's marked the beginning of a revolution in the world of electronics: the microprocessor was realized and more powerful 8, 16, and 32-bit units were developed. In recent years, the MC68000 has emerged as one of the most significant products of a family of very-large-scale-integrated (VLSI) circuit microprocessors. It represents a generation of mature microprocessors because of its powerful facilities, its computational throughput, simplicity to program and ease of interfacing to other components in a microcomputer system.

5.3.2 MC68000 Architicture

The MC68000 is a 16-bit microprocessor designed for high speed processing applications. It runs with a TTL compatible external clock generator. The current versions of the 68000 have maximum

TABLE 5.1

Filter Section: Coefficient and Control Data Definitions

Clock Pulses	Bits	Definition
R Coefficient Input Data		
1 to 12	a_{12} to a_1	The numerical bits of a', i.e. the bits of the fractional part of the a coefficient, in two's complement form; a_{12} is the least significant bit (LSB).*
13	a_0	The sign bit of a': $a_0 = 0$ for $2>a\geqslant 1$ or $0>a\geqslant -1$; $a_0 = 1$ for $1>a\geqslant 0$ or $-1>a\geqslant -2$.
14	a_s	The add/subtract control bit: $a_s = 0$ for $2>a\geqslant 0$; $a_s = 1$ for $0>a\geqslant -2$.
15 to 26	b_{12} to b_1	The numerical bits of the b coefficient, in two's complement form; b_{12} is the LSB.
27	b_0	The sign bit of the b coefficient: $b_0 = 0$ for $1>b\geqslant 0$; $b_0 = 1$ for $0>b\geqslant -1$.
28 to 31	s_4 to s_1	The input scaling coefficient; s_4 is the LSB.
32	C_1	Input selector control: if $C_1 = 1$, FILTER IN is selected; if $C_1 = 0$ then FILTER OUT is selected.
NR Coefficient Input Data		
1 to 12	A_{12} to A_1	The numerical bits of A'; i.e. the bits of the fractional part of the A coefficient, in two's complement form; A_{12} is the LSB.*
13	A_0	The sign bit of A': $A_0 = 0$ for $2>A\geqslant 1$ or $0>A\geqslant -1$; $A_0 = 1$ for $1>A\geqslant 0$ or $-1>A\geqslant -2$.
14	A_s	The add/subtract control bit: $A_s = 0$ for $2>A\geqslant 0$; $A_s = 1$ for $0>A\geqslant -2$.
15 to 26	B_{12} to B_1	The numerical bits of the B coefficient, in two's complement form; B_{12} is the LSB.
27	B_0	The sign bit of the B coefficient: $B_0 = 0$ for $1>B\geqslant 0$; $B_0 = 1$ for $0>B\geqslant -1$.
28	C_2	B input selector: if $C_2 = 1$, B INPUT is selected; if $C_2 = 0$, B is unity.

*The 14 a (or A) data bits can be determined by expressing $a/2$ (or $A/2$) in two's complement form, and inverting the second bit.

clock rates between 4MHz and 16 MHz. It is assembled in a 64 pin package. The MC68000 contains 24 address connections permitting 2^{23} 16-bit words to be uniquely addressed. The external data bus is 16-bits wide and transfers data between the CPU and its memory and peripherals (interfacing). It is a bidirectional bus controlled by the Read/Write ($R/\overline{W}$) control signal which selects the direction of data flow on the data bus. The data bus acts as an input during a CPU read cycle and as an output during a CPU write cycle. Basic read or write access requires four clock cycles, so one byte of data can be transfered every 500 nsec for an 8 MHz MC68000. The MC68000 has a register oriented architecture containing eight 32-bit data registers (D0-D7), seven address registers, a 32-bit program counter and a 16-bit status register. The MC68000 is also capable of stack operations, two 32-bit stack pointer registers are available on the chip (91,92,93,94).

(1) Data Registers

Each data register is 32 bits wide. Byte operands occupy the low order 8 bits, word operands the low order 16 bits, and long word operands the entire 32 bits.

All of the data registers are general purpose accumulators and can be used as index registers or counters (93).

(2) Address Registers

There are seven general purpose address registers (A0-A6), these registers do not support byte sized operands but can be used for 16-bit or 32-bit long words only. The address registers are used to handle addresses for indexed memory addressing.

(3) MC68000 Memory

The memory of the MC68000 is organized, the same as the registers, into bytes, words and long words. Each byte has an address consisting of 24 bits. Byte address as may have any value while the word and long word addresses must be even numbers (93).

5.4 FRC 68000 PROFI KIT

The FRC 68000 PROFI KIT board is designed as a 16-bit/32-bit single board microcomputer as illustrated in the photograph in fig. 5.6. It has a 16-bit data bus and a 24-bit address bus. The address bus provides a memory addressing range of greater than 16 Megabytes. A functional diagram of the system is shown in fig. 5.7 and the system features are as follows:

(I) Central Processing Unit CPU

The 68000 microprocessor chip is the CPU of the FRC 68000 PROFI KIT and interfaces with the rest of the components on the board. It has a clock rate of 8 MHz.

(II) Memory and Address Decode Logic

The FRC KIT has two types of memory; random access memory (or read write memory (RAM)) and read only memory (ROM).

(a) RAM

The on board RAM consists of either 16KW or 64KW. It is used for the temporary storage of user program and data. The first 64 KW are located from hex address $000000 to $01FFFF, where $ denotes the hex number, as illustrated in table 5.2.

(b) ROM

The FRC 68000 PROFI KIT monitor (FORCEMON) is contained in a

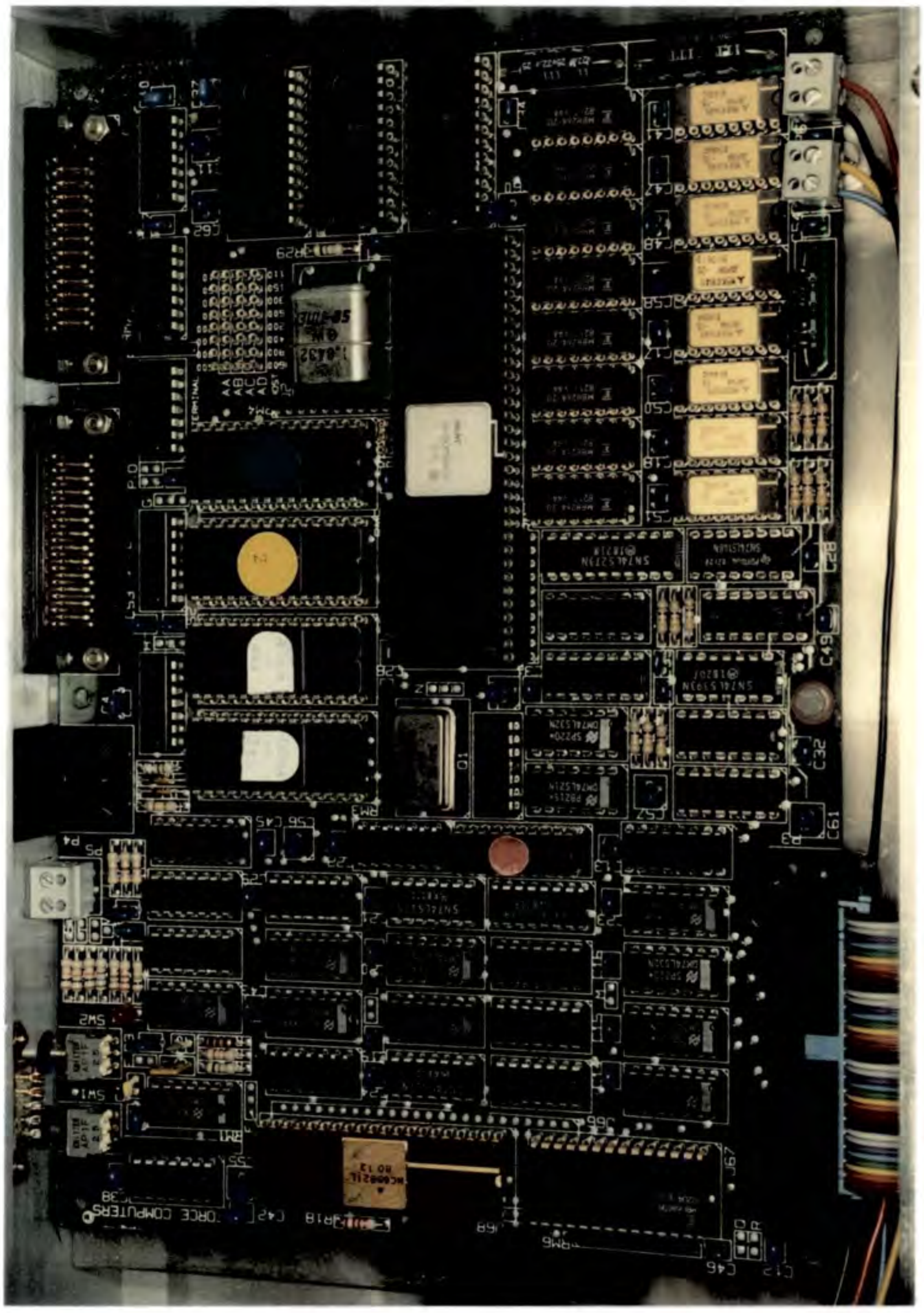


Fig. 5.4 FRC 68000 PROFI KIT board

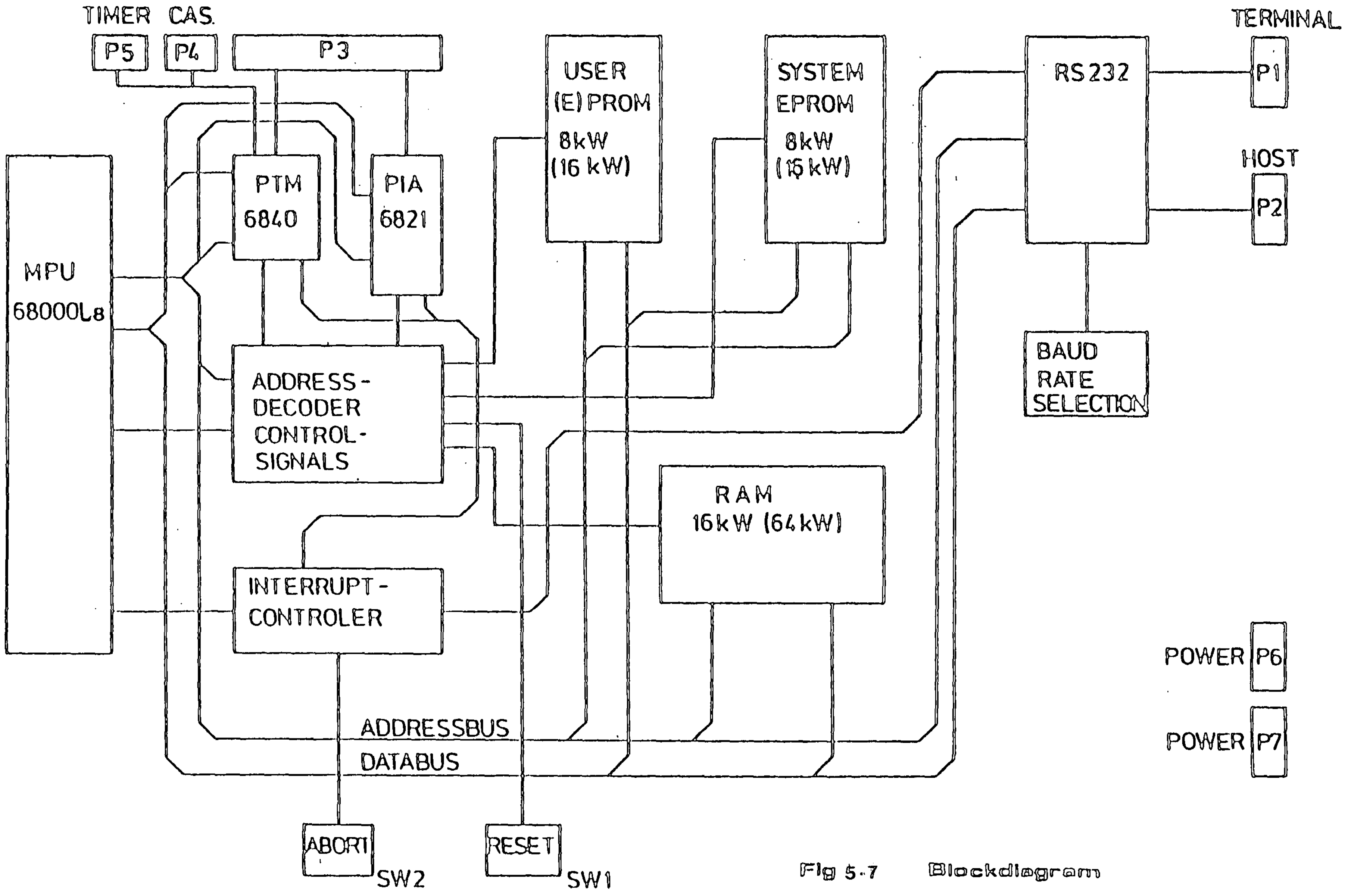


Fig 5.7 Blockdiagram

Memory-Map: 64kW RAM Version

```
------------------------------------------------------
  FFFFFF
     :                  unused
     :
  05FF36
------------------------------------------------------
  05FF35
     :                  P I / T
  05FF01
------------------------------------------------------
  05FF00
     :                  unused
     :
  05CEF8
------------------------------------------------------
  05CEF7
     :                  P I A
  05CEF1
------------------------------------------------------
  05CEF0
     :                  unused
     :
  050044
------------------------------------------------------
  050043
     &                  A C I A   ( HOST )
  050041
------------------------------------------------------
  050042
     &                  A C I A   ( TERMINAL )
  050040
------------------------------------------------------
  05003F
     :                  unused
     :
  04CF50
------------------------------------------------------
  04CF4F
     :                  P T M
  04CF41
------------------------------------------------------
  04CF40
     :                  unused
     :
  030000
------------------------------------------------------
  02FFFF
     :                  USER-(E)PROM
  028000
------------------------------------------------------
```

027FFF : 020000	SYSTEM-EPROM
01FFFF : 000800	USER-RAM
0007FF : 000400	WORK AREA FOR FORCEMON
0003FF : 000008	EXCEPTION-VECTORS
000007 : 000000	INITIALIZATION MPU

Table 5.2

16-KB of system ROM/EPROM (eraseable programmable ROM) expandable to 32KB. ROM can be addressed from $20000 to $27FFF.

(c) User EPROM/ROM Area

The 16kB user ROM/EPROM can be addressed as 2K*16, 4K*16 or 8K*16, and may be expanded to 32KB. That area allows one to integrate user EPROM/ROM into the PROFI KIT environment (95).

(III) I/O

Combination of some of these decoded outputs with some of the address lines or with some of other decode lines generate the peripheral I/O address range. The decoded address lines are also capable of decoding RAM and ROM. Each one of these ports has a different function. Ports 1,2,3 are of particular interest.

(1) Port 1 and 2

These are serial communication ports, port 1 is used for a terminal and port 2 is used to save programs. Both are RS232-compatible and have different selectable baud rates.

(2) Port 3

Port 3 is connected to a 6821 parallel interface adapter (PIA) which contains two 8-bit ports (A&B).

5.5 Practical Implementation Problems

5.5.1 Coefficients Timing

The FAD IC is required in some way to store the coefficients. One way to do this would be to use semiconductor memories. For fixed values coefficients ROM could be used. In this project, the coefficients are variable (time-variant) and temporary storage, namely RAM is used.

The problem of feeding the coefficients into the FAD chip from the memory is of major concern in the design and implementation of the ANC, because the time required for achieving this feeding process becomes the limiting factor and leads to a more complex implementation.

As has been pointed out previously, these coefficients are not fixed and are updated according to the eqns. 3.5.1 and 3.5.2 by use of the LMS algorithm. In order to achieve this, a significant amount of hardware logic is required which makes the system complex.

An alternative would be to develop a 68000 assembly language program to execute the updating operation and to write the data into RAM.

As has been mentioned before, the FAD IC requires one bit of recursive and non-recursive coefficient every clock cycle (in serial form) with the LSB first. In the case of the 16th order filter 2*256 clock cycles are needed for feeding all the coefficients bits (2*256 bits) into the FAD IC. In other words, the memory must be capable of writing one new bit of each coefficient every clock cycle into the FAD IC.

This must be achieved in the minimum time which inevitably requires high speed digital circuitry. However, since the required speed cannot be obtained by interfacing the MC68000 to a simple memory circuit, it is achieved by interfacing the MC68000 microcomputer to the circuit as shown in fig.5.8. Writing one bit of each coefficient in this period of time (one clock cycle) exceeds the capability of the MC68000, so the MC68000 is interfaced to two sets of high speed RAMs, namely RAM1 and RAM2, with multiplexed address lines between the MC68000 and a counter, in conjunction with other auxiliary components. Each set of RAMs consists of two 74S200 TTL devices (96) (tristate) with a maximum access time of 50 nsec, organized as 256 words by one bit per word with separate input-output pins and with eight address lines as illustrated in fig.5.8.

In principle, the writing function is generally separated into two phases:

(1) writing the data into RAM1 with the aid of the MC68000.

(2) transfering the data from RAM1 to RAM2.

5.5.1.1 Writing Data into RAM1

A software routine is executed to write one bit of each of the coefficients in serial form into RAM1, this will be described in detail in the next chapter. Eight address lines are connected to port A of the PIA to identify 256 locations. The MC68000 addresses the desired word of RAM1, beginning at word 0 and going sequentially through to the last word.

To synchronise the MC68000 and RAM1 so that write control inputs are activated simultaneously, the $R/\overline{W}$ control pulse of the MC68000 is connected to the $\overline{WE}$ control input of RAM1 during the write operation.

When the write operation is performed by the MC68000, the information is serially transfered from the output port of the PIA to the RAM1's locations that have already been addressed by the MC68000. The data is written into RAM1 whenever the $\overline{WE}$ pulse goes LOW. In regard to the fact that the coefficients bits should be

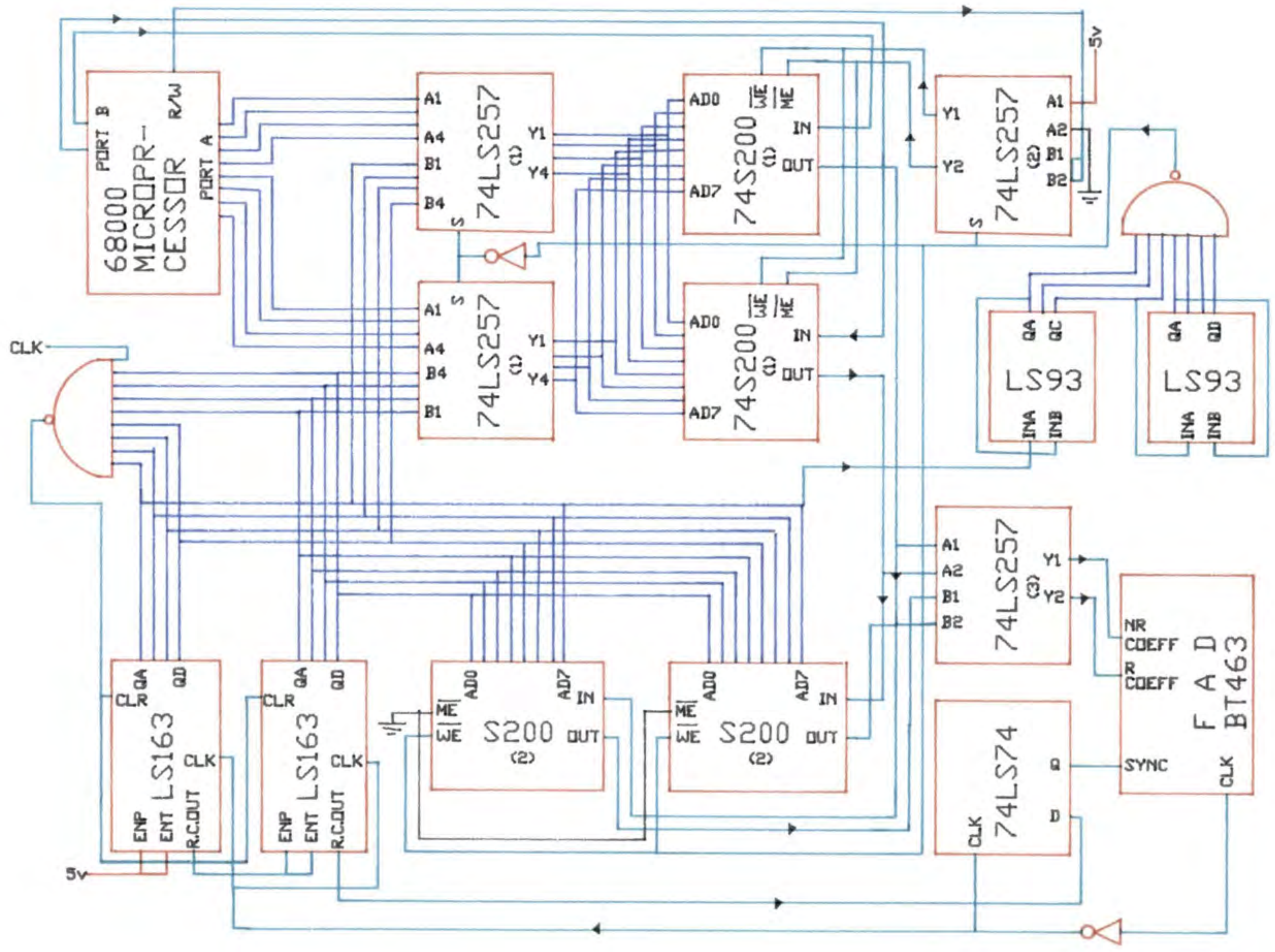


Fig.5·8 RAMs Interfacing Circuit Diagram available, one of each every clock cycle and this speed is not sufficient for achieving this function. Therefore RAM1 is disabled whenever R/$\overline{W}$ pulse goes HIGH. This could be done by connecting the MEMORY ENABLE control inpts ($\overline{ME}$1,2,3) to R/$\overline{W}$ pulse as well, as illustrated in the timing diagram in fig. 5.9. In that case the data are stored in RAM1 and cannot be fetched by the rest of the circuit.

One important consideration should be borne in mind, that during write process the FAD IC requires its coefficients to be fed into it. In order to accomplish this operation the control inputs of RAM2 have to satisfy the following conditions:

(a) The RAM2 address lines are connected directly to the outputs of a counter as shown in fig. 5.8, operating at the clock frequency of the FAD IC. So the address is incremented by one every clock cycle.

Two 74LS163 synchronous 4-bit counters (96,97), cascaded together by connecting the ripple carry output (R.C. OUT) of the first one with count enable inputs (ENP,ENT) of the second one, are used to generate the addresses for RAM2. The first ENP & ENT are wired HIGH to enable counting. This counter is cleared and reinitialized starting with address 000 at each computation period (8 timeslots for a 16th order filter), this will be described in more detail in the next chapter.

(b) The $\overline{WE}$ is activated (HIGH) during a write operation by RAM1, so RAM2 is in the READ mode and the data are fetched from the addressed locations, which come from the previous write operation (old data) and are fed into the FAD IC every clock cycle.

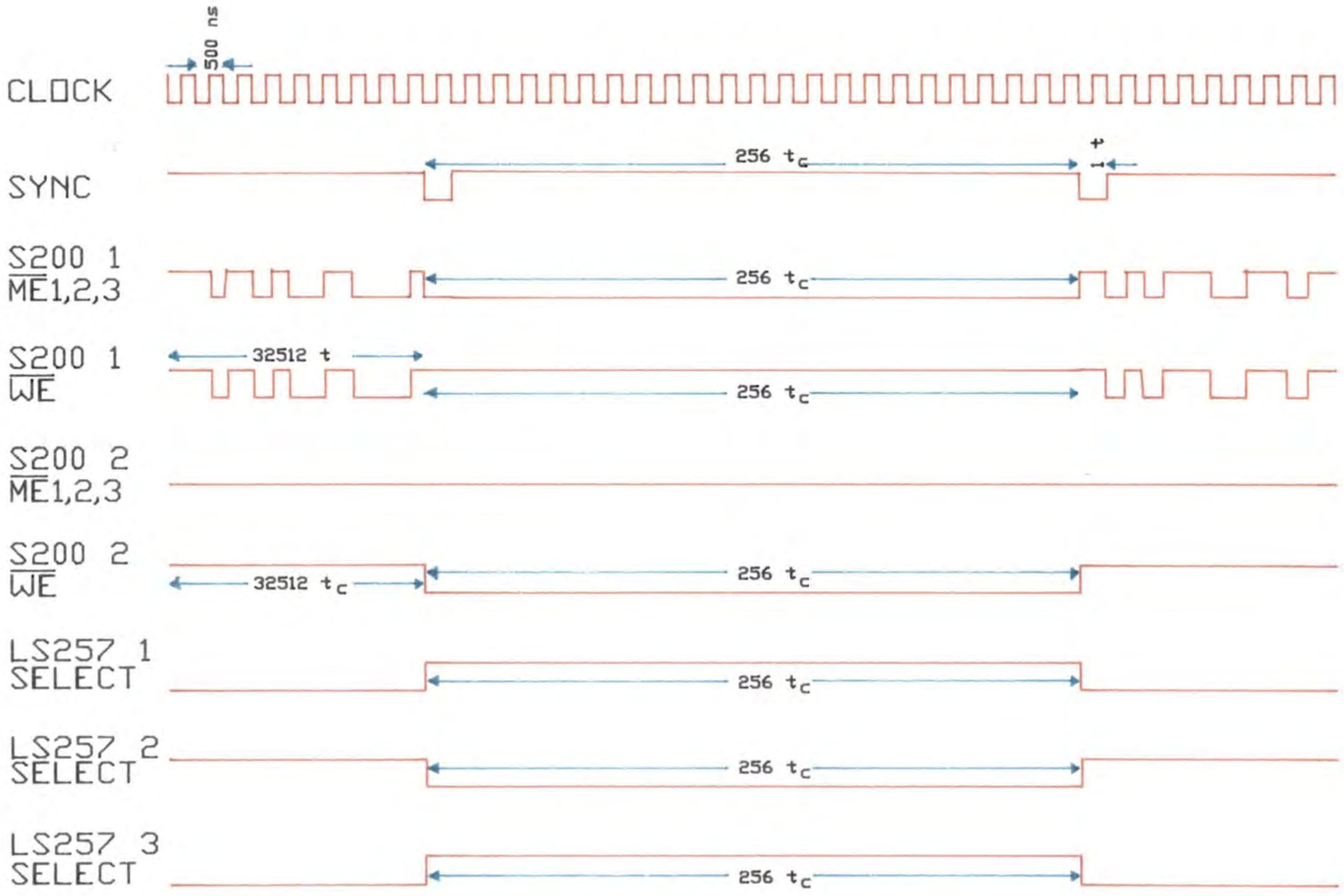


Fig. 5.9 RAMs Interfacing Timing Diagram

The write process progresses until all 256 bits are allocated in the appropriate addressed positions in RAM1 with the aid of the MC68000.

5.5.1.2 Transfering Data from RAM1 to RAM2

As mentioned previously implementing the writing process using only one RAM is impossible. A transfer process, from RAM1 to another RAM, namely RAM2, overcomes this drawback and feeds the data into the FAD IC within the time available within the adequate speed (time). RAM2 is identical to RAM1 in its performance.

To accomplish the interfacing process, so that one bit is read from RAM1 and written into the corresponding memory location in RAM2 (and simultaneously into the FAD every one clock cycle), the address lines of RAM1 are driven from the same counter that was used to address RAM2 as mentioned in the previous section.

During this process, the state of the control inputs of both RAMs are changed. For RAM1, the $\overline{WE}$ control input is held HIGH to activate the read operation, $\overline{ME}$1,2,3 are set to logic 0 enabling the memory, while the second RAM operates inversly ,i.e. in write mode instead of read mode, $\overline{WE}$ control input is held LOW, $\overline{ME}$1,2,3 remain as before, as illustrated in the timing diagram in fig 5.9.

During the transfer process no new or old data could be written from RAM2 to the FAD IC since the read mode from RAM2 is disabled. At this time the FAD IC is swapped from RAM2 to RAM1 in order to make the new coefficients on recursive and non-recursive inputs available every clock cycle, with the aid of a multiplexer.

5.5.2 Writing Data into the MC68000

Referring to eqns.3.5.1 and 3.5.2 in chapter 3, to update the coefficients with the aid of the MC68000, the input, desired response input and output samples must be available at its input port whenever the updating operation takes place. One difficult problem associated with using FRC PROFI KIT is the limited input/output ports (pins) available, which hinders direct connection between these data lines and the MC68000 and makes the hardware design more complex.

Unfortunately the number of input pins are limited to 6 while there are 36 bits (requiring 36 pins) of data required.

A particularly convenient method for overcoming this limitation involves using 3 sets of multiplexers, refer to appendix B, (96,97,98) with the aid of a software routine carried out by the MC68000, listed in appendix E.

5.6 Interfacing Considerations

5.6.1 Input Samples to the FAD

Most real world quantities are analogue and need some way of communicating with the digital systems or vice versa. In this project the ANC inputs (input signal and desired response input) are required to be converted to digital form. Conversely the ANC outputs (filter output and error output) should be converted from a digital form into an analogue form.

The ADC represents the connection between the analogue and digital systems and generates a digital estimate of the analogue signals.

One method of A/D conversion is successive approximation which is characterized by high speed processing, and uses a DAC to compare an analogue signal with an internally generated signal (99,100).

In this project an RS574 12-bit resolution successive approximation ADC (101) was used with a maximum conversion time of 32μsec for 12-bits, as shown in fig. C-1. It has been mentioned before that the FAD input is serial and since the binary numbers representing the ADC output are in parallel form. A 12-bit parallel to serial shift register (97) is employed in the hardware design.

5.6.2 FAD to Output Samples

To convert the digital output of the FAD IC to analogue form, a DAC is required. But since the FAD output is serial, a 12-bit serial-in parallel-out shift register, consisting of two 74LS164 8-bit parallel-out serial-in shift registers, is used as illustrated in figs. D-1 and D-2. The output lines of the shift register are latched by means of two SN74LS373 Octal D-Type Transparent Latches (96,102). This eliminates any possible race conditions that could be occur while new data is being loaded into the shift register. The output data is available on the FAD IC on the rising edge of the clock with LSB first.

An RS7545 12-bit DAC (103) is used to get the analogue output. It is a monolthic 12-bit CMOS multiplying DAC. Data are loaded using the CHIP SELECT ($\overline{\text{CS}}$) and WRITE ENABLE ($\overline{\text{WR}}$) control pulses which may be held low allowing direct unbuffered operation.

5.6.3 Producing the Error Output

An essential parameter of the ANC system is the error output signal e_j as has been discussed in chapter 3. One new sample of e_j must be obtained every computation period (8 timeslots). With regard to the speed of the MC68000 (section 5.5.1) and the impossibility of forcing the MC68000 to update the 32 coefficients (16 non-recursive and 16 recursive) of the filter in time. It is not prefered to interrupt the software program every computation period to compute the error output from the desired response input and the filter output. So it is difficult and awkward to obtain e_j in time with the aid of the MC68000. A solution to this is an Arithmetic Logic Unit (ALU) (97) which is incorporated into the hardware design of the ANC system. A set of ALU's formed from 3 74LS181 ALU/Function Generator are cascaded together by connecting the CARRY-OUT (Cout) of a stage to the CARRY-IN (Cin) of the succeeding stage as shown in fig.5.10.

The advantage of the 74LS181 (96) is that it can perform any of 16 binary operations and 16 logic functions without the use of external circuitry, just by changing the states of its control inputs. There are five control inputs (MODE and four SELECT inputs (S0,S1,S2,S3,) that determine the operation performed on the inputs.

In order to obtain e_j the filter output samples are subtracted from the latched desired response input samples. To accomplish the subtraction, the MODE control input (M) is set to logic 0 to define the arithmetic function. The SELECT control inputs are set to logical 6 to select the subtraction function. The CARRY-IN is set LOW. The output of the 74LS181 is encoded into four bits, so that

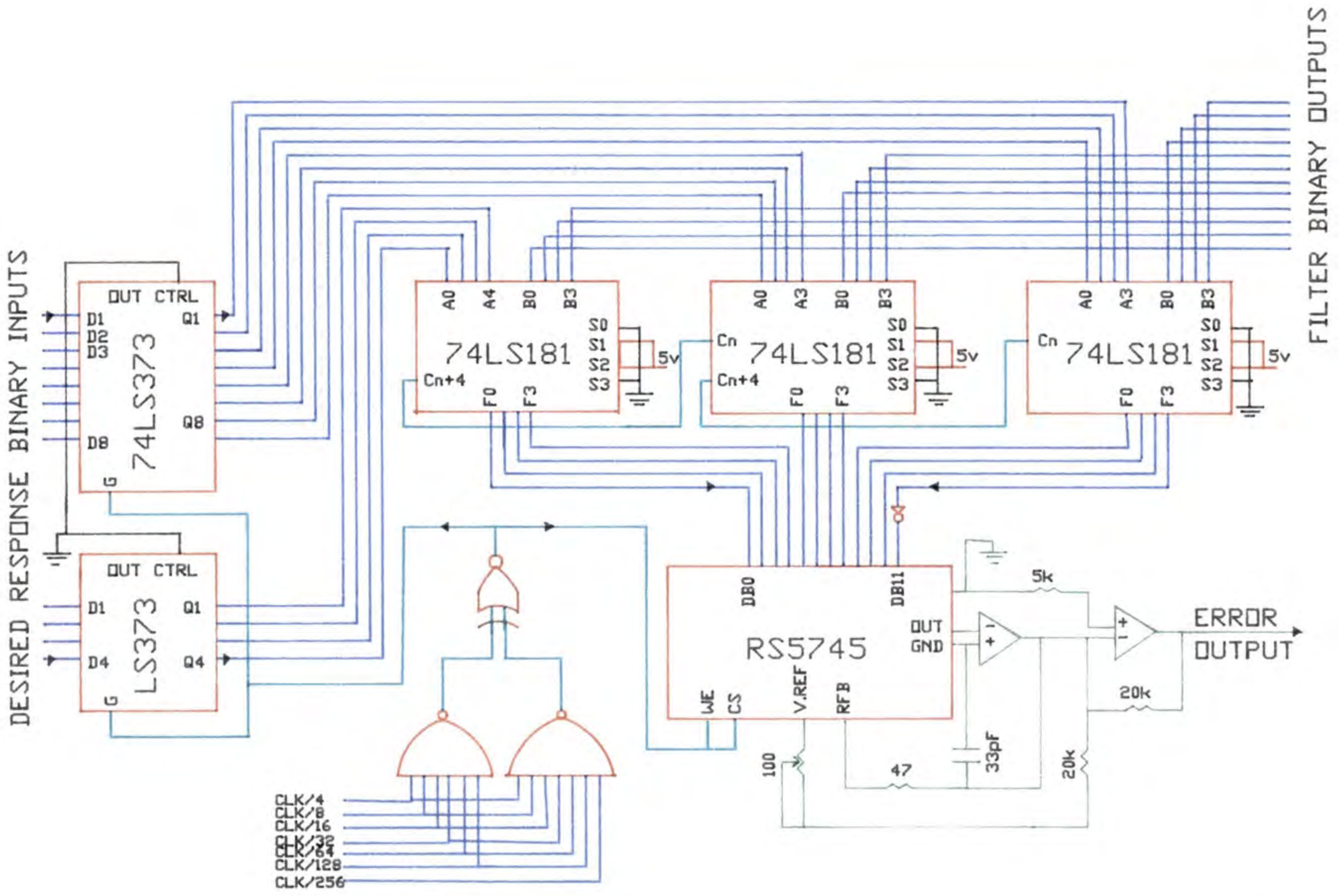


Fig. 5.10 Circuit Diagram for Producing the Analogue Error Output

e_j is represented in 12 bits. The hardware for converting e_j into its analogue represetation is similar to that used for the filter output. The 12-bit DAC is used as in the case of the filter output y_j as shown in fig.5.10.

5.7 Conclusion

This chapter is concerned with the description of the single board 68000 microcomputer and the digital filter IC (FAD) as the most two significant elements in the hardware design of the ANC. It is emphasised how successfully the limitation of the speed and interfacing connections of the 68000 microcomputer were overcome and how it is possible to write one bit of each coefficient every clock cycle, employing two sets of high speed RAMs, simultaneously. It has been shown that the system is capable of operating in real time, so one input sample fed to the FAD IC and a corresponding one output and error output sample are produced every computation period.

CHAPTER -6-

ADAPTIVE NOISE CANCELLER IMPLEMENTATION

6.1 Introduction

Many kinds of problems are encountered realizing a filter in hardware, for instance, design problems, where the theoretical basis for the desired system is translated into a hardware design; implementation problems, where the hardware design is translated into physical components, etc.. The previous chapter dealt with these kind of problems and ways of solving them. Another kind of problem is one associated with the components involved in the design. Unfortunately, in our case, the FAD IC, which is the heart of the adaptive noise cancelling system in the project failed to produce the expected result. The first section of this chapter is concerned with this problem and the test circuit employed for the purpose of detecting it. The rest of the chapter emphasizes how a 68000 software routine has been applied to update the coefficients as well as manipulating the sampled data required for this operation.

6.2 Solving the FAD IC Problem

It is well known that, in general, electronic components throughly tested before they are sold. A circuit of a real-time second order digital filter with variable coefficients, utilizing the FAD IC, was constructed as a first step in developing the design of the 16^{th} order LMS adaptive filter. Two streams of data (32bits each) comprising the coefficients and the control bits were entered (serially) into the NRCOEFF and RCOEFF inputs of the FAD

IC. One bit of each stream of these data were read by the FAD IC every clock cycle in order and at an appropriate time with respect to the SYNC pulse as illustrated in the photograph in fig.6.1.a. These data were written and stored in two sets of RAMs with the aid of a 68000 software routine as mentioned before and described in detail later in this chapter. The filter input was a sinusoid below the sampling frequency (for example below 32KHz at clock frequency 2MHz) converted to its digital form by means of a 12-bit ADC. One input sample was applied to the filter input every computation period (32 t_c). Consequently, one output sample was available every computation period, and it was converted into analogue form with the aid of a 12-bit DAC.

After the construction of the FAD circuit was completed, extensive continuity checkes verified that the circuit was correct, as far as could be determined. Unfortunately, functional tests of the system showed that the hardware was not performing as expected, as shown in fig. 6.1.b. Manipulation of the FAD clock frequency and the values of the coefficients was also carried out in order to see if some improvement in the filter behaviour resulted, but the output remained incorrect.

As a first stage in solving this problem the ADC, DAC and associated shift registers were omitted from the design to avoid any quantization noise which could degrade the filter performance. In this case the filter input was connected to ground. The pulse transfer function of such a filter, is expressed as

$$G(Z) = \frac{1+AZ^{-1}+BZ^{-2}}{1-aZ^{-1}-bZ^{-2}}$$

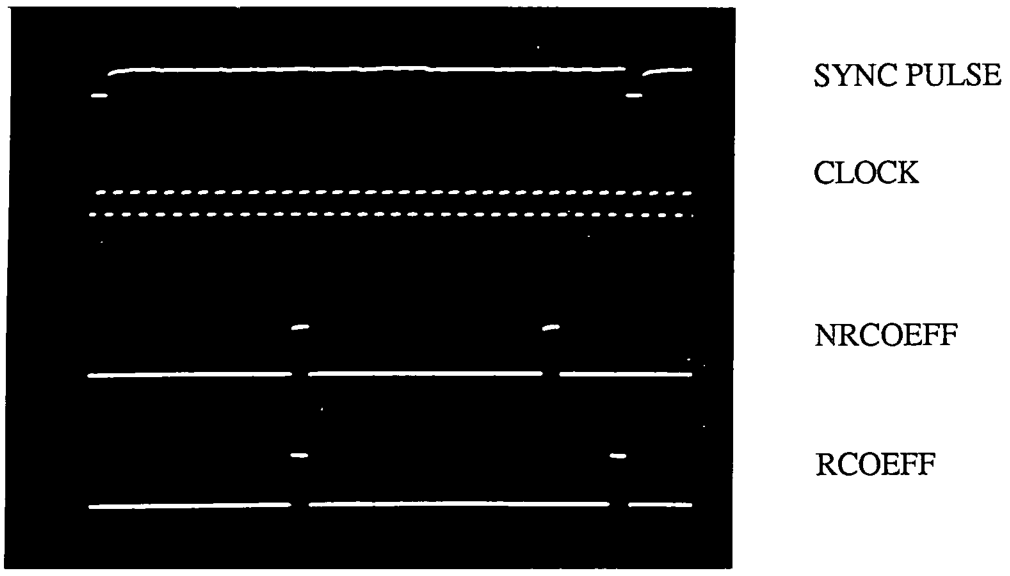


Fig. 6.1.a Coefficients timing configuration for second order recursive digital filter at clock frequency of 1 MHz

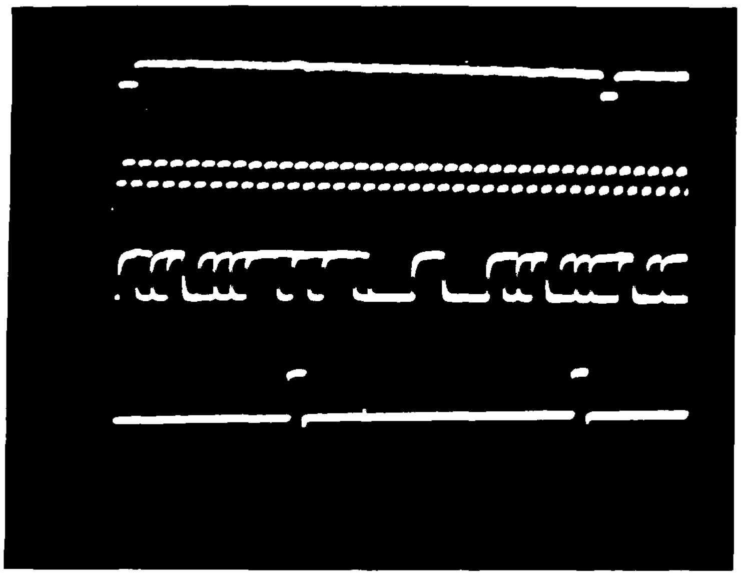


Fig. 6.1.b Second order recursive digital filter output of the faulty chip at clock frequency of 2.051 MHz with respect to the clock and SYNC pulse.

in the frequency domain and represented by the linear difference equation

$$y(n)T = x(n)T + Ax(n-1)T + Bx(n-2)T$$
$$+ ay(n-1)T + by(n-2)T$$

in the time domain. It is therefore obvious that $y(n)T = 0$ whenever the input samples are set to 0. Unfortunately the limit cycle oscillations occured at the filter output even with the filter input set to zero (with non-zero coefficients), and therefore all coefficients were set to zero. In spite of all these modifications the filter was still unable to produce a zero output.

We realized that we were facing a significant problem, which could be either due to the FAD IC or the coefficients and the control bits timing. So the second stage was to contact Plessey (the manufacturer of the FAD IC) and ask them for further information and discuss the problem with them. Unfortunately no further information had been published.

Assuming that the problem might arise from the coefficients timing, a simple second order digital filter, employing the FAD IC, was constructed (as described in the data sheet supplied with the chip). The circuit diagram of such a filter is depicted in fig. 6.2.

Only five ICs were required, the FAD IC, a 2*32 PROM, 2 LS163 forming the counter and a TTL quadruple 2-input positive OR gate (92) to generate the SYNC pulse. Zero external delay was required, so DELAY IN 1 was connected to DELAY OUT 1, and the inherent delay of 1T was selected by connecting the DELAY SELECT control input to 0. Hence the case of non-unity B coefficient was used, the B

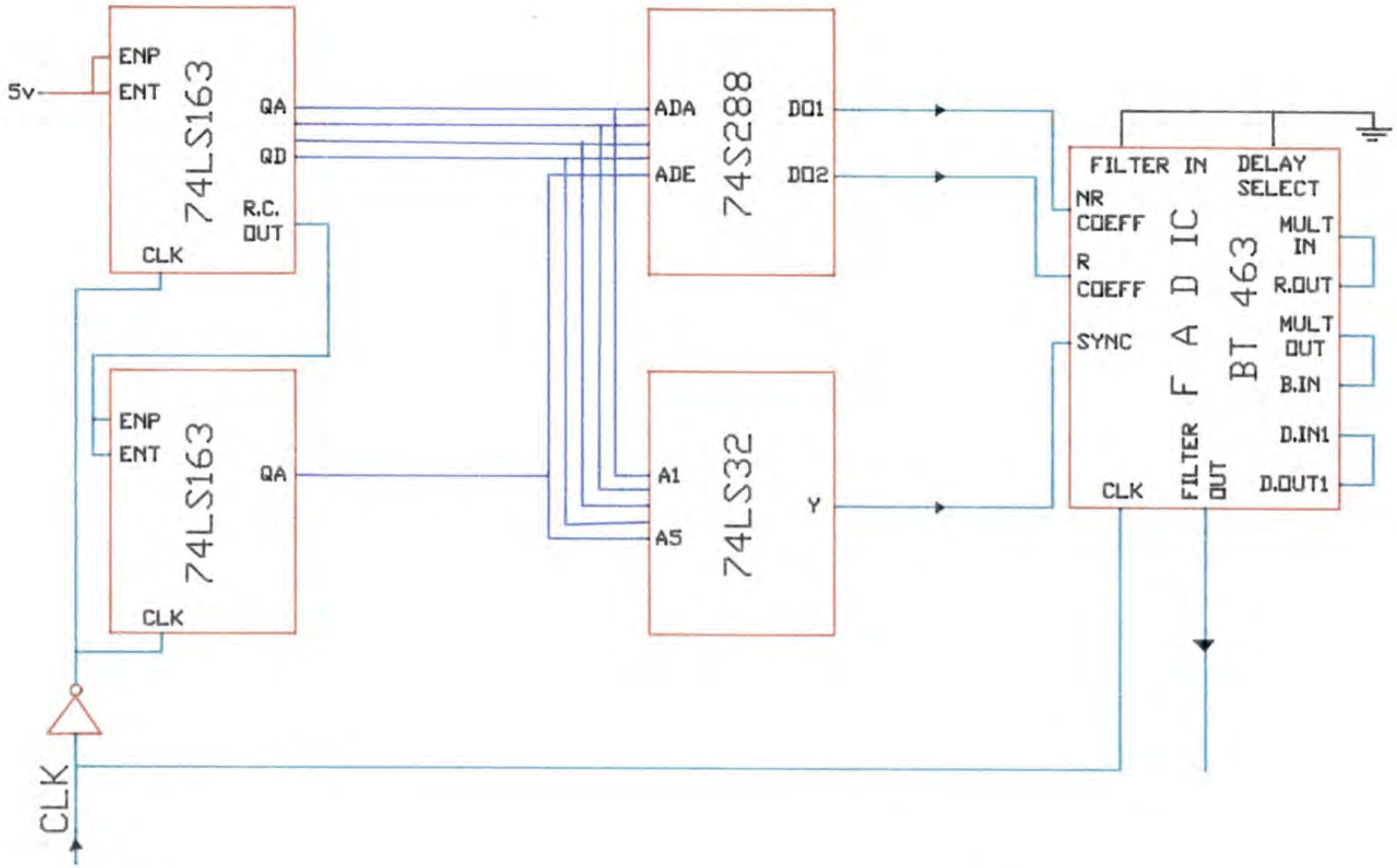


Fig. 6.2 Second Order Recursive Digital Filter

multiplier was employed by connecting R OUT to MULT IN and MULT OUT to B INPUT and C_2 was set to 1. The control bit $C_1=1$ so that the data applied to the FILTER IN is input to the filter section. The input scaling factor had no effect on the result in the case of zero input, so it could be set to any arbitrary value.

Coefficients and control data were applied one bit each clock cycle as before and they are tabulated in table 6.1 according to table 5.1.

More tests were carried out, but the problem still persisted (fig6.1.b).

In light of previous attempts, a fault in the FAD IC itself was suspected. It is well known that using CMOS devices is somewhat risky because of their sensitivity to electrostatic charges which can cause damage to the chip. However, a couple of new chips were ordered, and were used in the design, but unfortunately no further progress was achieved with the filter.

The next stage was to contact Plessey again and they confirmed that the chip was in good condition, so we contact the people who contributed the design of the FAD chip. Dr. Patrick Hughes, one of the research staff concerned with the chip design at British Telecom Research Laboratories kindly helped and examined the chip, but no fault could be detected in it with their test machine. However, a number of photographs for the timing of the coefficients with respect to the clock pulse and SYNC pulse control input and for the output were sent to him.

Finally the situation became clearer and Dr. Hughes detected

Table 6.1

Coeffecient and Control Data

Coeff	Decimal value	2's Complement Binary Form 12 (LSB) 1 (MSB)	Sign bit
a	0	0 0 0 0 0 0 0 0 0 0 0 0	$a_0 = 1$
a_s		0	
b	0	0 0 0 0 0 0 0 0 0 0 0 0	$b_0 = 0$
S	xxxx	xxxx	
C_1	1		
A	0	0 0 0 0 0 0 0 0 0 0 0 0	$A_0 = 1$
a_s			
B	0	0 0 0 0 0 0 0 0 0 0 0 0	$B_0 = 0$
C_2	–	1	

x denotes 'don't care'

an error in the multipliers, which represent the heart of the chip, and kindly sent us a couple of new versions that operated correctly. It appeared that, unknown to either ourselves, British Telecom or Plessey we had been supplied with a very early version of the FAD IC which, while fully functional according to their tester, suffered from errors in the multiplier section which caused the observed faults.

The new chip was then used and the output observed under the same conditions and was found to be operating correctly. Another test was carried out on the new FAD IC by setting all non-recursive coefficients of the 16th order filter to the same fixed value, and the recursive coefficients to zero. The frequency response of such a filter is illustrated in fig. 6.3.

6.3 The FAD IC as a 16th Order Filter

The FAD IC is designed to operate from a single +5v power supply and has a single TTL compatible clock input with 50% duty cycle. For the 16th order filter, a delay of 7T (224 clock cycles) is provided on chip, as well as the inherent delay T, so that the delay time necessary for the implementation is 8T. To select the 7T internal delay the DELAY SELECT input is wired high. It has been mentioned before that the coefficients are chosen to be less than unity to gain a better stability. To select the case of non-unity B coefficient the same procedures as in the second order filter are followed.

6.4 The Input & Output Sampled Data

It has been discussed earlier in chapter 3 that the adaptive noise canceller has two inputs, the filter input and the desired

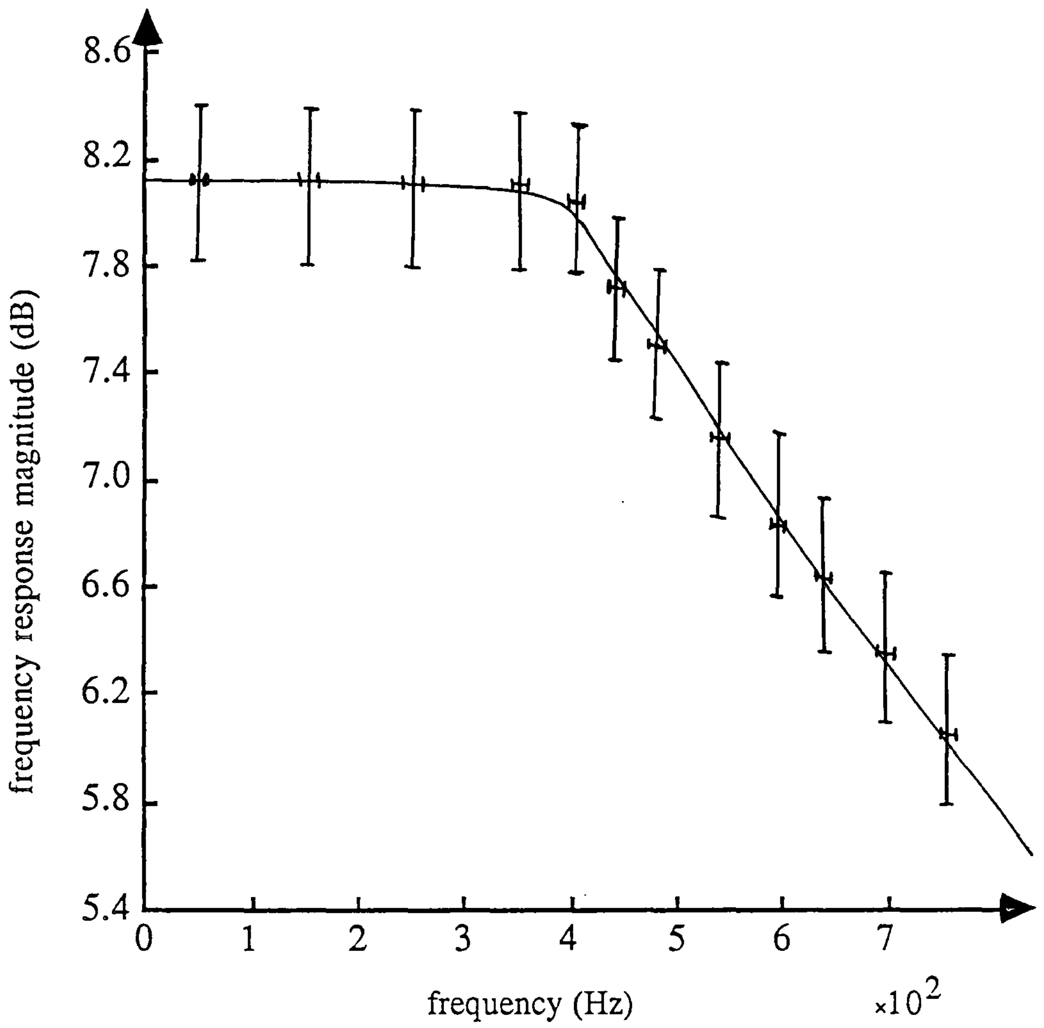


Fig. 6.3 Amplitude/Frequency Response of the FAD IC

response input and two outputs, the filter output and the error output. These sampled data are essential components in the updating operation of the filter coefficients which is achieved with the aid of the MC68000. These sampled data communicate with the processor through the PIA.

This section is concerned with describing the method of expressing the input sampled data in two's (2's) complement binary form, loading the data into the MC68000 and saving the input& output sampled data in the MC68000.

6.4.1 Expressing The Input Samples in Two's Complement

Any number expressed in binary form must be represented in 1's and 0's. Positive numbers cause no problem, but a decision must be made as to how to represent negative binary numbers. The most common method is the 2's complement representation. Here positive numbers are represented as a simple binary numbers, with the restriction that the MSB is 0. Negative numbers are represented as simple positive binary numbers which are then complemented and logic 1 added to the LSB (ignore any carries out of the MSB (MSB=1)).

In this project, the successive samples of the filter and desired response inputs are obtained from a 12-bit ADC operating in bipolar manner. This sampled data is expressed in binary offset form which cannot be directly processed by the CPU, in which a 2 s complement representation of binary numbers is normally used. Fortunately, however, the difference between these two forms of representation is that the sign bit value (MSB) of one is the inverse of the other. So to achieve a conversion, the sign bit

(MSB) of the ADC binary output is inverted using a TTL 74LS04 inverter gate before accessing the data.

6.4.2 Writing Data into the MC68000

It has been evolved from the simulation results, that the best performance of the filter could be achieved by employing blocks type (section 4.3.4) in the hardware implementation of the LMS ANC. In this case a block of 16 successive samples of the input as well as the output and the desired response input are read by the MC68000 every 128*16 computation period (sampling instants).

The PIA provides efficient communication between the input-output sampled data and the MC68000 CPU. As has been mentioned before the number of input pins that can be used for this task is limited to 6 (port B). This limitation could be surmounted by applying proper control pulses to two of the control lines of the PIA, in our case CA2 and CB1, as shown in fig 6.4. They should be square pulses of 50 % duty cycle and will act as INPUT READY signal generated by the components in the hardware design, of 32 and 64 clock cycles width respectively in order to govern the timing of data entering the MC68000.

This operation implys the following procedures:

(1) Initializing the PIA, this can be achieved by setting the data direction, data and control registers of the PIA as shown in appendix E.

Since the data is entered into the MC68000 every 128*16 (N*16) computation periods (sampling instants), another control pulse, namely CB2 is required. So the status bit of the control register B is set to 1 every 128*16 computation periods to load a new blocks

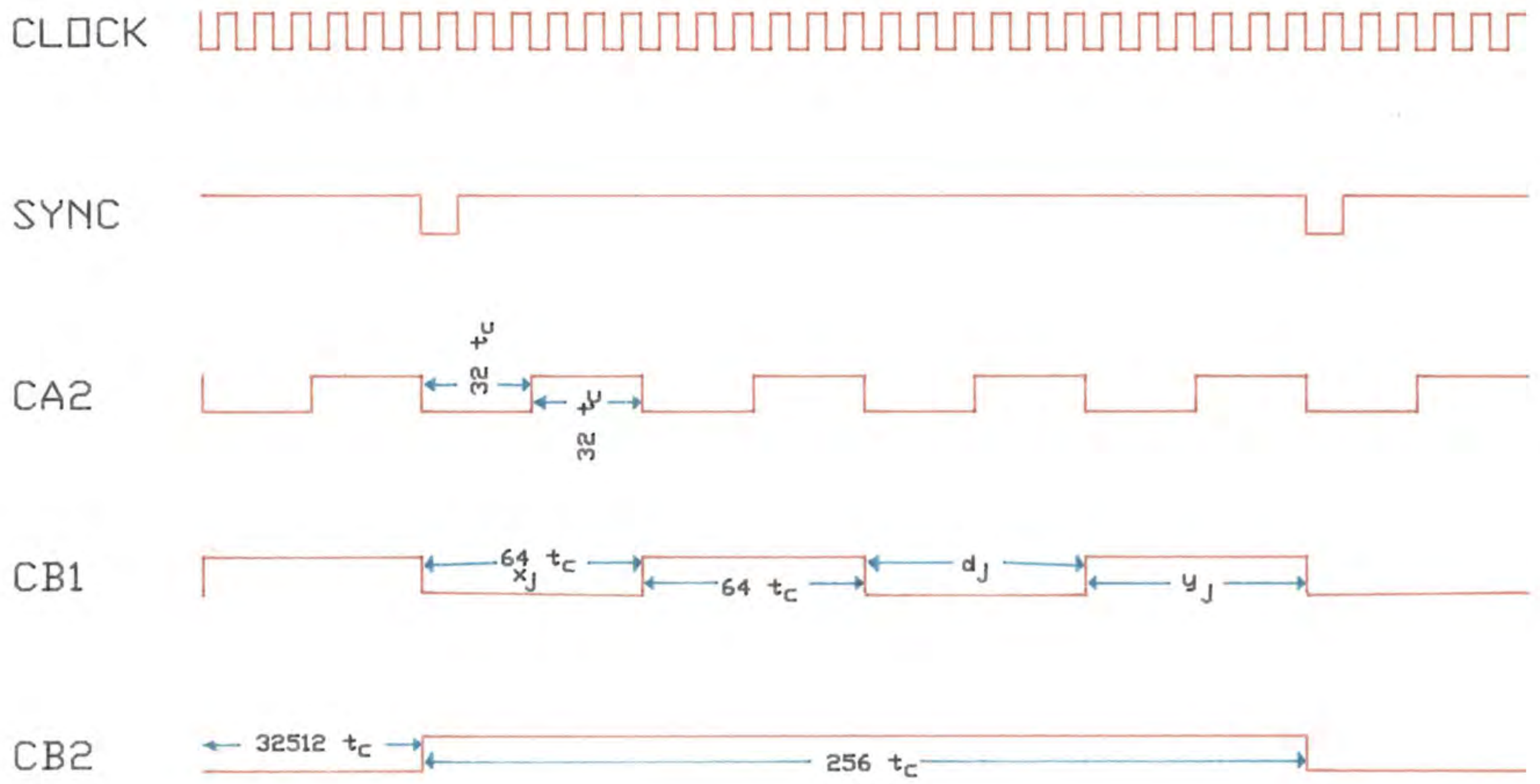


Fig. 6.4 The PIA's Control Inputs Timing Diagram

of data (16 successive filter input, desired response input, filter output samples). It is also set to 1 every 128 computation periods to start the coefficients updating operation and write the coefficients into the external RAMs. So this pulse goes low for one sampling period and high for the rest 127 sampling periods as illustrated in fig. 6.4.

(2) Testing the status bits and reading the data. After the PIA is initialized, the MC68000 tests bit 6 on LOW-to-HIGH transition of CB2 to see whether it is set to 1 or not. Once the bit is set, it informs the CPU that the INPUT READY and the 6MSBs of the filter input sample could be loaded into the processor via the data register. These 6MSBs are stored in a data register of the CPU, in our case D0. The 6LSBs are loaded whenever the status bit of control register A is set on LOW-to-HIGH transition of CA2 and are stored in the data register D1, for further processing. The desired response input and the filter output samples are read by testing the transition state on CB1 and CA2 as illustrated in fig. 6.4. Since the data is transfered to the CPU over port B, an important factor should be considered when the status bit 6 is tested. Since, the only way to clear the status bit 6 is to read the data register A, a dummy read is necessary because the data register A is not involved in this reading operation.

6.4.3 Adjusting Bits Locations

It is seen that after read instructions are terminated, the data word is split in two data registers which is an awkward situation. In order to get them in the desired locations, the 6MSBs are shifted 6 bit positions to the left to occupy the second byte of D0, (D0 is 32-bit). Then the 6LSBs transfered to take a place in the first byte of D0. The bits could be set in the correct order by shifting the contents of D0 2 bits to the right, so that the input sample occupies bit 0 through bit 11 of the D0 word (16 bits). The same set of instructions is carried out for the desired response input and filter output samples.

6.4.4 Extending the 12 Bits to 16 Bits

Since arithmetic operations are only performed on bytes, words or long words, the 12-bit data should be extended to a word. One should be careful in extending this data and take into account that it might be negative as well as positive. So one way to achieve this extension is by examining (testing) the sign bit of each binary number sample, whether it is zero (indicates positive number) or one (indicates negative number) and setting bits 12 to 15 to the detected sign bit value.

The same sequence of instructions will be involved in the case of reading the desired response input and filter output, but instead different data registers are used.

The error output may be computed by subtracting the filter output from the desired response samples fed to the MC68000.

6.4.5 Circular List Process and Saving Input & Output Data

Referring to the coefficients updating eqns. 3.5.1 and 3.5.2, updating 16 recursive and 16 non-recursive coefficients requires the existence of the present input and output samples as well as the previous 15 input and 15 output samples. In other words, these previous input-output samples must be saved somewhere and invoked whenever called by the CPU in order to implement the coefficients updating operation.

One way of storing the data is using a circular list process. The filter input and filter output samples are stored in the memory locations addressed by the address registers A4 and A6 in the MC68000 respectively. The memory holds a record of the current and previous input and output samples represented in two blocks each in which the time interval between the two blocks is N*16 sampling periods. The first one contains the previous samples and the second one contains the new samples. These samples are expressed as x_j, x_{j-1}, ..., x_{j-15}, $x_{j-(N*16)}$, $x_{j-(N*16)-1}$, ..., $x_{j-(N*16)-15}$ and y_{j-1}, y_{j-2}, ..., y_{j-16}, $y_{j-(N*16)-1}$, $y_{j-(N*16)-2}$,..., $y_{j-(N*16)-16}$, where N equals 128 sampling periods.

By applying the address register indirect with postincrement addressing mode, the CPU increments the address registers A4 and A6 to allow the sequential addressing of the sampled data, starting with the most recent input and output samples x_j and y_{j-1}. After the last samples x_{j-n+1} and y_{j-n} are individually addressed, a new input and output sample are then transfered from the first block into the second block and written into the memory locations addressed by these addresses replacing the last samples. Therefore whenever the coefficients updating operation takes place, A4 and A6 address the latest sampled data record and the previous samples are rippled through the memory locations addressed by the contents of A4 and A6 in the first block. That is to say, in each updating period (N sampling periods) one new input and output sample are transfered from the second block to the first one replacing the latest input and output samples x_{j-n+1}, y_{j-n}. This operation is continued until all new samples in the second block replace the old samples in the first block, one by one every updating period, so at the end a new block of 16 successive input and output samples, as mentioned before, are read by the MC68000. So after N*16 sampling periods, 16 new successive samples are loaded into the second block and so on.

An example of the circular list operation is shown in table 6.2, where a sampled data record of five samples is addressed. The addressing of sampled data is shown for the first three updating periods in the first block. x1, x2, x3, x4, x5 represent the successive samples in the first block, z1, z2, z3, z4, z5 represent the successive samples in the second block, where x1 and z1 are the most recent samples.

first updating period

address	sampled data
0	x1
1	x2
2	x3
3	x4
4	x5

second updating period

address	sampled data
4	z5
0	x1
1	x2
2	x3
3	x4

third updating period

address	sampled data
3	z4
4	z5
0	x1
1	x2
2	x3

Table 6.2

6.5 The Filter Coefficients

The coefficients play an important role in the design and implementation of the ANC system and govern the stability of the system. This section investigates how the MC68000 can be applied to update these coefficients in the LMS algorithm and how the data, comprising the FAD multiplier coefficients, scaling factor and control bits can be represented according to table 5.1 (described in section 5.2). Finally it discusses the software routine that combines two recursive coefficients in one location of a data register (and the same for non-recursive coefficients and how they can be seriall written into an external RAM used as the coefficients storage.

6.5.1 Updating the Coefficients

In the view of the LMS algorithm, the next non-recursive and recursive coefficients may be computed according to the following equations

$$\underline{a}_{j+1} = \underline{a}_j + 2\mu_1 \underline{X}_j e_j$$

and

$$\underline{b}_{j+1} = \underline{b}_j + 2\mu_2 \underline{Y}_j e_j$$

respectively.

Thus the updating operation involves time shifting, multiplication, division (since $0<\mu<1$), and addition. It has been pointed out earlier in chapter 5, that the 16th order filter is implemented by cascading 8 second order sections each one with an individual transfer function. A 68000 assembler software routine, listed in appendix E, is carried out to accomplish this task as follows.

(1) The updating operation is first established by loading a data register, for instance D6, with the number of second order filters cascaded together (8).

(2) The CPU fetches the input sample from the memory location addressed by the contents of the address register A4, as has been mentioned previously, and multiplies it by the previous error output (e_j). Fortunately the signed multiply instruction (MULS) is provided in the 68000 which makes the task simpler and easier to understand.

(3) The multipilcation result is scaled down by a proper convergence factor, μ, to approach the state in which the desired signal could be separated from the noise contaminated with.

(4) The data extracted from the former step is added to the previous non-recursive coefficient, of the same order, and written back into the same location addressed by the contents of A5. and is therefore saved for use in the updating operation of the next updating period.

The recursive coefficient can be updated following rather the same steps. However the filter output vector ($\underline{Y}_j$) is involved in the multiplication rather than $\underline{X}_j$ and a different convergence factor is used to scale down the resultant data obtained.

An identical sequence of instructions is carried out to complete the updating of the 8 second order sections by incrementing the contents of D6 by one after the completion of each section updating operation.

Care must be taken when the input or output samples are fetched from their storage in the MC68000 memory. The address of their locations should be examined while fetching every sample and compared with the last address (which is $XX1F since the contents of A4 and A6 are incremented by two on word operand operations). If the last addresses have been reached, A4 and A6 should be reloaded with their first addresses immediatly to avoid the overflow.

6.5.2 The Coefficients and Control Bits Representation

One problem that arises in implementing recursive digital filter (fixed coefficient and adaptive), is the possibility of instability when the poles take values outside the unit circle in the Z-plane (the unstable region). Because of this instability problem the recursive coefficients are considered to be less than 1. In fact there is no restriction on the locations of zeros, so they could be on any side of the unit circle. But for the sake of simplicity, and because the updating process is carried out in fixed point arithmetic (finite precision representation), the resultant coefficients are considered as a fractions less than 1 as well.

Referring to table 5.1, it is observed that the first bit of the second non-recursive and recursive coefficient entering the FAD chip coincides with clock pulse number 15. It is more convenient to join both of them in one data register. Both of recursive coefficients ,a, and ,b, are transfered to a data register, for instance D3, so the first one is located in the least significant word (16 bits) while the second one occupies the most significant word of D3 (16 bits) in the MC68000. The non-recursive coefficients A and B occupy the data register D7 in the MC68000. In the view of table 5.1 the negative and positive values are distinguished by the sign control bits a_0, b_0 and A_0, B_0 for recursive and non-recursive coefficients respectively. (1) A test instruction is used to examine the sign bit of each coefficient individually, a_0 and A_0 are set to 1 if the coefficients are positive and to 0 if negative. (2) The ADD/SUBTRACT control bits a_s and A_s are set to zero in the case of positive values and to 1 for the negative values. Since b and B are already equal to or less than 1 this control bit is eliminated. (3) The test command is employed to examine the sign of A and a individually. It has been emphasised earlier that the FAD accepts 12 bits of the fractional part of each coefficient. Since A and a are 16 bits including the sign bit, the coefficient.s bits are truncated to 13 bits (the MSB is the sign bit) by shifting them to the right 3 bits. Then bits 13 (A_0,a_0) are set to 1 if the coefficient is positive or to zero if it is negative. Bits 14 (A_s,a_s) are set to zero in the case of positive values and to one for the negative values.

Steps (1) and (3) are repeated for b and B coefficients. Notice that the ADD/SUBTRACT bit is omitted in these coefficients since they are equal to or less than unity.

In order to allocate the LSB of B and b in the bit positions 14 (coincides with clock pulse number 15) in the data register D3 and D7 respectively, the least significant word (LSW) of each is shifted 2 bit positions to the left and then the entire 32 bits of D3 and D7 (long word) are shifted to the right 2 bits to join or combine each set of the coefficients in one data register.

In addition to the coefficient bits, 4 bits (s4,s3,s2,s1) representing the input scaler factor are allocated into bits 27 to 30 (coincides with clock pulse number 28-31) of D3 in order to prevent the overflow at all nodes of the FAD IC, it is applied,in our case, to the FAD in each second order filter section.

The input data selector C1 is set to 0 for the first 7 timeslots. In this case the data emerging from the filter output is fed back internally to the filter section at the begining of next timeslot. C1 is set to zero in the last second order to indicate that the output of the 16^{th} order filter is valid at the filter output (pin 15) and the data applied to the FILTER IN (pin9) is input to the filter section in the next sampling period. C2=1 for all second order sections since the non-unity B coefficient case is chosen.

6.5.3 Coefficients Entry

Because the limited output pins used to write the coefficients into the FAD IC, and since it has been emphasised that the FAD IC requires its coefficients in serial, bit by bit, simultaneously, a software routine is carried out by the MC68000 for this purpose to overcome the limitation of the output ports.

Once the coefficients and the control bits are located in their desirable positions, the MC68000 starts executing the software routine by loading a data register, for instance D1, with the number of each set of coefficient in each second order section (2) and D3 for instance, with the hex number (0F) (number of bits in each coefficient).

In the last section the updated non-recursive and recursive coefficients, including the control bits, are saved in D3 and D7 respectively. In order to write them into RAM in the serial form, half of the recursive (16 bits) and half of the reversed order bits non-recursive coefficients (16 bits) are bit by bit joined in D0. The reversing process of the non-recursive coefficients is accomplished in the following steps:

(1) D2 is loaded with the number of bits-1 (15 bits).

(2) The coefficient is loaded into D0 and saved in the memory location addressed by the contents of A, refer to appendix E.

(3) D0 is shifted as a word number of bits contained in D2 to the left.

(4) D0 is shifted as a long word to the left one position and the MSB is placed in the LSB position in the MSW of D0.

(5) D2 is decremented by one each time and this routine is continued until all the 15 bits are completely reversed in order. To reverse the order of the MSB, the contents of D0 are shifted one bit position to the left (as a long word) allocating the MSB in the LSB position in D0.

(6) recursive coefficient word occupies the LSW of D0. An example of reversing a 6-bit non-recursive coefficient assuming it as a word is illustrated in fig. 6.5.

				LONG WORD									
		WORD 2						WORD 1					
		11	10	9	8	7	6	5	4	3	2	1	0
D2 = 5	(2)	x	x	x	x	x	x	a5	a4	a3	a2	a1	a0
	(3)	x	x	x	x	x	x	a0	0	0	0	0	0
	(4)	x	x	x	x	x	a0	0	0	0	0	0	0
D2 = 4	(2)	x	x	x	x	x	a0	a5	a4	a3	a2	a1	a0
	(3)	x	x	x	x	x	a0	a1	a0	0	0	0	0
	(4)	x	x	x	x	a0	a1	a0	0	0	0	0	0
D2 = 3	(2)	x	x	x	x	a0	a1	a5	a4	a3	a2	a1	a0
	(3)	x	x	x	x	a0	a1	a2	a1	a0	0	0	0
	(4)	x	x	x	a0	a1	a2	a1	a0	0	0	0	0
D2 = 2	(2)	x	x	x	a0	a1	a2	a5	a4	a3	a2	a1	a0
	(3)	x	x	x	a0	a1	a2	a3	a2	a1	a0	0	0
	(4)	x	x	a0	a1	a2	a3	a2	a1	a0	0	0	0
D2 = 1	(2)	x	x	a0	a1	a2	a3	a5	a4	a3	a2	a1	a0
	(3)	x	x	a0	a1	a2	a3	a4	a3	a2	a1	a0	0
	(4)	x	a0	a1	a2	a3	a4	a3	a2	a1	a0	0	0
The MSB	(2)	x	a0	a1	a2	a3	a4	a5	a4	a3	a2	a1	a0
	(3)	a0	a1	a2	a3	a4	a5	a4	a3	a2	a1	a0	0

Fig. 6.5 Reversing the order of the NRcoeff bits

A precaution should be taken before executing word shifting instruction. The data register D0 must be reloaded with the data in D3 which is stored in the memory location addressed by the contents of A before the shift operation takes place.

As soon as the last process is terminated, the write operation takes place in the software routine. The basic idea of this operation is to transfer (or transmit) data from the CPU to RAM1 controlled by the R/$\overline{W}$ pulse of the CPU (or the PIA). There are two types of data which should be transfered, the external memory addresses and the coefficients. First the MC68000 addresses RAM1 by sending the starting address 0, defined by the contents of D5, over the 8 data lines of the PIA control register A, in order to write one bit of each coefficient simultaneously in that location. The long word D0 is rotated one bit to the left setting the LSB of each coefficient in bit locations 0 and 1 as exemplified for a 6-bit non-recursive and recursive coefficient in fig. 6.6. These locations coincide with the first 2 data lines of the PIA control register B which are used to write the coefficient.s bits into RAM1. Once these 2 bits are transmitted, there is no point in keeping them in the CPU so they are eliminated by executing LSR.W #02,D0 (logical shift right word) instruction, which shifts the contents of D0 right two bits and clears bits 14 and 15. The next address of the RAM1 is generated by incrementing the contents of D5 by one; and so on.

By following an identical sequence of instructions and taking into account that the data register D0 should be loaded with new coefficients (stored in D3 and D7) after the transfering of the previous one is completed, the complete 2*256 bits of the

	LONG WORD											
	WORD2 (NRcoeff)						WORD1 (Rcoeff)					
	11	10	9	8	7	6	5	4	3	2	1	0
(1)	a0	a1	a2	a3	a4	a5	b5	b4	b3	b2	b1	b0
(2)	a1	a2	a3	a4	a5	b5	b4	b3	b2	b1	b0	a0
(3)	a1	a2	a3	a4	a5	b5	0	0	b4	b3	b2	b1
(2)	a2	a3	a4	a5	b5	0	0	b4	b3	b2	b1	a1
(3)	a2	a3	a4	a5	b5	0	0	0	0	b4	b3	b2
(2)	a3	a4	a5	b5	0	0	0	0	b4	b3	b2	a2
(3)	a3	a4	a5	b5	0	0	0	0	0	0	b4	b3
(2)	a4	a5	b5	0	0	0	0	0	0	b4	b3	a3
(3)	a4	a5	b5	0	0	0	0	0	0	0	0	b4
(2)	a5	b5	0	0	0	0	0	0	0	0	b4	a4
(3)	a5	b5	0	0	0	0	0	0	0	0	0	0
The MSBs	0	0	0	0	0	0	0	0	0	0	b5	x

Fig. 6.6 The coefficients writing operation
(1): Contents of D0
(2): Rotate the contents of D0 left one bit
(3): Shift WORD1 right two bits
x denotes 'don't care'

coefficients are located in the proper RAM1 locations addressed by the CPU.

Attention must be paid to the situation when the transfer of the last bit (MSB) of each coefficient takes place, since after the bit 14th of each is transmited, these 2 MSBs occupies the last 2 bit positions (30 and 31). Fortunately the non-recursive MSB could either be 0 or 1 (refer to table 5.1). So to allocate the recursive MSB in bit position 1, the contents of D0 (long word) are rotated left three bits.

There are some points which should be remembered when executing this routine:

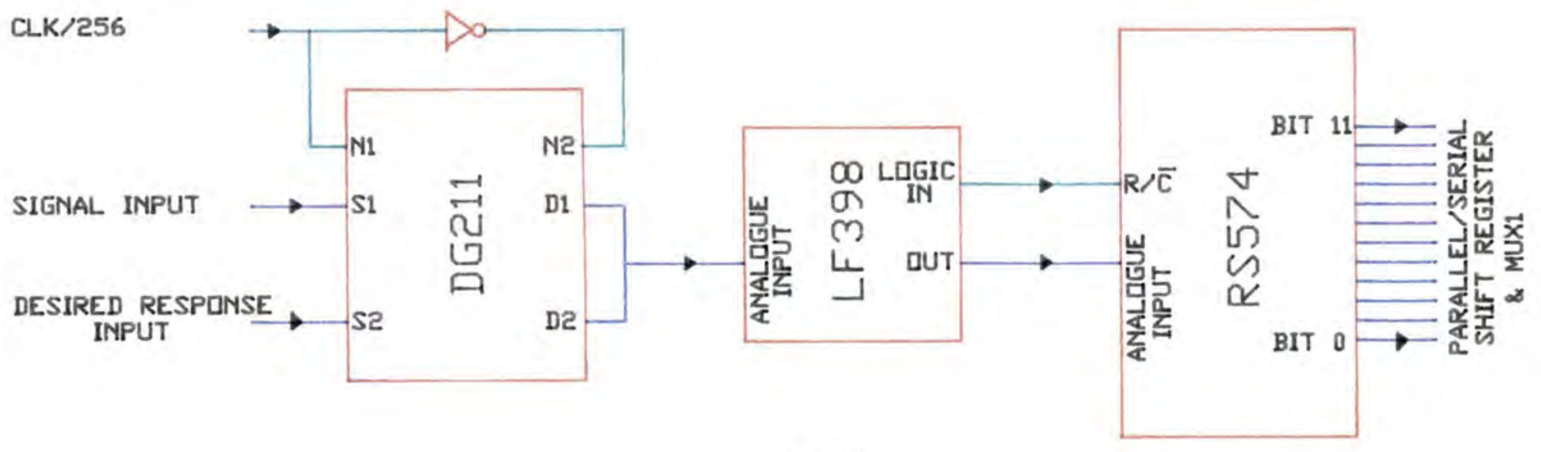
(1) the contents of the control registers A&B should be cleared since both are used in a write operation, in order to address the data direction registers.

(2) selecting the 8 data lines of A (port A) and the first 2 lines of B as outputs by the loading data direction register A with $FF and B with $03.

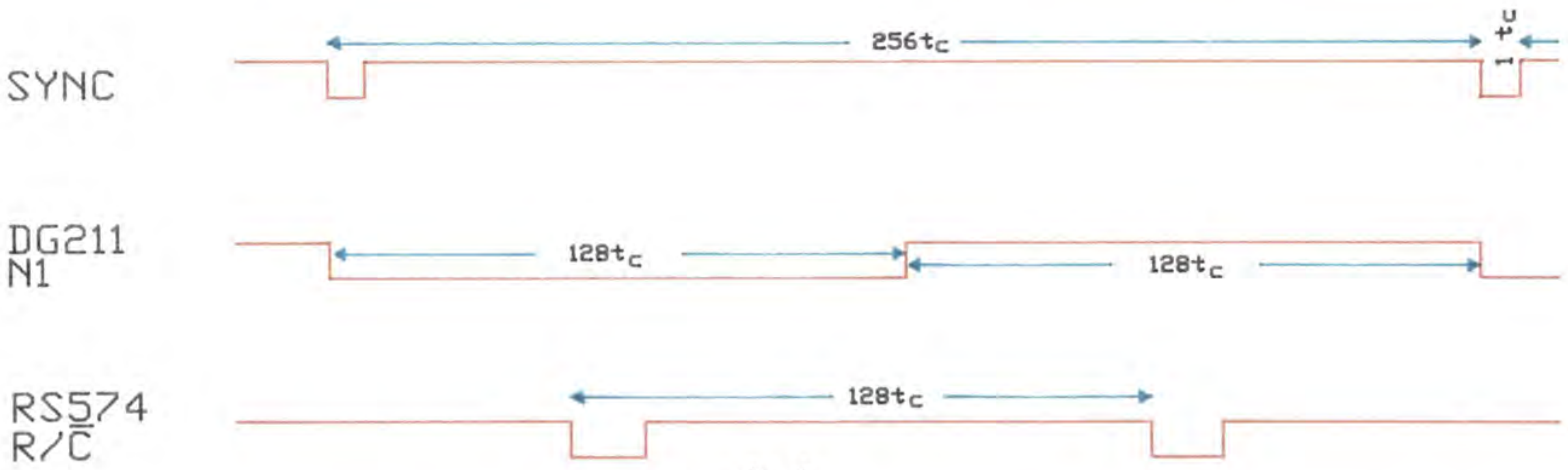
(3) addressing the data (peripheral) register by setting bit 2 of each control register to 1, refer to appendix E.

6.6 System Implementation

In this project the realization of the adaptive filter, applied as a noise cancelling system, implies hardware design supported by a 68000 assembly language program, listed in appendix E. When the software program is first run, the CPU sets up the PIA and organizes its ports (A & B) in the desirable directions. Since the sampled input-output data must be saved, the CPU loads the address registers A4, A6 and A5 with the starting address of the memory locations used as a storage of input samples, output samples and the updated coefficients respectively. When the PIA is first initialized and the address registers are loaded, the CPU executes the coefficients updating program. This execution is initialized by means of a control pulse produced by the system circuit and sent to the MC68000 through the control input line CB2. The positive edge of this pulse coincides with the existence of the sampled data entering the MC68000, see fig. 6.4. It has been mentioned before that the complexity of the system hardware arises from the limitation of the input-output ports available on the MC68000. Another effect of this limitation is multiplexing the ADC analogue input between x_j and d_j. A DG211 analogue switch is used to switch between x_j and d_j fed into the ADC as illustrated in figs. 6.7 (a) and (b). Since the data should be written into the MC68000 within one computation period (8 timeslots), the control input N1 of the analogue switch is held LOW for 4 timeslots passing x_j and then is held HIGH for the other 4 timeslots allowing d_j to pass to the ADC as shown in the timing diagram in fig.6.7. Once the data has entered the MC68000 the updating process takes place and is executed using eqns. 3.5.1 and 3.5.2 according to the LMS algorithm. As soon as this process is terminated, a stream of the coefficients comprising control bits and scaling factor bits is written into RAM1 and consequently into the RAM2 and the FAD IC. The new set of coefficients are written into the FAD every 128 computation periods and remain constant for that length of time. The input samples enter the FAD every computation period occupying the first half of the first timeslot (16 bits). The filter output is valid after a delay of 8 timeslots, so it appears in the first half of timeslot number 9, the next output is valid in timeslot 17



(a)



(b)

Fig. 6.7 (a,b) Filter Inputs Multiplexer Circuit and Timing Diagrams and so on. This output is then converted to analogue form by the DAC and simultaneously applied to the ALU to be subtracted from d_j producing the error output e_j, which is also converted to the analogue form ,as illustrated in fig. 5.10.

6.7 Conclusion

The 68000 microcprocessor was chosen because of its ability to effeciently implement the LMS equations for updating the coefficients. It has a comprehensive set of instructions, a wide number of addressing modes and its most convenient feature is its ability to operate with any of the instructions on more than one data size (byte, word, or long word). We also successfully detected an obscure fault in the FAD IC after much careful analysis.

This chapter has demonstrated the reduction and saving in hardware that can be achieved by developing many 68000 software routines that are compatible with the hardware implementation of the ANC. These software routines include, reading blocks of 16 samples of the filter input, desired response input and the output every 128*16 computation periods (one set of samples every computation period), saving the previous input and output samples, updating the coefficients, writing non-recursive and recursive coefficients simultaneously, bit by bit into the external RAMs. Finally, the general function of the complete hardware implementation of the ANC has been investigated.

CHAPTER 7

Experimental Results

7.1 Introduction

In recent years, there has been considerable interest among researchers to extend the FIR adaptive filter to more general feed-back or IIR configurations. The computational cost of FIR filters was a great encouragement to the development of the IIR filters. An advantage of such an extension is the substantial decrease in filtering hardware (and/or software) that the IIR design presents over an FIR design with equivalent performance.

The first part of this chapter explores the hardware implementation results of the 16^{th} order adaptive filter based upon the LMS algorithm, performed as an ANC. The second part is devoted to a discussion of the stability of the IIR adaptive filter.

7.2 The ANC System Performance and the Results

The ANC system was constructed employing the FAD IC as a real time 16^{th} order digital filter operating at a clock frequency of 2MHz. So to apply one new input sample every computation period (8 time-slot), the input signal was sampled at 4 KHz with 12 bits resolution (including the sign bit). To avoid the overflow at the ANC outputs, the input data vector $\underline{X}j$ was scaled down by the scaling factor $S=2^{-2}$.

Referring to the computer simulation results obtained in chapter 4, a 68000 software program was developed to update 16 feed-forward and 16 feed-back coefficients following the type discussed in section 4.3.4 of representing the input samples as well as the output samples involved in this operation. These coefficients were updated in the LMS manner according to the eqns. 3.5.1 and 3.5.2.

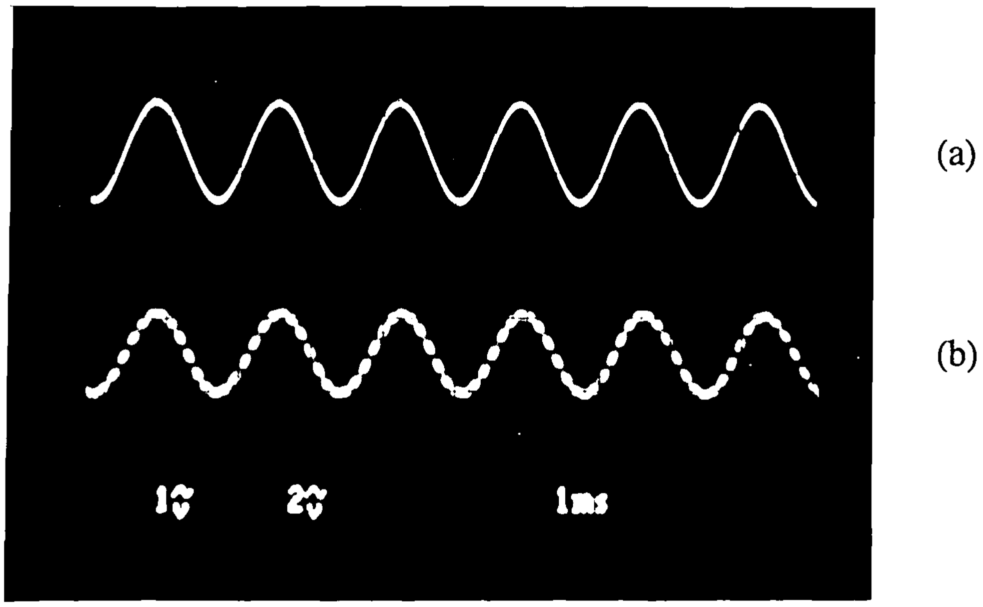
Because of the limited speed available for the updating operation, it was accomplished once every 128 sampling instants (computation periods). Using two sets of RAMs (as has been emphasised earlier in chapter 5 and 6), one bit of each coefficient was applied to the NRCOEFF and RCOEFF inputs of the FAD IC every 500nsec.

The results shown in fig. 7.1 demonstrate the use of the experimental system as an ANC. The d_j input signal was a composite signal made up of two equal amplitude sinusoidal inputs, the fundamental signal at 300 Hz and the second harmonic representing the interference at 600 Hz. The x_j input signal was the second harmonic interference at 600 Hz altered in phase and magnitude. The second harmonic sinusoid was generated by doubling the frequency of the fundamental signal via an analogue multiplier. The resultant output signal was added to its input signal using an analogue adder circuit. Fig. 7.2 shows the photograph of the inputs signals circuit.

After the adaptation, the filter reproduce the 600 Hz signal at the filter output and cancelled from d_j to produce the desired signal at 300 Hz at the error output of the ANC. This result was obtained with convergence factors $\mu_1 = 2^{-19}$ and$\mu_2 = 2^{-19}$.

7.3 Further Results

The ANC was inefficient at any frequency (below the sampling frequency) other than the one illustrated in the experimental



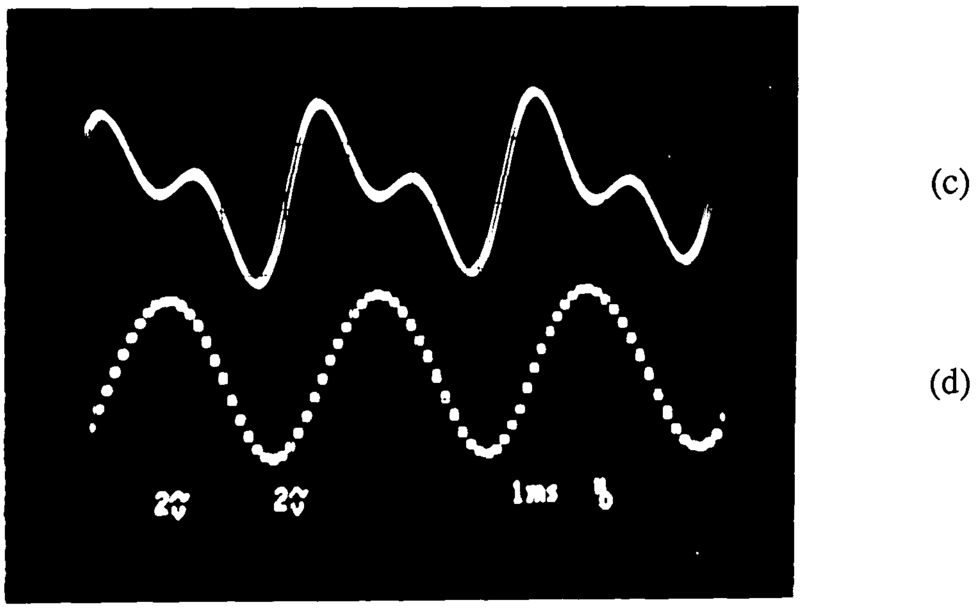


Fig. 7.2 Hardware experiment result illustrating the performmance of the LMS recursive adaptive filter
(a) filter input (2v/div)
(b) filter output (1v/div)
(c) desired response input (2v/div)
(d) error output (2v/div)

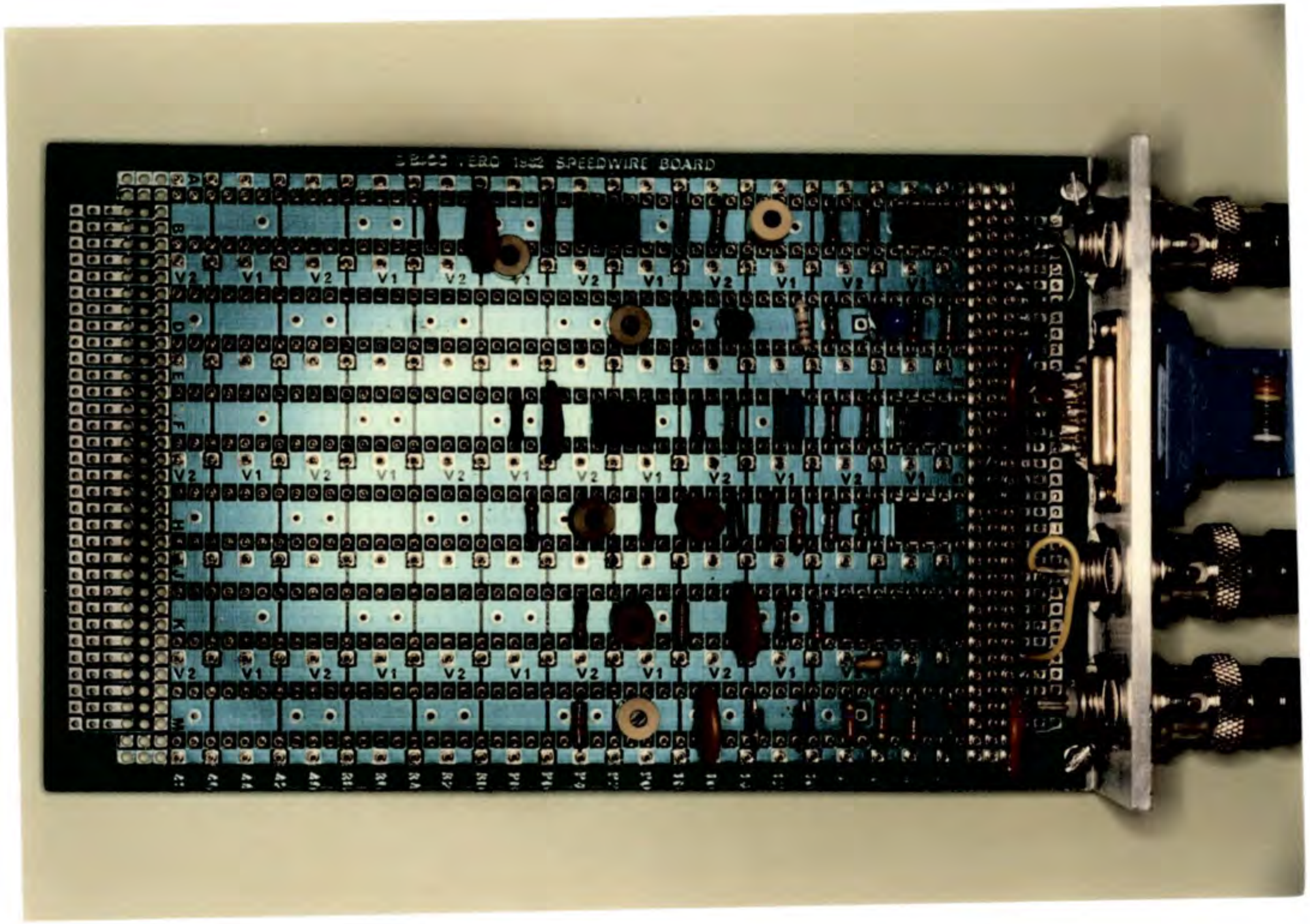


Fig. 7.2 The ANC inputs signals circuit

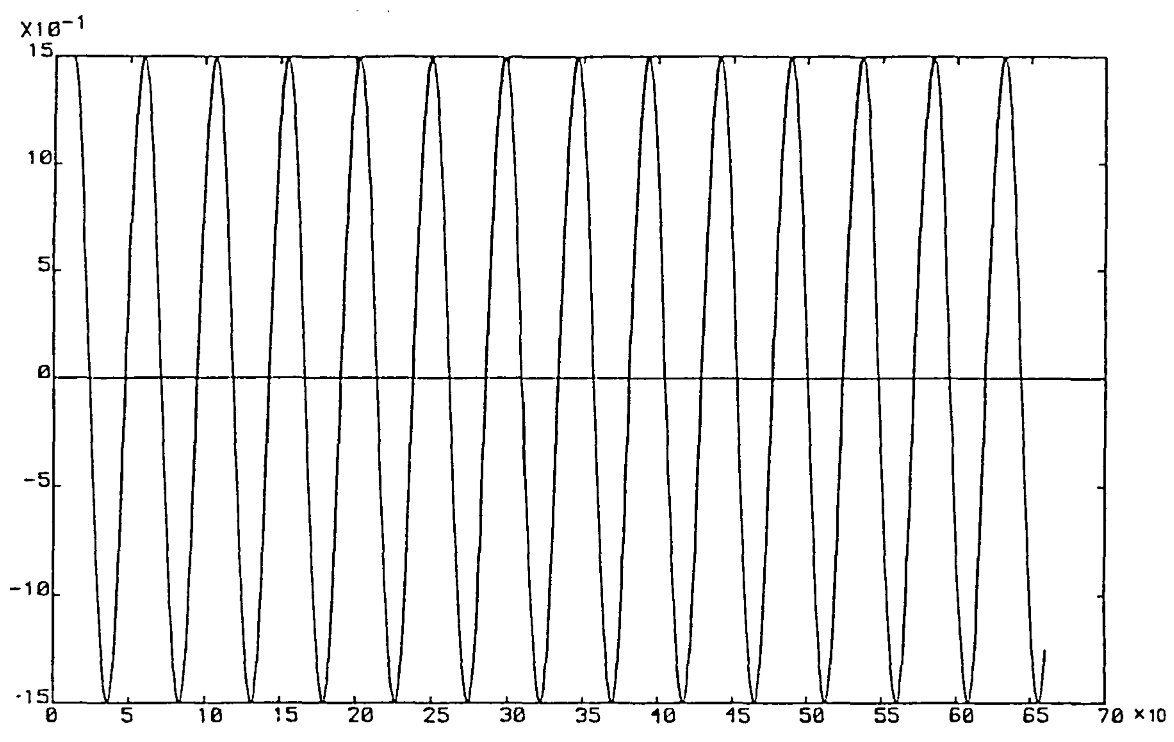
result in the previous section. So any small change in the frequency had a significant effect and degraded the performance of the ANC. In this test all other parameters were kept constant and the relation between the filter input and the signals comprising d_j were fixed.

In the light of these tests, the computer simulation program developed in chapter 4, was run for N = 128 and N = 1 at two frequencies of the input signals. The frequencies were then changed after half number of iterations. Fig. 7.3 shows the typical signals that were used while figs. 7.4 and 7.6 demonstrate the performance of the ANC when the change in the frequencies has taken place (for N=128 and N=1 respectively). Figs. 7.5 and 7.7 show the corresponding outputs y_j and e_j at the end of the same number of iterations computed at the first frequency (for N=128 and N=1 respectively). The convergence factors were set to $\mu_1 = 2^{-19}$ and $\mu_2 = 2^{-19}$, which are equal to their values in the hardware implementation.

It is obvious that both the computer simulation and the hardware implementation results, evaluated for the identical values of the parameters and under similar conditions, were in agreement. In fact, the values of μs used in that test were relatively small and were not capable of readjusting the performance of the ANC due to any modification associated the system. So the ANC was not able to track these changes, and the filter was inefficient in adapting itself fast enough to track these changes too.

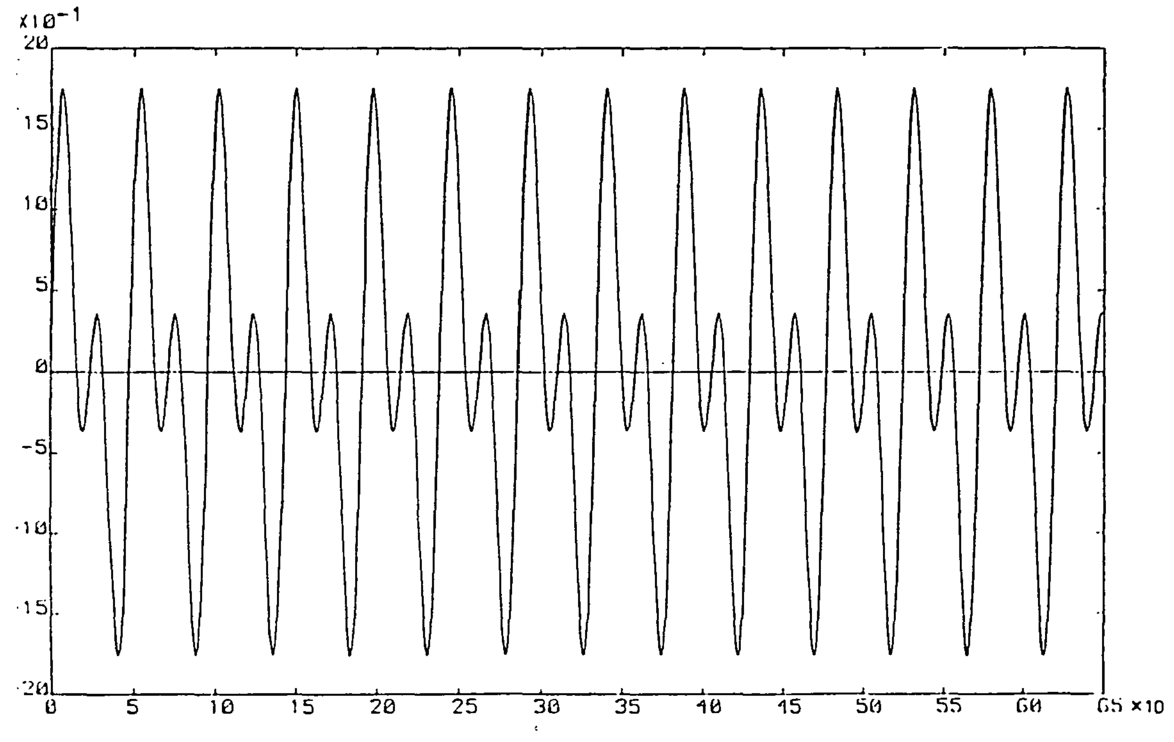
7.4 IIR Adaptive Filter Stability

Many attempts were carried out to study the behaviour of the



Time Index

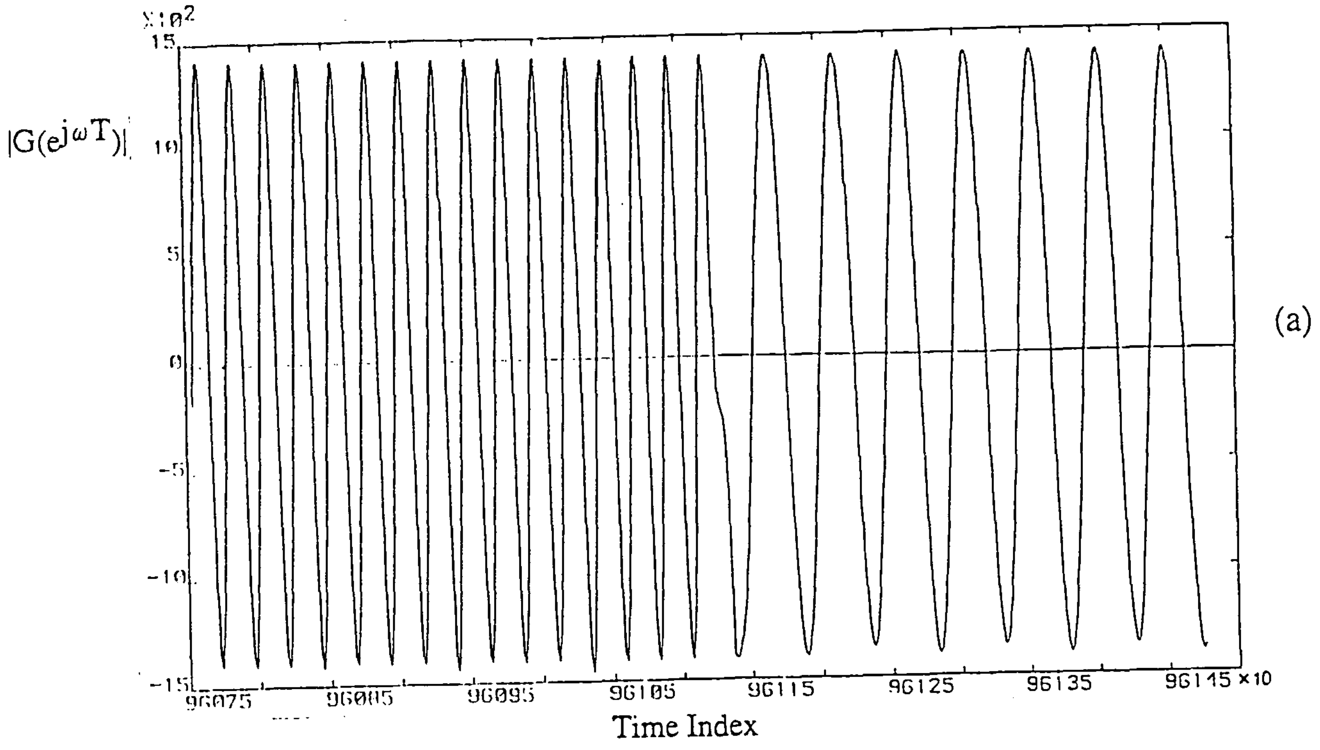
(a)



Time Index

(b)

Fig. 7.3 The LMS adaptive filter inputs
(a) signal input
(b) desired response input



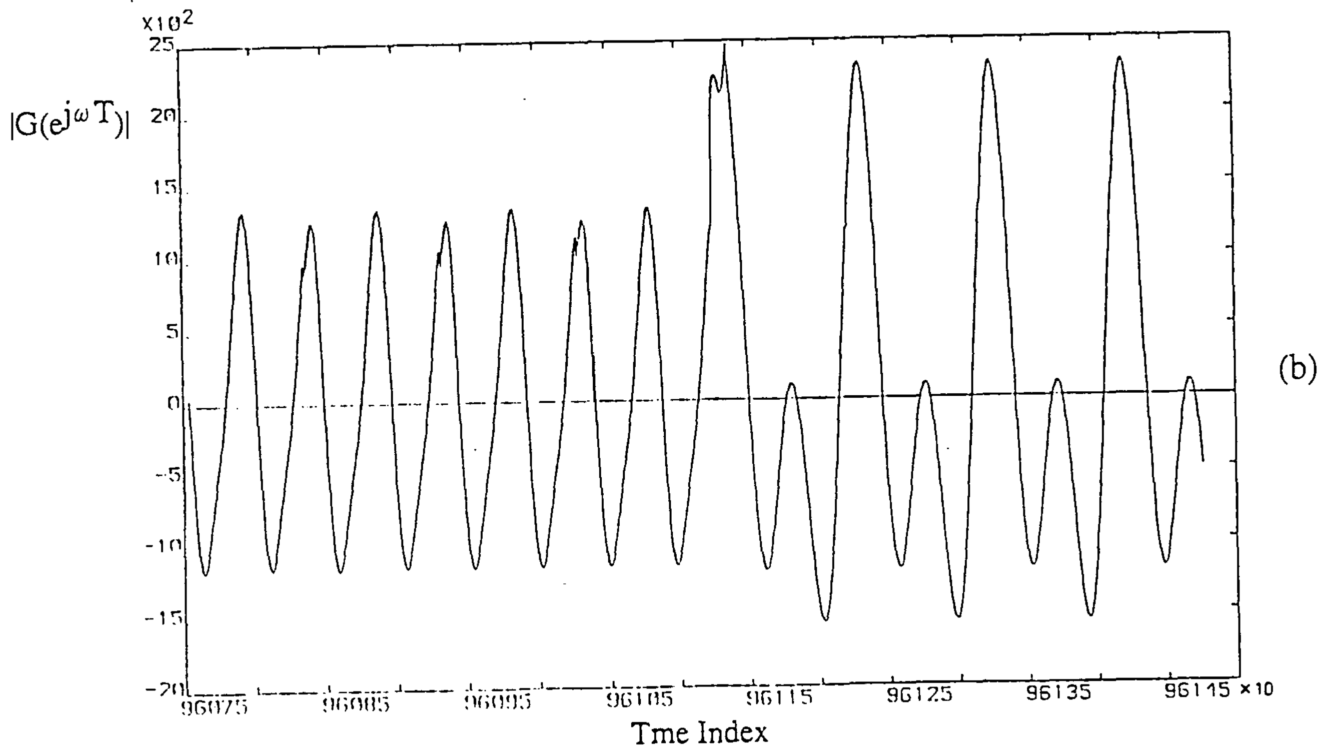
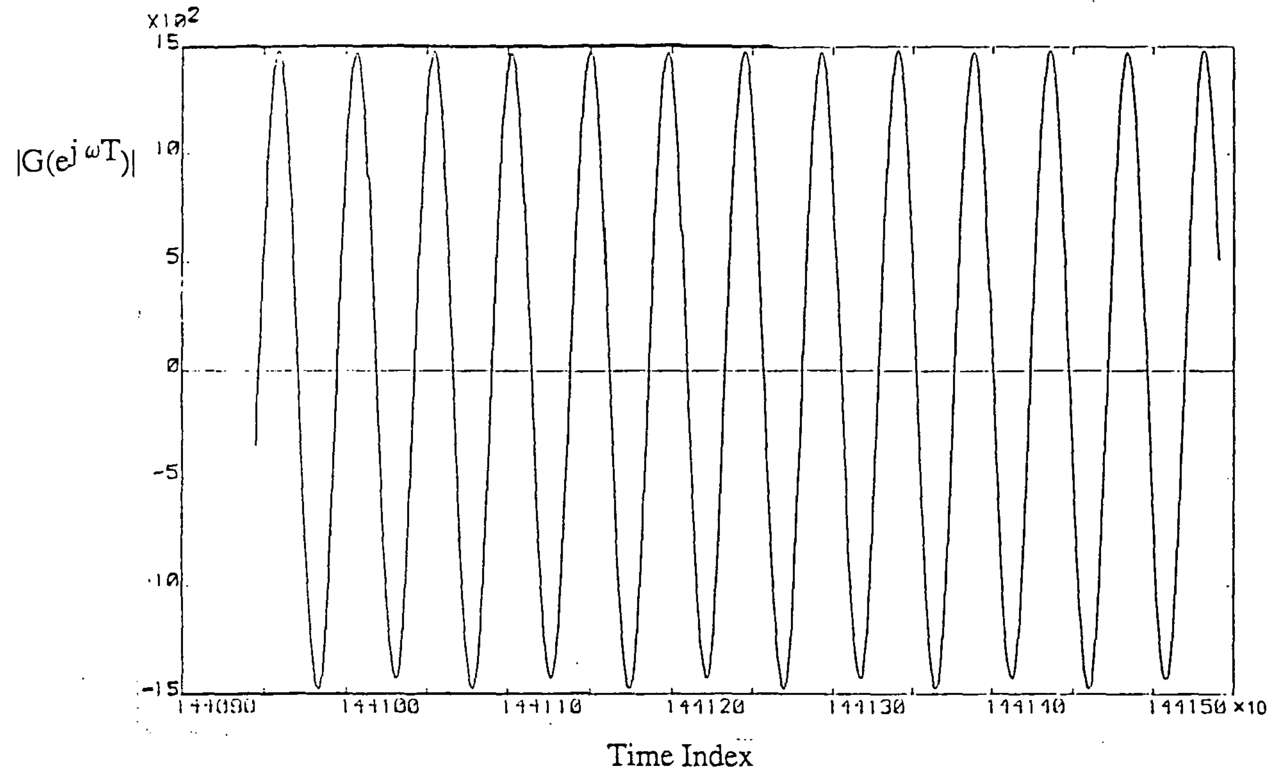
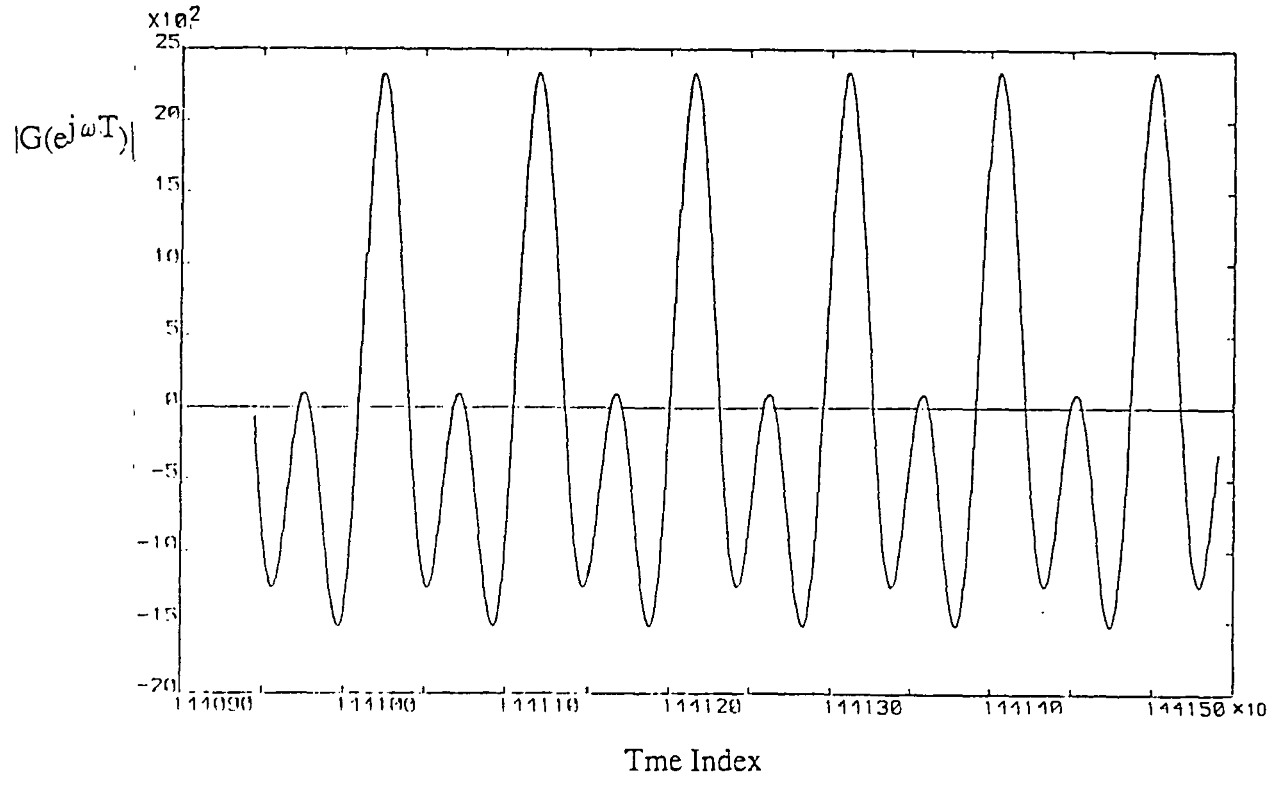


Fig. 7.4 The Adaptive filter performance at changing the frequency
(a) filter output
(b) error output

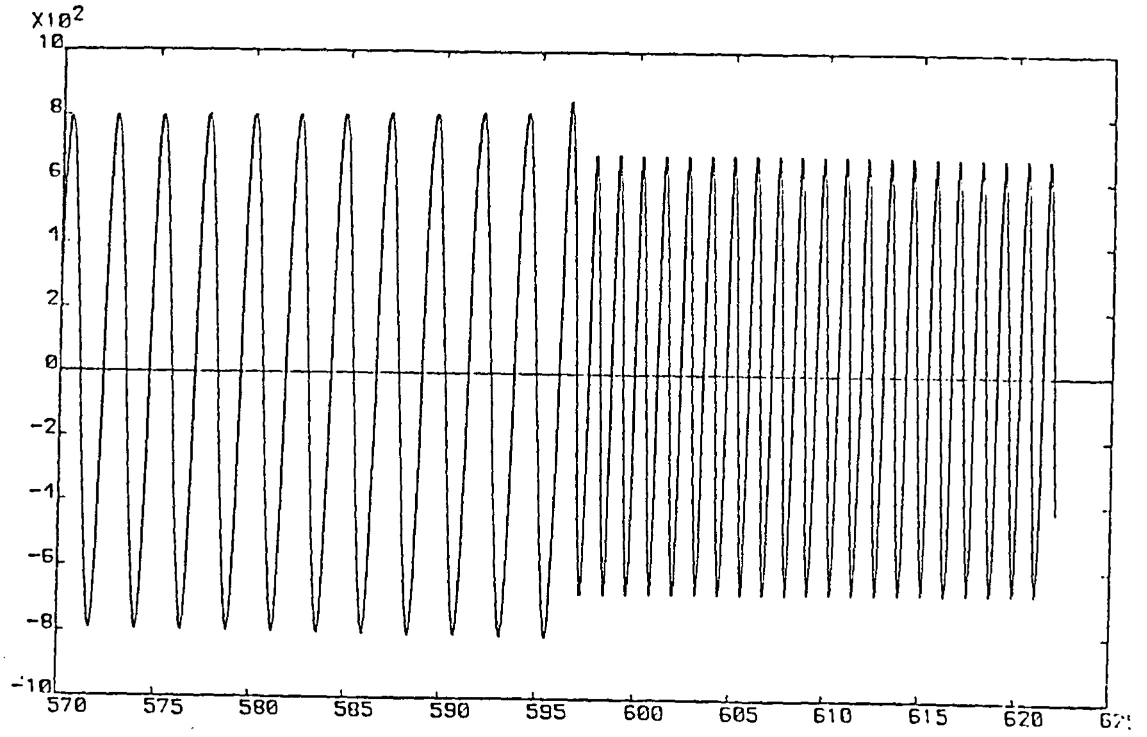


(a)



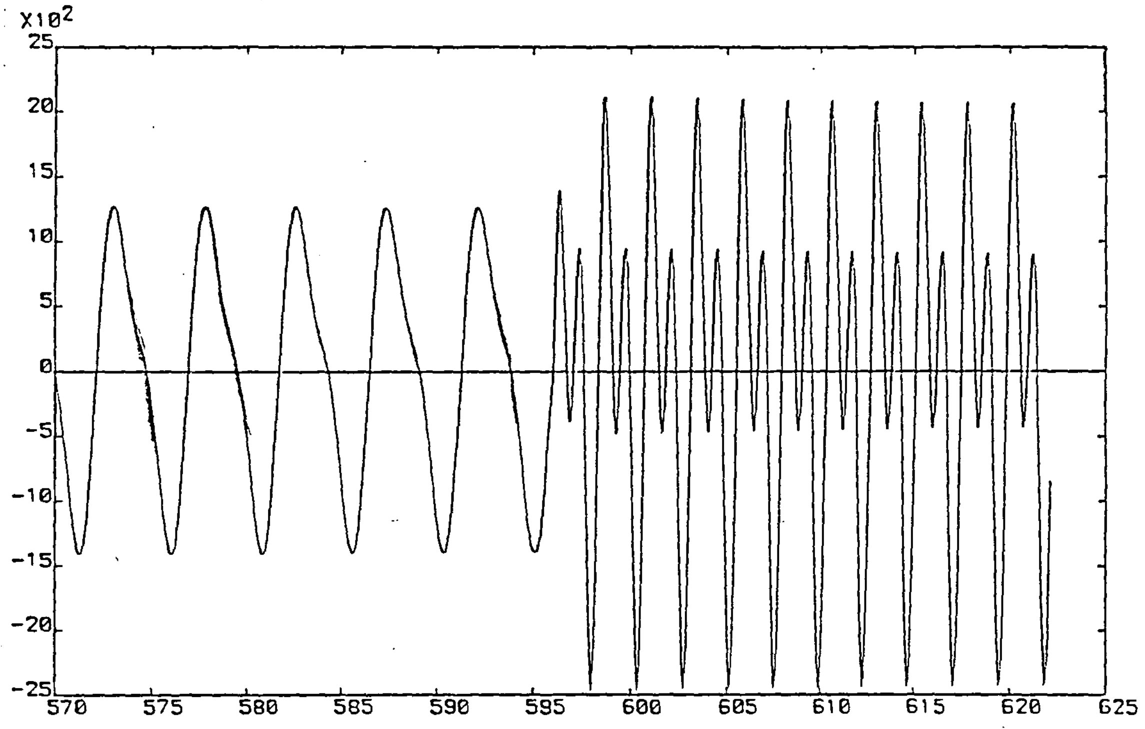
(b)

Fig. 7.5 Simulation result illustrating the LMS adaptive filter performance for the second frequency
(a) filter output
(b) error output



Time Index

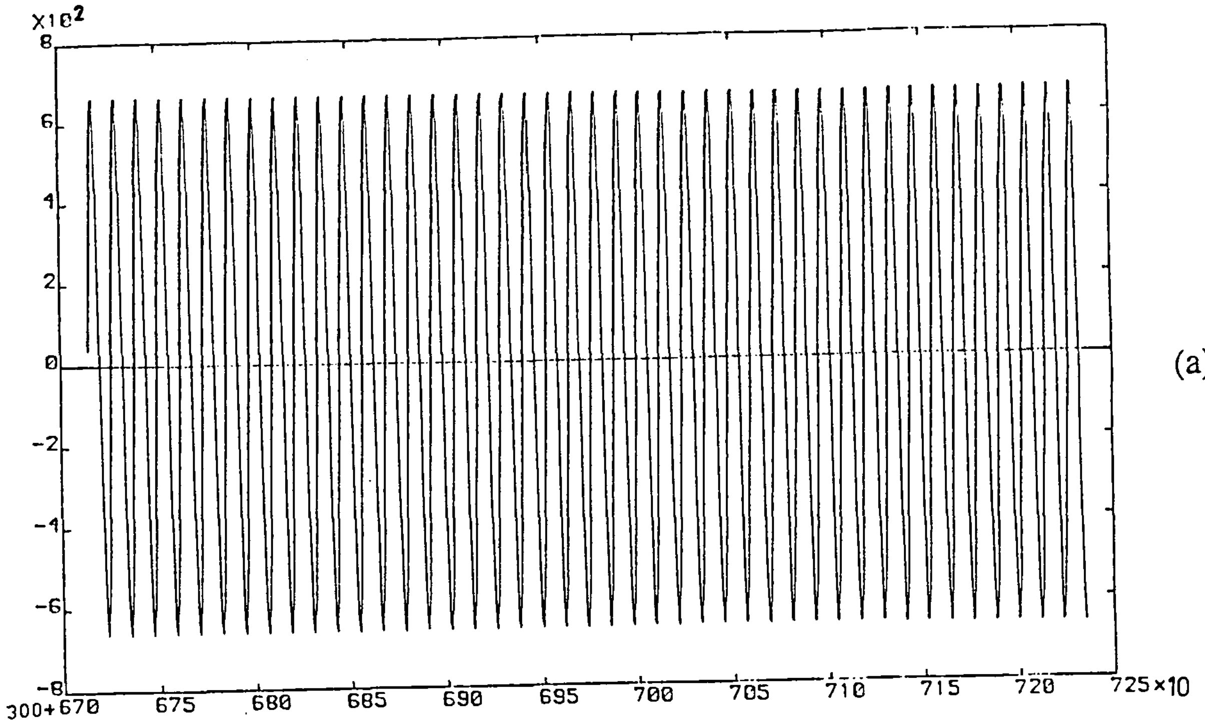
(a)



Tme Index

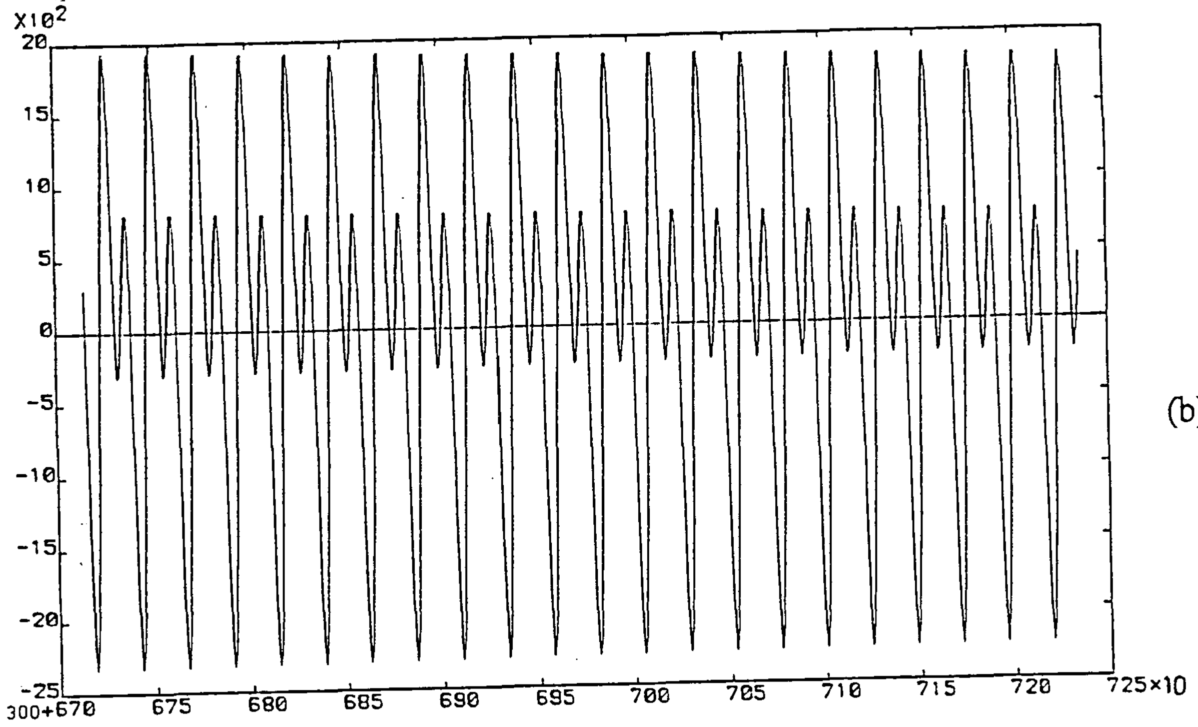
b)

Fig. 7.6 The Adaptive Filter Performance at Changing the Frequency for N=1
(a) filter output
(b) error output



(a)

Time Index



(b)

Tme Index

Fig. 7.7 Simulation result showing the adaptive filter performance for the second frequency for N=1
(a) filter output
(b) error output

ANC at larger values of the convergence factors μ_1 and μ_2. However a major problem that arose was the instability of the filter for μ_1 and $\mu_2 > 2^{-19}$. Two sources of the instability were noteable in that case, the overflow oscillations at the filter output due to the 2's complement addition and the variation in the filter coefficients. It is well known that (69), unlike FIR filters, the IIR filter stability may not be guaranteed for all choices of coefficients. So they are generally more sensitive to any change in the coefficients than are FIR filters. In fact the frequency response of these types of filters depends on the accurate placement of the few poles. Due to implementing the filter and updating the coefficients in fixed point arithmetic and the quantization employed, all these effects taken together then will lead to change in the filter's poles and might move into an unstable region.

7.5 Conclusion

This chapter has demonstrated that the ANC is effective in cancelling the additive noise contaminating the desired signal for a single frequency. The results have shown the impotence of the ANC to produce the expected performance at different frequencies with the constant parameters of the ANC at relatively small values of μs, and the adaptation is very slow and the filter does not track the variation in the frequencies of the input signals. Indeed these results agree closely with those from the simulation. For the case of larger values of μs , the instability problem arose and degraded the performance of the ANC.

CHAPTER 8

Conclusion

This work has emphasised the application of the recursive LMS adaptive filter as a microprocessor controlled ANC. Simulation studies have provided proof of the possibility of extending the FIR ANC to the more economical IIR one. It shows the opportunity of implementing the filter with fewer weights, offering a significant decrease in the software and hardware complexity of the IIR filters over an FIR filter design with equivalent performance. The hardware implementation of the ANC was extended from an extensive FIR with a significant computational load to a simple and flexible design with lower cost and memory saving using IIR filters with all digital techniques.

IIR LMS adaptive filters, which are concerned with the use of a programmable filter, in which their frequency response is adapted to suppress or attenuate the undesired signal leaving the desired components of the input signals without degradation, can also be applied in situations where an absolute minimum of information is available about the incoming signal such as in adaptive equalizers for telecommunications, data transmission systems and echo cancelation.

Although the theory of adaptive systems has been well understood for many years, it is only very recently that adaptive filters have been designed in hardware as progress in the technology advances. Many hardware implementations of FIR filters depend predominantly on the use of sampled-data analogue structures (i.e. based on charge-coupled-device (CCD) and surface acoustic wave (SAW) device). However, the performance of these analogue filters is restricted by the limitation in dynamic range, caused by effects such as nonlinearities and noise effects. Digital signal processing has many advantages over analogue techniques. These advantages include higher reliability, insensitivity to temperature changes and component tolerances, greater accuracy and repeatability, and a higher level of flexibility because they are programmable. Almost all IIR research publications are concerned with computer simulations.

The recent development in VLSI and LSI devices and, in particular microprocessors, have enabled substantial reductions to be made in both the size and the cost of high speed digital signal processing techniques. This allows adaptive filters to be designed in small, powerful single chip devices with realistic sampling rates which enable them to act as real-time processors. Recently microprocessors have been found to be efficient at implementing the LMS algorithm used to adjust the filter's coefficients with regard to the incoming signals.

The implementation described in this thesis demonstrates some of the advantages obtained by the use of these techniques. These advantages include introducing a high degree of flexibility allowing the realization of the adaptive filter with any order and the wide range of sampling rates which could be achieved by altering the clock frequency and the order of the FAD.

Computer simulation investigations demonstrate the desired performance of the adaptive filter as an ANC for stationary periodic and random signals. The effect of the inefficiency associated with updating the coefficients for every sampling period has been analyzed and it is shown that the ANC can achieve an acceptable performance when the updating operation takes place only once every 128 sampling periods, in particular, at relatively small values of the convegence factors μ_1 and μ_2. The experimental result has been illustrated to agree closely with the result of computer simulations in the case of the fixed-point as well as floating-point representations of the data involved in the design. Both experimental and simulation results demonstrate the inefficiency of the ANC performance in tracking any variation in the input signals frequency. In the light of the experimental attempts to operate the system at large values of μ, which might lead to faster convergence and better adaptation, the stability problem has been discussed as a result of the feed-back in the IIR filter performance.

We hope that this thesis has provided a coherent and comprehensive investigation of the application of the adaptive filter to noise cancelling systems. Due to the limited literature currently available concerning the convergence of the IIR adaptive filter, further work is required to investigate this property of the IIR LMS filter. Such future work should take adaventages of advances in VLSI and LSI technology and high speed digital signal processors, such as the DSP microprocessor TMS320 and the recent Fujitsu MB8764, to achieve real time implementations of adaptive filter algorithms with a minimum of hardware and a more flexible software controlled system. Further work may be carried out into developing the existing system using variable coefficients (variable step-size algorithm) which provide a better response to the change in the frequency or statistics of the input signals. Furthermore this system could be applied to different broad band signals, for example, white noise.

REFERENCES

(1) Carlson, A.B.,"Communication Systems, An Introduction to Signals and Noise in Electrical Communication", McGraw-Hill Book Company, Second Edition, 1975.

(2) Shanmugam, K.S.,"Digital and Analog Communication Systems", John Wiley & Sons, 1979.

(3) Peled, A. and Liu, B.,"Digital Signal Processing Theory, Design and Implementation", John Wiley & Sons, 1976.

(4) Terrel, T.J.,"Introduction to Digital Filters", The Macmillan Press LTD, London and Basingtoke, First Edition, Reprinted 1983.

(5) Bellanger, M.,"Digital Processing of Signals, Theory and Practice, John Wiley & Sons, 1984.

(6) Oppenheim, A.V. and Schafer, R.W."Digital Signal Processing", Prentice-Hall, Inc., 1975.

(7) Lynn, P.A.,"An Introduction to the Analysis and Processing of Signals", The Macmillan Press LTD, London and Basingtoke, Second Edition, 1983.

(8) Visuwan, S.,"Design of a Programmable Digital Filter", M.Sc. Thesis, 1973.

(9) Kuo, F.F. and Kaiser, J.F.,"System Analysis by Digital Computer", Wiley, 1967.

(10) Kaunitz, J.,"Adaptive Filtering of Broadband Signals as Applied to Noise Cancelling", Stanford Electronics Lab., Stanford Univ., August 1972, Ph.D. Dissertation.

(11) Widrow, B., Glover, JR., McCool, J.M., Kaunitz, J., Williams, C.S., Hearn, R.H., Zeider, J.R., Dong, E. and Goodline, R.C.,"Adaptive Noise Cancelling: Principles and Applications",

Proceedings of the IEEE, Vol.63, No.12, December 1975, PP. 1692-1716.

(12) Holt, A.G. and Mulholland, P.J.,"A Microprogrammed Adaptive Filter Implementation", The Radio and Electronic Engineer, Vol.53, No.5, May 1983.

(13) Paul, J.E.,"Adaptive Digital Techniques for Audio Noise Cancellation", IEEE, Circuits and Systems Magazine, Vol.1, No.4, 1979, PP.2-7.

(14) Sambur, M.R.,"LMS Adpative Filtering for Enhancing the Quality of Noisy Speech", Proceedings of the 1978, IEEE International Conference on Acoustics, Speech and Signal Processing, PP.610-613.

(15) Sambur, M.R.,"Adptive Noise Cancelling for Speech Signals", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-26, No.5, October 1978, PP.419-423.

(16) Harrison, W.A., Lim, J.S. and Singer, E.,"A New Application of Adaptive Noise Cancellation", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-34, No.1, February 1986, PP.21-27.

(17) Lawrence, R.E. and Kaufman, H.,"The Kalman Filter for the Equalization of a Digital Communications Channel", IEEE Transactions on Communication Technology, Vol.Com-19, No.6, December 1971, PP.137-41.

(18) Nissen, C.W. and William, D.K.,"Adaptive Equalizer for Pulse Transmission", IEEE Transactions on Communication Technology, Vol.Com-18, No.4, August 1970, PP.379-395.

(19) Storius, E.H. and Alexander, S.T.,"Channel Equalization Using adaptive Lattice Algorithm", IEEE Transactions on Communication, Vol.COM-27, No.6, June 1979, PP.899-905.

(20) Gersho, A.,"Adaptive Equalization of Highly Dispersive Channels for Data Transmission", The Bell System Technical Journal, January 1969, PP.55-70.

(21) Ghang, Y.H. and Bershad, N.J.,"Weight Modulation Effects in the Adaptive Line Enhancer", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-32, No.5, October 1984,PP. 1078-1081.

(22) Rickard, J.T. and Zeidler, J.R.,"Second-Order Output Statistics of the Adaptive Line Enhancer", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-27, No.1, February 1979, PP.31-39.

(23) Bershad, N.J. and Feintuch P.L.,"The Recursive Adaptive LMS Filter-Aline Enhancer Application and Analytical Model for the Mean Weight Behaviour", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-28, No.6, December 1980, PP.652-660.

(24) Maragos, P.A., Schafer, R.W. and Mersereau, R.M.,"Two-Dimensional Linear Prediction and its Application to Adpative Predictive Coding Images", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSp-32, No.6, December 1984, PP.1213-1229.

(25) Morgan, D.R. and Craic, S.E.,"Real-Time Adpative Linear Prediction Using the Least Mean Square Gradient Algorithm", IEEE Transactions on acoustics, Speech and Signal Processing, Vol.ASSP-24, No.6, December 1976, PP.495-507.

(26) Howells, P.,"Intermediate Frequency Side-Lobe Canceller", U.S. Patent 3 202 990, August 24, 1965.

(27) Widrow, B. and Hoff, M.Jr.,"Adaptive Switching Circuits", in IRE WESCON COnv. Rec., Pt.4, 1960, PP.96-104.

(28) Koford, J. and Groner, G.,"The Use of an Adpative Threshold Element to Design a Linear Optimal Pattern Classifier", IEEE Transactions on Information Theory, Vol.IT-12,January 1966, PP.42-50.

(29) Gabor, D. Wilby, P.L. and Woodcock, R.,"A Universal Nonlinear Filter Predictor and Simulator Which Optimizes Itself by a Learning Process", Proceedings of the IEEE, Vol.108B, July 1960.

(30) Lucky, R.,"Automatic equalization for Digital Communication", Bell System Technical Journal, Vol.44, April 1965, PP.547-88.

(31) Lucky, R.W., Salez, J., and Weldon, J.R.,"Principles of Data Communication", New York, McGraw-Hill, 1968.

(32) Reed, F.A. and Feintuch, P.L.,"A Comparison of LMS Adaptive Cancellers Implementated in the Frequency Domain and the Time Domain", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-29, No.3, June 1981, PP.770-775.

(33) Feintuch, P.L.,"An Adaptive Recursive LMS Filter", Proceedings of the IEEE, Vol.64, No.11, November 1976, PP.1622-1624.

(34) Johnson, C. and Larimore, M.,"Comments on and Addition to "An adaptive LMS Filter"", Proceedings of the IEEE, Vol.65, No.9, September 1977, PP.1399-1404.

(35) Mikhael, W.B., Wu, F.H. and Kazovsky, L.G.,"Adaptive Filters With Individual Adaptation of Parameters", 1984 IEEE International Symposium on Circuits and Systems, Proceeding, Vol.2, 7-10 May 1984, PP.763-7.

(36) Cowan, C.F.N., Smith, S.G., and Elliott, J.H.,"A Digital Adaptive Filter Using A Memory- Accumulator Architecture: Theory and Realization", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-31, No.3, June 1983,

PP.541-549.

(37) Peled, A. and Liu, B.,"Monolithic Adaptive Equalizer", in Proc. ESSCIRC, September 1982.

(38) Cowan, C.F.N. and Grant, P.M., "Hardware Architectures for Adaptive Processors", The Instituation of Electrical Engineers, Colloquiam on "Adaptive Processing and Biomedical Applications", 24-October 1984, PP.3/1-3/8.

(39) South, C.R.,"An Adaptive Filter in LSI", British Telecom Technol. Journal, Vol.3, Part No.1, January 1985, PP.30-46.

(40) Schmidt, L.A.,"Designing Programmable Digital Filters for LSI Implementation", Hewelt Packard J. (USA), Vol.29, Part No.13, September 1978, PP.15-23.

(41) LSI Digital Filter and Detect Circuit, British Telecom Research Laboratories, March 1982.

(42) Angelo, E.J.,"Digital Signal Processor; A Tutorial Introduction to Digital Filtering", the Bell System Technical Journal, Vol.60, No.7, September 1981, PP.1499-1547.

(43) Murakami, H., Reed, I.S. and Arcese, A.,"Recursive FIR Digital Filter Design Using a Z-Transform on a Finite Ring", IEEE Transactions on Acoustics, Speech And Signal Processing, Vol.ASSP-31, No.5, October 1983, PP.1155-1164.

(44) McWilliam, A. J.,"Roundoff Error Sequence in Fixed-Point arithmetic Digital Processors", Ph.D. Thesis, 1976.

(45) Blichikoff, H.J. and Zevereu, A.I.,"Filtering in the Time and Frequency Domains", John-Wiley & Sons, 1976.

(46) Kuo,"Discrete-Data Control Systems".

(47) Liu, C.L. and Liu, J.W.S.,"Linear Systems Analysis", McGraw-Hill, Inc., 1975.

(48) McGillem, C.D. and Cooper, G.R.,"Continuous and Discrete

Signal and System analysis", Holt, Rinehart and Winston, Inc. 1974.

(49) Kailath, T.,"Linear Systems", Prentic-Hall, Inc., 1980.

(50) Robert, A.G. and Richard, A.R."Signals and Linear Systems", John Wiley and Sons Inc., 1973.

(51) Papoulis, A.,"Circuits and Systems, A Modern Approach", Holt, Rinehart and Winston, Inc., 1980.

(52) Gorgui-Naguib, R.N. and Henein, K.M.,"Digital Filter Design Techniques", Electronic and Wireless World, Vol.89, Part No.1574, November 1984, PP.67-9.

(53) Forsythe, W.,"A New Method for the Computation of Digital Filter Coefficients-Part I", Simulation, Vol.44, Part1, January 1985, PP.23-31.

(54) Rabiner, L.R. et al.,"Terminology in Digital Signal Processing", IEEE Transactions on Audio and Electroac, Vol.AU-20, December 1972, PP.322-37.

(55) Hwang, S.Y.,"Realizion of Canonical Digital Networks", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-22, No.1, February 1974, PP.27-38.

(56) Constantinides, A.G.,"Frequency Transformations for Digital Filters", Electrnics Letters, Vol.3, No.11, November 1967, PP.487-489.

(57) Constantinides, A.G.,"Frequency Transformations for Digital Filters", Electronics Letters, Vol.4, April 1968, PP. 115-116.

(58) Kwan, H.K.,"On the Problem of Designing IIR Digital Filters with Short Coefficient Word Lengths", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-27, No.6, December 1979, PP.620-624.

(59) Abu-El-Haija, A.I. and Peterson, A.M.,"An Approach to

Eliminate Roundoff Errors in Digital Filters", IEEE transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-27, No.2, April 1979, PP.195-198.

(60) Mitra, S., Hirano, K. and Sakaguchi, H.,"a Simple Method of Computing the Input Quantization and Multiplication Roundoff Errors in a Digital Filter", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-22, No.5, October 1974, PP. 326-329.

(61) Liu, B.,"Effect of Finite Word-Length on the Accuracy of Digital Filters- A Review", IEEE transactions on Circuit Theory, Vol.CT-18, No.6, November 1971, PP.670-677.

(62) Barnes, C.W.,"Computationally Efficient Second-Order Digital Filter Sections with Low Roundoff Noise Gain", IEEE Transactions on Circuits and Systems, Vol.CAS-31, No.10, October 1984, PP.841-847.

(63) Bamor, B.W. and Hung, J.C.,"Minimum Roundoff Noise Digital Filters with Some Power-of-Two Coefficients", IEEE Transactions on Circuits and Systems, Vol.CAS-31, No.10, October 1984, PP.833-840.

(64) Meron, P., Sekey, A.A. and Zeheb, E.,"Design Method for Stable Second-Order Digital Filters", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-22, No.3, June 1974, PP. 196-202.

(65) Knowles, J. B. and Olcayto, E.M.,"Coefficient Accuracy and Digital Filter Response", IEEE Transactions on Circuit Theory, Vol.CT-15, No.1, March 1968, PP.31-41.

(66) Kieburtz, R.B.,"Rounding and Truncation Limit Cycles in a Recursive Digital Filter", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-22, No.1, February

1974, PP.73.

(67) Munson, D.C., Mullis, C.T. and Roberts, R.A.,"Maximum Amplitude Zero-Limit Cycles in Digital Filters", IEEE Transactions on Circuits and Systems, Vol.CAS-31, No.3, March 1984, PP.266-275.

(68) Mills, W.L. et al.,"Digital Filter Realizations without Overflow Oscillations", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-26, No.4, August 1978, PP.334-338.

(69) Cowan, C.F.N, and Grant, P.M.,"Adaptive Filters", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.

(70) Ferrara, E.R.,"Fast Implementation of LMS Adaptive Filters", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-28, No.4, August 1980, PP.475.

(71) Moscner, J.L.,"Adaptive Filter with Clipped Input Data", SEL-70-053 (TR, No.6796-1), Stanford Electronics Laboratories, Stanfo rd, Calif., June 1970.

(72) Deivasigamani, L.,"A Fast Clipped-Data LMS Algorithm", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-30, No.4, August 1982, PP.648-649.

(73) Fridlander, B.,"System Identification Techniques for Adaptive Noise Cancelling", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-30, No.5, October 1982, PP.699-708.

(74) Widrow, B.,"Adaptive Filters I: Fundamentals", Stanford Electrnics Lab, Stanford Univ., Rep. SU-SEL-66-126, December 1966.

(75) Wei, C.H. and Lou, J.J.,"Multimemory block structure for implementing a digital adaptive filter using distributed

arithmetic", IEE Proceedings-G, Electronics Circuits and Systems, Vol.133, Part G, February 1986, PP.19-26.

(76) McCool, J.M. and Widrow, B.,"Principles and Applications of Adaptive Filters: A Tutorial Review", IEEE International Symposium on Circuits and Systems, 80CH1564-4, Vol.3 of 3, 1980, PP.1143-1157.

(77) Widrow, B., McCool, J.M., Larimore, M.G. and Johnson, C.R.,"Stationary and Non-Stationary Learning Characteristics of the LMS Adaptive Filter", Proceedings of the IEEE, Vol.64, No.8, August 1976, PP.1151-1162.

(78) Tanik, N.T. and Yucel, M.D.,"Conditions for the Convergence of the LMS Algorithm with Gaussian Inputs".

(79) Harris, R.W., Chabries, D.M. and Bishop, F.A.,"A Variable Step (VS) Adaptive Filter Algorithm", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-34, No.2, April 1986, PP.309-316.

(80) Kikuchi, A., Sigeru, O. and Soeda, T."Applications of Adaptive Digital Filtering to the Data processing for the Environmental System", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-27, No.6, December 1979, PP.790-803.

(81) Chen, Ch.T.,"On Digital Wiener Filters", Proceedings of the IEEE, December 1976, PP.1736-1737.

(82) Hill, P.D. and Mikhael, W.B.,"Real Implementation of a Variable Stepsize Adaptive Algorithm", Proceedings of the twenty-seventh Midwest Symposium on Circuits and Systems, Vol.1, 11-12 June 1984, PP.181-184.

(83) White, S.A.,"An Adaptive Recursive Digital Filter", Proceeding 9th Annual Asilomar Conference on Circuits, Systems and Computers, Nov. 1975, PP.21-25.

(84) Stearns, S.D. and Elliot, G.R.,"On Adaptive Recursive Filtering", Proceedings 10th Asilomar Conference on Circuits, Systems and Computers, Nov. 1976, PP.5-11.

(85) Caraiscos, Ch. and Liu, B.,"A Roundoff Error Analysis of the LMS Adpative Algorithm", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol.ASSP-32, No.1, February 1984, PP.34-41.

(86) Boland, F.H. and Normile, J.P.,"Quantization and Truncation Effects in the Design of Adaptive Digital Filters", Proceedings of ICASSP 82, IEEE International Conference on Acoustics, Speech and Signal Processing, Vol.1, 3-5 May 1982, PP.64-68.

(87) Konstantinous, Ki; et al.,"Calculation of the Quantization Noise in an Adaptive Filter which Minimize the Mean Square Error", Izvestya VUZ. Radioelektronika, Vol.25, No.1, 1982, PP.31-37.

(88) Widrow, B. and Walach, E.,"On the Statistical Efficiency of the LMS Algorithm with Non-Stationary Inputs", IEEE Transactions on Information Theory, Vol.IT-30, No.2, March 1984, PP.211-221.

(89) Digital Signal Processing with the FAD, Application Notes", British Telecom Research Laboratories.

(90) Challener, P.,"(FAD)- Flexibility in Digital Signal Processing", Micropcessors and Microsystems, Vol.7, no.10, December 1983.

(91) Gaonkar, R.S.,"Microprocessor Architecture, Programming and Applications with the 8085/8080A", Charls E. Merrill Publishing Company, Bell & Howell Company, 1984.

(92) Brey, B.B.,"Microprocessor/Hardware Interfacing and

Applications", Charls E. Merrill Company, Bell & Howell Company, 1984.

(93) Kane, G., Hawkins, D. and Leventhal, L.,"68000 assembly Language Programming", Osborne/McGraw-Hill, 1981.

(94) Clements, A.,"The 68000 and its Interface", Microprocessors and Microsystems, Vol.8, No.7, September 1984, PP.324-337.

(95) 68000 PROFI KIT Manual, First Edition, April 1982.

(96) The TTL Data Book for Design Engineers, The Engineering Staff of Texas Instruments Components Group, Vol.1, 1980 and 1982.

(97) Greenfield, J.D.,"Practical Digital Design Using ICs",John Wiley & Sons, Second Edition 1983.

(98) Dempsey,"Basic Digital Electronics with MSI Applications".

(99) Holt, C.A.,"Electronic Circuits", Digital and Analogue",

(100) Clayton, G.B.,"Data Converters", The MacMillan Press LTD, 1983.

(101) ADC Data Sheet.

(102) Horouitz, P. and Hill, W.,"The Art of Electronics", Cambridge University Press, 1983.

(103) DAC Data Sheet.

APPENDIX A-1

```
*************************************************************************
*                                                                       *
*     PASCAL program for computing the transfer function magnitude      *
*          of the recursive LMS adaptive filter, with 16 non-           *
*               recursive and 15 recursive coefficients                 *
*                                                                       *
*************************************************************************

         program transfer (nrcoeff2,rcoeff2,output) ;

         const
               count = 128 ;                             * number of angles *
               n = 16                                    * number of recursive (including,b0,)
                                                           and non-recursive coefficients *
         type
                  nrcoeff1 = array[1..n] of real ;
                  rcoeff1  = array[1..n] of real ;
         var
             i,j,m : integer ;
             x : real ;                                  * angle's value *
             nrcoeff2,rcoeff2 : text ;                   * input files contained coefficients *
             zeros    : real ;                           *the numerator (X(ej t),refer to section 2.9
             poles    : real ;                           * the denomirator (Y(ejwt) *
             response : real                             * transfer function magnitude, |G(ej t)|
        begin
             reset (nrcoeff2,'unit=scards') ;           * open input files nrcoeff2 and rcoeff2 *
             reset (rcoeff2,'unit=0') ;
               for   i:=1    to    count   do

*************************************************************************
*                                                                       *
*              compute the transfer function magnitude for              *
*                  points (angles) in range of o to π                   *
*                                                                       *
*************************************************************************

                 begin
                   m := n-1 ;
                   zeros :=0.0 ;                         * initialize accumulators *
                   poles :=0.0 ;
                   response :=0.0 ;
                   x := (3.141592*i)/count ;
                   for   j :=1   to   n  do
                     begin
                       read (nrcoeff2,a[j]) ;            * read non-recursive and recursive
                       read (rcoeff2,b[j]) ;               coefficients one by one *
                       zeros := zeros+(a[j]*cos(m*x)) ; * compute X(ej t) *
                       poles := poles+(b[j]*cos(m*x)) ; * compute Y(ej t) *
                               m := m-1 ;
                     end ;
                   response := zeros/poles ;             * compute G(ej t) *
                   response := abs(response) ;           * compute |G(ej t)| *
                   writeln (x,response) ;
                 end
             end.
```

APPENDIX A-2

```
************************************************************************
*                                                                      *
*   PASCAL program for implementing the recursive LMS adaptive filter, *
*        with 16 non-recursive and 15 recursive coefficients, in       *
*          accordance with the type discussed in section 4.3.1         *
*                                                                      *
************************************************************************

          program filter (data1,data2,output) ;

          const k1 = 7.2e-06 ;                    * convergence factors μ1 and μ2 *
                k2 = 3.6e-06 ;
                p = 17 ;
                q = 16 ;
                r = 15 ;

          type
               signal    = array [1..p] of real ;  * filter input samples *
               filterout = array [1..p] of real ;  * filter output samples *
               nrcoeff   = array [1..q] of real ;  * non-recursive coefficients array *
               rcoeff    = array [1..r] of real ;  * recursive coefficients array *

          var
               atot,                               * filter's linear difference equation
               btot,                                 accumulators *
               e,u,w,                              * filter error output *
               xe,de : real ;                      *filter input and desired response input samples*
               i,j,k,f,m,                          * time indices *
               count : integer ;                   * number of iterations *
              x : signal ;
              y : filterout ;
              a : nrcoeff ;
              b : rcoeff ;
               data1,                              * input files contained the filter input and the
               data2: text ;                         desired response input samples respectively *
          begin
            reset (data1,'unit=scards') ;          * open input files data1 and data2 *
            reset (data2,'unit=0') ;

************************************************************************
*                                                                      *
*        initialize the previous arrays to prevent the overflow        *
*                                                                      *
************************************************************************

           for  i :=1   to   p   do
             begin
               x[i] := 0.0 ;
               y[i] := 0.0 ;
             end ;
           for  i :=1   to   q   do
             begin
               a[i] := 0.0 ;
             end ;
           for  i :=1   to   r   d0
             begin
          b[i] := 0.0 ;
        end ;
      j := p ;
      e := 0.0 ;
      u := 0.0 ;
      w := 0.0 ;
      count := 20000 ;
      for i :=1   to   count  do
        begin
          read (data1,xe) ;                * read filter input and desired response
          read (data2,de) ;                  input samples contained in data1 and data2 *
          x[j] := xe ;                     * the newest sample is the most
          y[j-1] := y[j] ;                   recent sample has been read *
          w := 2*k1*e ;                    * 2 μ1 e *
          u := 2*k2*e ;                    * 2 μ2 e *

*************************************************************************
*                                                                       *
*     update 16 non-recursive and 15 recursive coefficients every       *
*        sampling period and write them back in their previous          *
*              locations in nrcoeff and rcoeff arrays                   *
*                                                                       *
*************************************************************************

          for  k:=1   to   q    do
            begin
              a[k] := (a[k]+w*x[j-k]) ;
              writeln (a[k]:10) ;
            end ;
          for  f:=1   to   r    do
            begin
              b[f] := (b[f]+u*y[j-1-f]) ;
              writeln (b[f]:10) ;
            end ;
          atot := 0.0 ;                    * clear accumulators *
          btot := 0.0 ;

****************************************************************************************************

          for  k :=1   to  q   do

            begin
              atot := atot+(a[k]+x[j+1-k]) ;
            end ;

****************************************************************************************************

          for  f :=1   to  r   do

            begin
              btot := btot +(b[f]+y[j-f]) ;
            end ;

****************************************************************************************************

          y[j] := (atot-btot)/8 ;          * compute the present filter output *
          e := de-y[j] ;                   * compute filter error output *
```

$$\text{atot} = \sum_{k=1}^{q} a_k x_{j+1-k}$$

$$\text{btot} = \sum_{f=1}^{r} b_f y_{j-f}$$

```
***********************************************************************
*                                                                     *
*    shift the previous filter input and output samples one location  *
*       and save them in signal and filterout arrays for the next     *
*            filtering and coefficients updating operations           *
*                                                                     *
***********************************************************************

               for  m:=1  to   p   do
                 begin
                   x[m] := x[m+1] ;
                 end ;
               for m :=1  to   r   do
                 begin
                   y[m] := y[m+1] ;
                 end ;
             end
          end.
```

APPENDIX A-3

```
***********************************************************************
*                                                                     *
*     PASCAL program for implementing the LMS adaptive filter, with   *
*       16 non-recursive and 15 recursive coefficients, according     *
*                to the type discussed in section 4.3.2               *
*                                                                     *
***********************************************************************

 program filter (data1,data2,output) ;

 const   k1 = 7.2e-06 ;                 * convergence factors μ1 and μ2 *
         k2 = 3.6e-06;
         n = 129 ;                      * coefficients updating operation is
         t = 16 ;                         accomplished every n sampling periods *
         r = 15 ;
         z = 17 ;

 type
      signal    = array [1..t] of real ; * filter input samples *
      filterout = array [1..t] of real ; * filter output samples *
      nrcoeff   = array [1..t] of real ; * array of 16 updated non-recursive coefficients *
      rcoeff    = array [1..r] of real ; * array of 15 updated recursive coefficients *
      savein    = array [1..z] of real ; * filter input and output
      saveout   = array [1..z] of real ;   samples delayed by n *
 var
      atot,btot,                         * filter's linear difference equation accumulators *
      u,w,e,                             * filter error output *
      xe,de,des : real ;                 * filter input and desired response input samples *
      i,j,f,k,m,                         * time indices *
      s,h,
      count: integer ;                   * number of iterations *

     x: signal ;
     y: out ;
     a: nrcoeff ;
     b: rcoeff ;
     p: savein ;
     q: saveout ;
     data1,                              * filter input and desired response
     data2 :text ;                         input samples input files *
 begin

***********************************************************************
*                                                                     *
*      initialize the previous arrays at zero to avoid the overflow   *
*                                                                     *
***********************************************************************

  for     i:=1     to    t    do
    begin
      x[i] := 0.0 ;
      y[i] := 0.0 ;
      a[i] := 0.0 ;
    end ;
  for     i:=1     to    r    do
    begin
```

```
      b[i] := 0.0 ;
    end ;
  for     i:=1     to    z    do
    begin
      p[i] :=0.0 ;
      q[i] :=0.0 ;
    end ;
  e := 0.0 ;                              * set e and the associated parameters
  u := 0.0 ;                                u & w to zero for the first
  w := 0.0 ;                                updating period *
  s := 1 ;
  h := t ;
  j := t ;
  count := 520000 ;
  reset (data1,'unit=scards') ;           * open input file data1 *
  reset (data2,'unit=0') ;                * open input file data2 *
  for    i:=1    to      count   do
    begin
      if   s<>n    then
        begin
          read (data1,xe) ;               * read one filter input and desired
          read (data2,de) ;                 response input every sampling period *
          x[j] := xe ;
          y[j-1] := y[j] ;
          p[h] := xe ;                    * save one filter input and filter output sample every n
          q[h-1] := y[j] ;                  sampling period for the coefficients updating operation *
      if    h:=t   then
          de := des ;
          atot := 0.0 ;                   * clear accumulators *
          btot := 0.0 ;

**************************************************************************************************

          for     k :=1       to     t   do

            begin
              atot := atot+(a[k]*x[j+1-k]) ;
            end ;
```

$$atot = \sum_{k=1}^{t} a_k x_{j+1-k}$$

```
***************************************************************************************************

          for      f:=1       to     r   do

            begin
              btot := btot+(b[f]*y[j-f]) ;
            end ;
```

$$btot = \sum_{f=1}^{r} b_f y_{j-f}$$

```
**************************************************************************
*                                                                        *
*      shift filter input and filter output samples one location,        *
*         and save them in their previous locations, occupying           *
*            the most recent samples in locations 16 and 15              *
*             of arrays signal and filterout respectively                *
*                                                                        *
**************************************************************************

          for    m:=1       to     r    do
            begin
              x[m] := x[m+1] ;
            end ;
          for   m :=1      to   r-1   do
            begin
              y[m] := y[m+1] ;
            end ;
          y[j] := (atot-btot)/8 ;          * compute filter output *
          s := s+1 ;                       * next sampling point *
          h := z ;                         * one new input and output samples are applied to
        end                                  the updating operation every n sampling points *
      else
        begin
          j := t ;
          e := des - g[j-1] ;              * compute filter error output *
          w := 2*k1*e ;                    * 2 $\mu_1$ e *
          u := 2*k2*e ;                    * 2 $\mu_2$ e *

**************************************************************************
*                                                                        *
*    update 16 non-recursive and 15 recursive coefficients every n       *
*       sampling periods and write them back in their previous           *
*              locations in nrcoeff and rcoeff arrays                    *
*                                                                        *
**************************************************************************

          for    k:=1     to    t    do
            begin
              a[k] := a[k]+(w*p[j+1-k]) ;
            end ;
          for   f :=1     to    r    do
            begin
              b[f] := b[f]+(u*q[j-f]) ;
            end ;

**************************************************************************
*                                                                        *
*      shift filter input and filter output samples delayed by n,        *
*        used in the coefficients updating operation, and save           *
*           them in arrays savein and saveout respectively               *
*                                                                        *
**************************************************************************

          for   m :=1     to    r    do
            begin
             p[m] := p[m+1] ;
            end ;
          for   m :=1     to   r-1   do
            begin
             q[m] := q[m+1] ;
            end ;
          s := 1 ;
          h := t ;
        end ;
    end
 end.
```

APPENDIX A-4

```
************************************************************************
*                                                                      *
*    PASCAL program for implementing a recursive LMS adaptive filter   *
*          with 16 non-recursive and 15 recursive coefficients         *
*           according to the type discussed in section 4.3.3           *
*                                                                      *
************************************************************************

program filter (data1,data2,output) ;

const
      k1 = 7.2e-06 ;                       * convergence factors μ1 and μ2 *
      k2 = 3.6e-06 ;
      n := 129 ;
      t := 16 ;
      r := 15 ;                            * update the coefficients every n sampling periods *

type
      signal    = array [1..t]  of real ; * filter input samples array *
      filterout = array [1..t]  of real ; * filter output samples array *
      savein    = array [1..40] of real ; * filter input and output samples,
      saveout   = array [1..40] of real ;   used to update the coefficients *
      nrcoeff   = array [1..t]  of real ; * updated non-recursive coefficients array *
      rcoeff    = array [1..r]  of real ; * updated recursive coefficients array *

var
      atot,                                * storages for the implemented linear difference
      btot,                                  equation of the adaptive filter *
      u,w,e,                               * filter error output *
      xe,de: real ;                        * filter and desired response inputs samples *
      i,j,k,f,m,                           * time indicies *
      h,s: integer ;
    x : signal ;
    y : filterout ;
    p : savein ;
    q : saveout ;
    a : nrcoeff ;
    b : rcoeff ;
    data1, data2 : text ;                  * input files *
begin
  reset (data1,'unit=scards') ;            * open the input files data1 and data2 *
  reset (data2,'unit=0') ;

************************************************************************
*                                                                      *
*    initialize the previous arrays at zero to prevent the overflow   *
*                                                                      *
************************************************************************

      for  i :=1   to  t  do
    begin
      a[i] := 0.0 ;
      x[i] := 0.0 ;
      y[i] := 0.0 ;
      for  i :=1   to  r  do
```

```
        begin
          b[i] :=0.0 ;
        end ;
      for  i :=1   to  40  do
    begin
      p[i] := 0.0;
      q[i] := 0.0 ;
    end ;
      e := 0.0 ;
  u := 0.0 ;
  w := 0.0 ;
      s := 1 ;                              * initialize the number of sampling period at one *
  j := t ;                                  * location of the most recent filter input and
  h := t ;                                    filter output samples *
  count := 192000 ;
  for  i :=1  to   count  do
    begin
      if s<>n  then
        begin
          read (data1,xe) ;                 * read the filter input and the desired
          read (data2,de) ;                   response input samples one by one *
          x[j] := xe ;
          y[j-1] := y[j] ;
          p[h] := xe ;
          if h =1  then                     * to avoid the case of zero or minus elements
          h := 40 ;                           orders in arrays savein and saveout *
          q[h-1] := y[j] ;
          atot := 0.0 ;                     * clear accumulators *
          btot := 0.0 ;

*****************************************************************************************************

          for  k :=1  to  t   do

            begin
              atot := atot+(a[k]*x[j+1-k]) ;
            end ;
```

$$\mathrm{atot} = \sum_{k=1}^{t} a_k x_{j+1-k}$$

```
*****************************************************************************************************

         for  f :=1   to  r   do

            begin
              btot := btot+(b[f]*y[j-f]) ;
            end ;
```

$$\mathrm{btot} = \sum_{f=1}^{r} b_f y_{j-f}$$

```
**************************************************************************
*                                                                        *
*          shift filter input and  filter output samples and             *
*              save them in the one dimensional arrays                   *
*                 signal and filterout respectively                      *
*                                                                        *
**************************************************************************

         for  m :=1   to  r   do
            begin
              x[m] := x[m+1]
            end ;
         for  m :=1  to  r-1  do
            begin
            y[m] := y[m+1] ;
          end ;
        y[j] := (atot-btot)/8 ;          * compute filter output and scale it down by 8 *
        e := de-y[j] ;                   * compute filter error output *
        s := s+1 ;                       * get the next input sample *
        h := h-1 ;                       * get the next input sample used in the
      end                                  coefficients updating operation *
    else
      begin
        h := t ;                         * reset (h) to get the first 16 successive samples, every
        j := t ;                           n sampling periods, used in the coefficients
        w := 2*k1*e ;                      updating opration *
        u := 2*k2*e ;


************************************************************************
*                                                                      *
*         update 16 non-recursive and 15 recursive coefficients        *
*            and write them back in their previous locations           *
*               in arrays nrcoeff and rcoeff respectively              *
*                                                                      *
************************************************************************

          for  k :=1  to  t  do
            begin
              a[k] := a[k]+(w*p[h+1-k]) ;
              writeln (a[k]:10) ;
            end ;
          for  f :=1  to  r  do
            begin
              b[f] := b[f]+(u*q[h-f]) ;
              writeln (b[f]:10) ;
            end ;
          s := 1 ;
          h := t ;
        end ;
    end
end.
```

APPENDIX A-5

```
*************************************************************************
*                                                                       *
*            PASCAL program for implementing recursive LMS              *
*               adaptive filter in accordance with the                  *
*                  type discussed in section 4.3.4                      *
*                                                                       *
*************************************************************************

program filter (data1,data2,output) ;

const
      k1 = 7.2e-06 ;                      * convergence factors μ1 and μ2 *
      k2 = 3.6e-06 ;
      n = 129 ;                           * update the coefficients every 128 sampling periods *
      t = 16 ;
      r = 15 ;
      tr = 18 ;

type
      signal    = array [1..t]  of real ; * filter input signal samples *
      filterout = array [1..t]  of real ; * filter output samples *
      nrcoeff   = array [1..t]  of real ; * updated non-recursive coefficients *
      rcoeff    = array [1..r]  of real ; * updated recursive coefficients *
      iport     = array [1..tr] of real ;
      oport     = array [1..tr] of real ;
      savein    = array [1..t]  of real ; * save the 16 input samples and 15 output samples,
      saveout   = array [1..r]  of real ;   used in the coefficients updating operation *
      desired   = array [1..tr] of real ; * the desired response input samples *

var
      atot,                                * the accumulated non-recursive and
      btot,                                  recursive parts of the filter *
      e,u,w,                               * filter error output *
      xe,de: real ;                        * filter input and desired response input samples *
      i,j,k,f,m,                           * time indices *
      count,                               * number of iterations *
      n,c,h,s: integer ;
      x: signal ;
      y: filterout ;
      a: nrcoeff ;
      b: rcoeff ;
      z: iport ;
      g: oport ;
      p: savein ;
      q: saveout ;
      d: desired ;
      data1,data2: text ;
begin
      reset (data1,'unit=scards') ;          * open the input files contained the filter input
      reset (data2,'unit=0') ;                 and desired response input samples *
```

```
************************************************************************
*                                                                      *
*        initialize the previous arrays to prevent the overflow        *
*                                                                      *
************************************************************************

  for      i :=1    to    t    do
    begin
      a[i] := 0.0 ;
      x[i] := 0.0 ;
      y[i] := 0.0 ;
      p[i] := 0.0 ;
    end ;
  for      i :=1     to    r   do
    begin
      b[i] :=0.0 ;
      q[i] :=0.0 ;
    end ;
  for      i :=1     to   tr   do
    begin
      g[i] :=0.0 ;
      z[i] :=0.0 ;
      d[i] :=0.0 ;
    end ;
      e := 0.0 ;                          * set e and the associated
  u := 0.0 ;                                parameters u & w to zero *
  w := 0.0 ;
      s := 1 ;                             * initialize the number of the sampling periods at one *
  h := 2 ;
  j := t ;
  c := 1 ;
  count := 192000 ;
  for     i :=1    to   count    do
    begin
      if    s<>n    then
        begin
          read (data1,xe) ;               * read the filter input and the
          read (data2,de) ;                 desired response input samples *
          x[j] := xe ;                    * the most recent sample is the newest sample *
          y[j-1] := y[j] ;
          z[h] := xe ;
          g[h-1] := y[j] ;
          d[h] := de ;
          atot := 0.0 ;                   * clear accumulators *
          btot := 0.0 ;
***************************************************************************************************

          for    k :=1    to   t   do

           begin
             atot := atot+(a[k]*x[j+1-k])   ;
           end ;
***************************************************************************************************

          for    f :=1    to    r   do

           begin
             btot := btot+(b[f]*y[j-f]) ;
```

$$atot = \sum_{k=1}^{t} a_k x_{j+1-k}$$

$$btot = \sum_{f=1}^{r} b_f y_{j-f}$$

```
            end ;

*************************************************************************
*                                                                       *
*          shift the previous filter input and filter output            *
*             samples and save them in the one dimensional              *
*               arrays signal and filterout respectively                *
*                                                                       *
*************************************************************************

          for   m :=1     to    r    do
            begin
              x[m] := x[m+1] ;
            end ;
          for   m :=1     to   r-1   do
            begin
              y[m] := y[m+1] ;
            end ;
          y[j] := (atot-btot)/8 ;         * compute filter output and scale it down by 8 *
          s := s+1 ;                      * next sampling period *
          h := h+1 ;                      * get the next filter input, desired input, filter output
                                            samples involved in the coefficients updating operation*
        end
      else
        begin
          j := t ;
          c := c+1 ;
          p[j] := z[c] ;                  * save filter input and output samples employed in
          q[j-1] := g[c-1] ;                updating operation in savein and saveout *
          e := d[c]-q[c-1] ;              * compute filter error output *
          w := 2*k1*e ;
          u := 2*k2*e ;

*************************************************************************
*                                                                       *
*         update 16 non-recursive and 15 recursive coefficients         *
*             and write them back into the one dimensional              *
*                arrays nrcoeff and rcoeff respectively                 *
*                                                                       *
*************************************************************************

          for    k :=1    to    t   do
            begin
              a[k] := a[k]+(w*p[j+1-k]) ;
              writeln (a[k]:10) ;
            end ;
          for    f :=1    to    r  do
            begin
              b[f] := b[f]+(u*q[j-f]) ;
              writeln (b[f]:10) ;
            end;

*************************************************************************
*                                                                       *
*       shift and save the filter input, the filter output samples,     *
*             used in the coefficients updating operation in            *
*               the arrays savein and saveout respectively              *
*                                                                       *
*************************************************************************

          for    m :=1    to   r    do
```

```
            begin
              p[m] := p[m+1] ;
            end ;
          for    m :=1    to  r-1   do
            begin
              q[m] := q[m+1] ;
            end ;
          h := tr ;                           * save only the first 16 input and 15 output
          s := 1 ;                              samples for every n*16 sampling period *
        end ;
      if   c=17   then                        * is it the last sampled data? if yes
        begin                                   initialize h and c to get the
          h := 2 ;                              new sampled data *
          c := 1 ;
        end ;
    end
end.
```

APPENDIX A-6

```
************************************************************************
*                                                                      *
*   PASCAL program for illustrating the performance of the 16th order  *
*      recursive LMS adaptive filter at two different frequencies,     *
*          representing the data in floating point arithmetic          *
*                                                                      *
************************************************************************

 program filter (output) ;

 const   k1 = 1.96e-06                    * covergence factors μ1 and μ2 *
         k2 = 1.96e-06 ;
         n = 129 ;
         t = 17 ;
         r = 16 ;
         tr = 18 ;

 type
      signal    = array [1..t]  of real ; * adaptive filter input samples *
      filterout = array [1..t]  of real ; * adaptive filter output samples *
      nrcoeff   = array [1..t]  of real ; * updated non-recursive coefficients array *
      rcoeff    = array [1..r]  of real ; * updated recursive coefficients array *
      iport     = array [1..tr] of real ;
      oport     = array [1..tr] of real ;
      savein    = array [1..t]  of real ; * filter input and output samples used in
      saveout   = array [1..r]  of real ;    the coefficients updating operation *
      desired   = array [1..tr] of real ; * desired response input samples *

 var

      atot,btot,u,w,e,xe,de,
      angle,ang1,ang2,wave1,wave2: real ; * sinusoids angles and the corresponding input samples *
      i,j,f,k,m,s,h,c,count: integer ;
     x: signal ;
     y: filterout ;
     a: nrcoeff ;
     b: rcoeff ;
     z: iport ;
     g : oport ;
     p: savein ;
     q: saveout ;
     d: desired ;
 begin

************************************************************************
*                                                                      *
*                initialize the previous arrays at zero                *
*                                                                      *
************************************************************************

  for     i:=1     to   t   do
    begin
      x[i] := 0.0 ;
      y[i] := 0.0 ;
      a[i] := 0.0 ;
    b[i] := 0.0 ;
    p[i] := 0.0 ;
    q[i] := 0.0 ;
  end ;
 for    i:=1     to  tr  do
  begin
    g[i] := 0.0 ;
    d[i] := 0.0 ;
    z[i] := 0.0 ;
  end ;
 e := 0.0 ;
 u := 0.0 ;
 w := 0.0 ;
 s := 1 ;
 h := 2 ;
 j := t ;
 c := 1 ;
 count := 1900000 ;
 for   i:=1    to      count   do
  begin
    if   s<>n    then
      begin

****************************************************************************************************

        if   (i<950000) and (i>=1)  then
          begin
            angle := 11.25*i ;
            xe := sin((angle*3.141592)/180) ;   generate the filter input and the desired response
            xe := xe*1.5 ;                         input sinusoidal signals in accordance with
            ang1 := 11.25*i ;                         the system discussed in section 4.3.b
            wave1 := sin((ang1*3.141592)/180) ;            at f1 for the first count/2
            ang2 := 5.625*i ;                                iterations and at f2 for
            wave2 := sin((ang2*3.141592)/180) ;                  the last count/2
            de := wave1+wave2 ;                                     iterations
          end ;
        if   (i<1900000) and (i>=950000)  then
          begin
            angle := 5.625*i ;
            xe := sin((angle*3.141592)/180) ;
            xe := 1.5*xe ;
            ang1 := 5.625*i ;
            wave1 := sin((ang1*3.141592)/180) ;
            ang2 := 2.8125*i ;
            wave2 := sin((ang2*3.141592)/180) ;
            de := wave1+wave2 ;
          end ;

****************************************************************************************************

        x[j] := xe ;                    * read the newest filter input, the desired
        y[j-1] := y[j] ;                  response input and filter output
        z[h] := xe ;                      floating point samples *
        g[h-1] := y[j] ;
        d[h] := de ;
        atot := 0.0 ;                   * clear the accumulators atot and btot *
        btot := 0.0 ;
```

```
*************************************************************************************************

          for    k :=1     to    t    do
            begin

              a[1] := 1.0 ;
              atot := atot+((a[k]/8)*x[j+1-k]) ;
            end ;

*************************************************************************************************
```

$$atot = \sum_{k=1}^{t} a_k x_{j+1-k}$$

```
          for     f:=1     to    r   do

            begin
              btot := btot+((b[f]/8)*y[j-f]) ;
            end ;
```

$$btot = \sum_{f=1}^{r} b_f y_{j-f}$$

```
***********************************************************************
*                                                                     *
*         shift the previous input and output samples, used to        *
*            compute the present filter output one location           *
*              and save them in savein and saveout arrays             *
*                                                                     *
***********************************************************************

          for    m :=1    to   r    do
            begin
              x[m] :=x[m+1] ;
            end ;
          for    m :=1    to  r-1   do
            begin
              y[m] :=y[m+1] ;
            end ;
          y[j] :=(atot-btot)/2 ;          * compute filter output and scale it down by 2 *
          e :=de-y[j] ;                   * compute filter error output *
          writeln (i:12,y[j]:12,e:12) ;
          s :=s+1 ;                       * next sampling period *
          h :=h+1 ;                       * get the next sampled data used in the
        end                                 coefficients updating operation *
       else
        begin
          c := c+1 ;                      * get the next sampled data used in
          j := t ;                          the updating operation *
          p[j-1] := z[c] ;
          q[j-1] := g[c-1] ;
          e := d[c]-q[c-1] ;
          w := 2*k1*e ;                   * compute 2 μ1 e and 2 μ2 e *
          u := 2*k2*e ;

***********************************************************************
*                                                                     *
*       update 16 non-recursive and 16 recursive coefficients         *
*         and save them back in their previous locations in           *
*              nrcoeff and rcoeff arrays respectively                 *
*                                                                     *
***********************************************************************

          for  k :=2   to  t  do
            begin
```

```
              a[k] :=a[k]+(w*p[j+1-k]) ;
            end ;
          for  f :=1   to  r   do
            begin
              b[f] := b[f]+(u*q[j-f]) ;
            end ;

*************************************************************************
*                                                                       *
*        shift the filter input and output samples, used in the         *
*          coefficients updating operation, one location and            *
*               save them in arrays savein and saveout                  *
*                                                                       *
*************************************************************************

          for   m :=1   to    r    do
            begin
              p[m] := p[m+1] ;
            end ;
          for   m :=1   to   r-1   do
            begin
              q[m] := q[m+1] ;
            end ;
          s := 1 ;
          h := tr ;                         * reset h and c and read only the first 17 filter
        end ;                                 input and 16 filter output samples for
      if   c=17   then                        every (n*16) sampling periods *
        begin
          h := 2 ;
          c := 1 ;
        end ;
    end
 end.
```

APPENDIX A-7

```
************************************************************************
*                                                                      *
*      PASCAL program for illustrating the recursive LMS adaptive      *
*           filter performance at two different frequencies            *
*                    in fixed point arithmetic                         *
*                                                                      *
************************************************************************

 program filter (output) ;

 const   k1 = 1020033 ;                      * k1 = 2/μ1 & k2 = 2/μ2 *
         k2 = 1020033 ;
         n = 129 ;
         t = 17 ;
         r = 16 ;
         tr = 18 ;

 type
      signal    = array [1..t]  of integer ;
      filterout = array [1..t]  of integer ;
      nrcoeff   = array [1..t]  of integer ;
      rcoeff    = array [1..r]  of integer ;
      iport     = array [1..tr] of integer ;
      oport     = array [1..tr] of integer ;
      savein    = array [1..t]  of integer ;
      saveout   = array [1..r]  of integer ;
      desired   = array [1..tr] of integer ;

 var
      sig,dis,angle,ang1,ang2,wave1,wave2: real ;
      atot,btot,u,e,w,xe,de,i,j,f,k,m,s,h,c,count: integer ;
     p: savein ;
     q: saveout ;
     x: signal ;
     y: filterout ;
     a: nrcoeff ;
     b: rcoeff ;
     z: iport ;
     g: oport ;
     p: savein ;
     q: saveout ;
     d: desired ;
 begin
  for     i:=1     to    t   do
    begin
      x[i] := 0 ;
      y[i] := 0 ;
      a[i] := 0 ;
      b[i] := 0 ;
      p[i] := 0 ;
      q[i] := 0 ;
    end ;
  for     i:=1     to   tr   do
    begin
      z[i] := 0 ;
      g[i] := 0 ;
    d[i] := 0 ;
   end ;
  e := 0 ;
  u := 0 ;
  w := 0 ;
  s := 1 ;
  h := 2 ;
  j := t ;
  c := 1 ;
  count := 1900000 ;
  for    i:=1    to      count   do
   begin
    if   s<>n    then
     begin

**************************************************************************************************

      if   (i<950000) and (i>=1)  then
       begin
        angle := 11.25*i ;
        sig := sin((angle*3.141592)/180) ;
        sig := sig*1.5 ;
        sig := sig*1164 ;
        xe := trunc(sig) ;
        ang1 := 11.25*i ;
        wave1 := sin((ang1*3.141592)/180) ;     generate the filter input and the desired
        ang2 := 5.625*i ;                          response input samples represented
        wave2 := sin((ang2*3.141592)/180) ;           in fixed point arithmetic in
        dis := wave1+wave2 ;                           accordance with the system
        dis := dis*1164 ;                                 discussed in section
        de := trunc(dis) ;                                  4.3.b at f1 and
       end ;                                                 f2 (f2=f1/2)
      if   (i<1900000) and (i>=950000)  then
       begin
        angle := 5.625*i ;
        sig := sin((angle*3.141592)/180) ;
        sig := 1.5*sig ;
        sig := sig*1164 ;
        xe := trunc(sig) ;             * get the integer part of the input samples
        ang1 := 5.625*i ;
        wave1 := sin((ang1*3.141592)/180) ;
        ang2 := 2.8125*i ;
        wave2 := sin((ang2*3.141592)/180) ;
        dis := wave1+wave2 ;
        dis := dis*1164 ;
        de := trunc(dis) ;             * get the integer part of the desired response samples*
       end ;

**************************************************************************************************

      x[j] := xe ;
      y[j-1] := y[j] ;
      z[h] := xe ;
      g[h-1] := y[j] ;
      d[h] := de ;
      atot := 0 ;
      btot := 0 ;
      for    k :=1      to    t   do
       begin
        a[1] := 1 ;
        atot := atot+((a[k] div 8)*x[j+1-k]) ;
```

```
          end ;
        for     f:=1       to    r   do
          begin
            btot := btot+((b[f] div 8)*y[j-f]) ;
          end ;
        for    m:=1      to    r     do
          begin
            x[m] := x[m+1] ;
          end ;
        for    m :=1      to    r-1   do
          begin
            y[m] := y[m+1] ;
          end ;
        y[j] :=(atot-btot) div 8192 ;
        e := de-y[j] ;                          * compute filter output and scale it down by 2,
        writeln (i:12,y[j]:12,e:12) ;
        s := s+1 ;                                and express the coefficients as a fraction of
        h := h+1 ;                                12 bits by dividing the output by 2^12 *
      end
     else
      begin
        j := t ;
        c := c+1 ;
        p[j-1] := z[c] ;
        q[j-1] := g[c-1] ;
        e := d[c]-q[c-1] ;
        for   k :=2    to    t    do
          begin
            w := e*p[j+1-k] ;
            w := w div k1 ;
            a[k] :=a[k]+w ;                     * a_{k+1} = a_k + 2*μ_1*x_k*e_k *
          end ;
        for   f :=1    to    r    do
          begin
            u :=e*q[j-f] ;
            u :=u div k2 ;
            b[f] :=b[f]+u ;                     * b_{f+1} = b_f + 2*μ_2*y_f*e_f*
          end ;
        for   m:=1    to    r    do
          begin
            p[m] :=p[m+1] ;
          end ;
        for   m :=1   to   r-1   do
          begin
            q[m] :=q[m+1] ;
          end ;
        s := 1 ;
        h := tr ;
      end ;
    if   c=t   then
      begin
        h := 2 ;
        c := 1 ;
      end ;
  end
end.
```

APPENDIX B

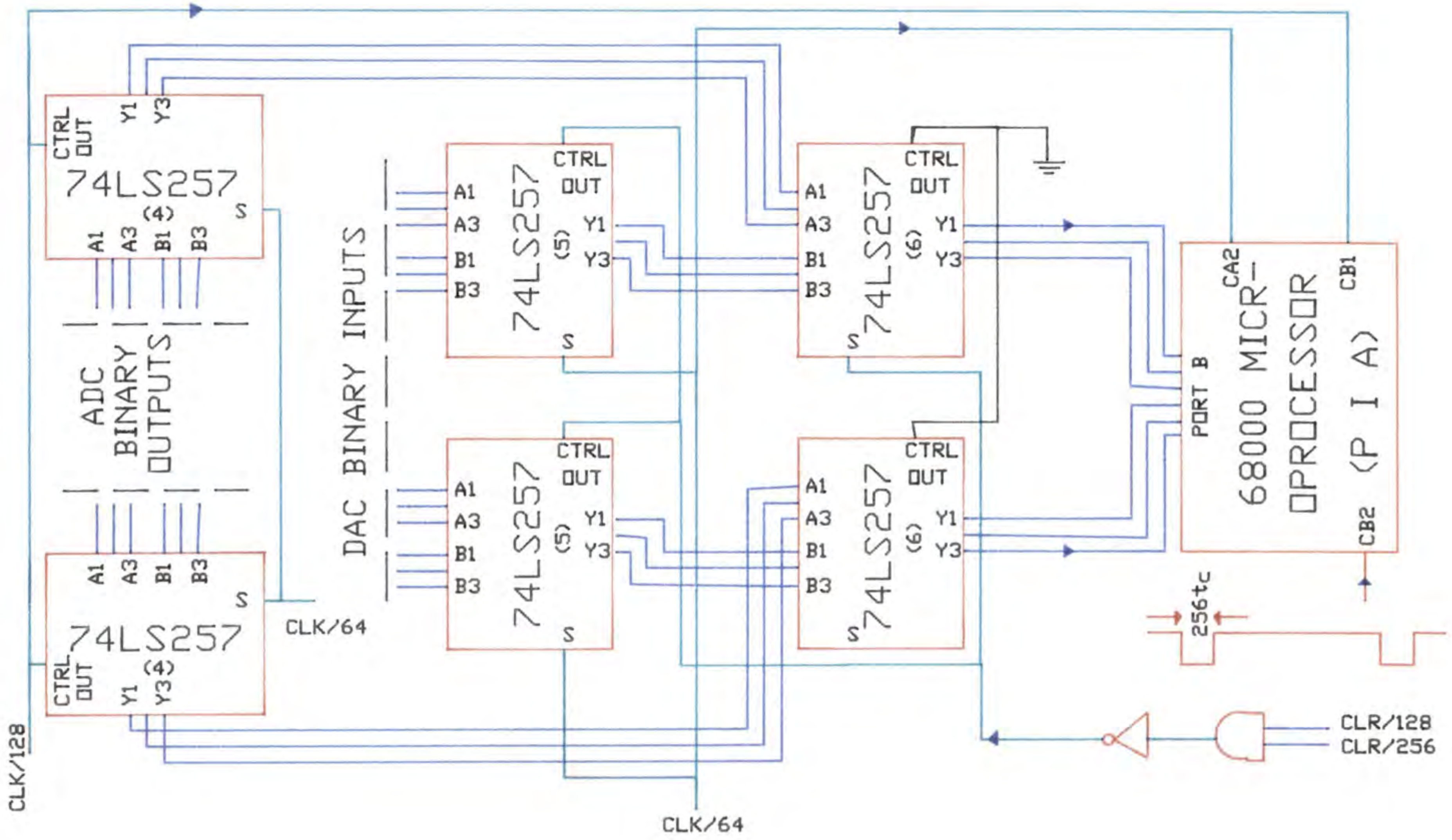


Fig. B-1 The Data Multiplexer Circuit Diagram

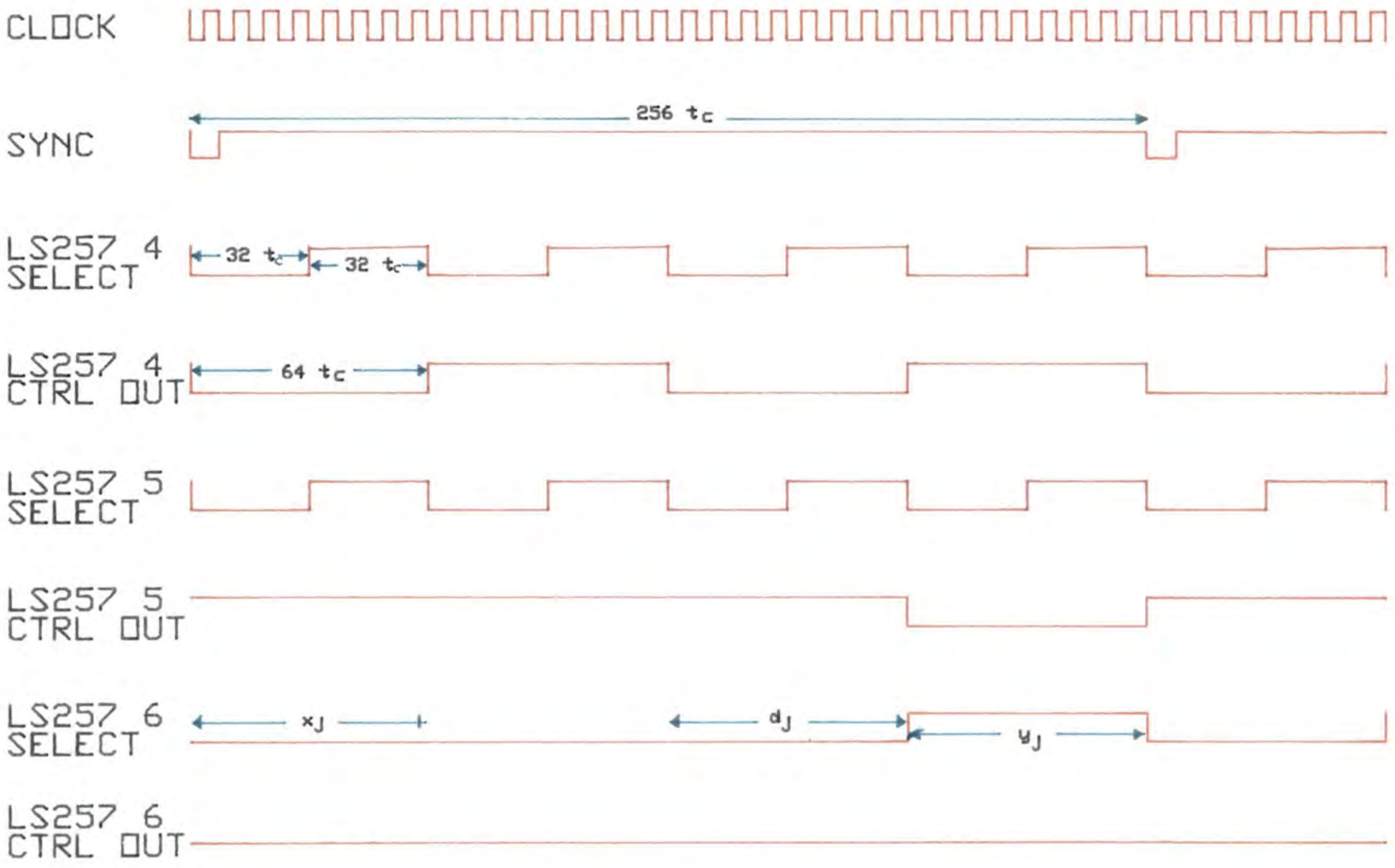


Fig. B-2 The Data Multiplexer Timing Diagram

APPENDIX C

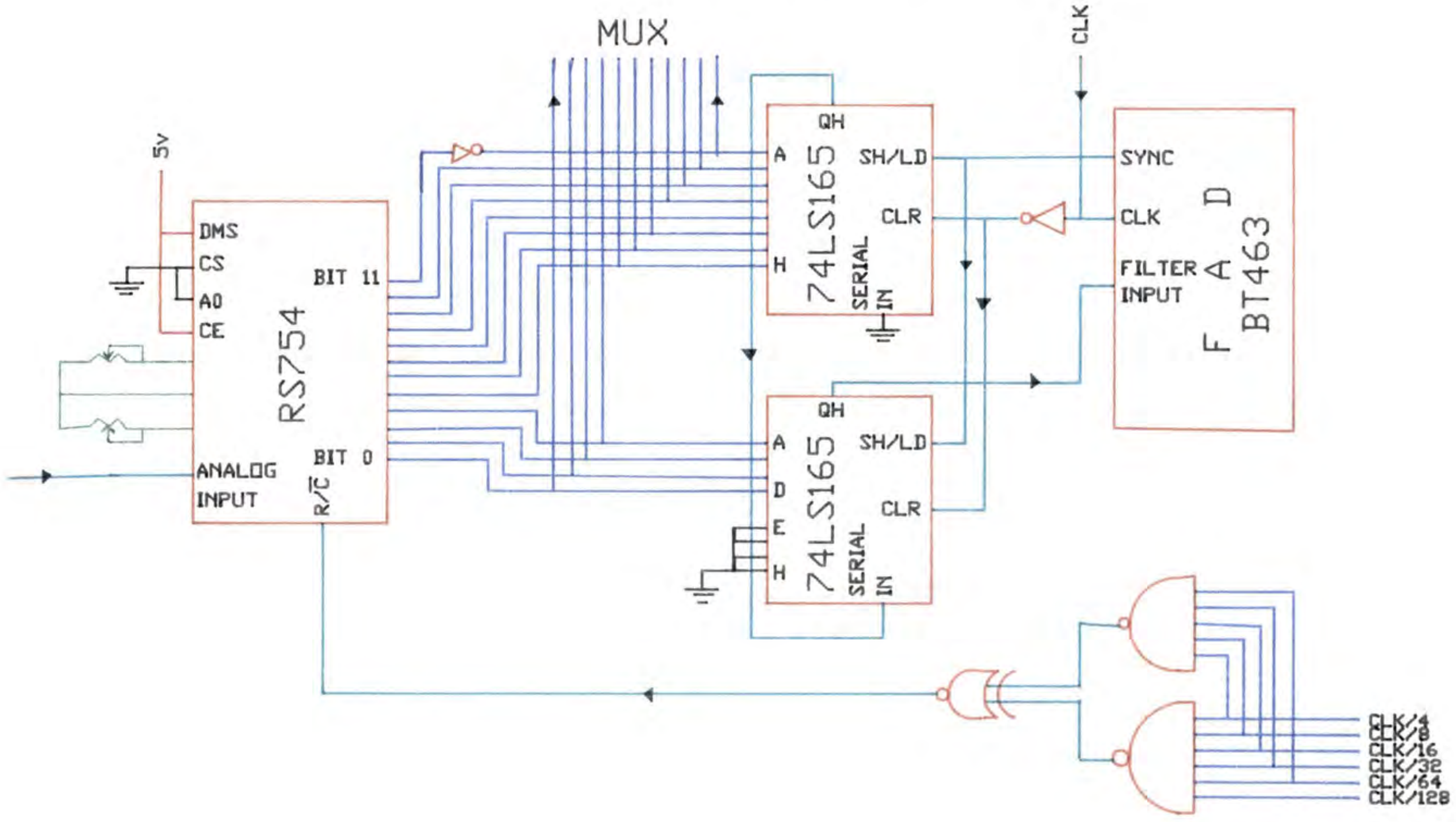


Fig. C-1 Parallel to Serial Conversion Diagram of the Filter Input

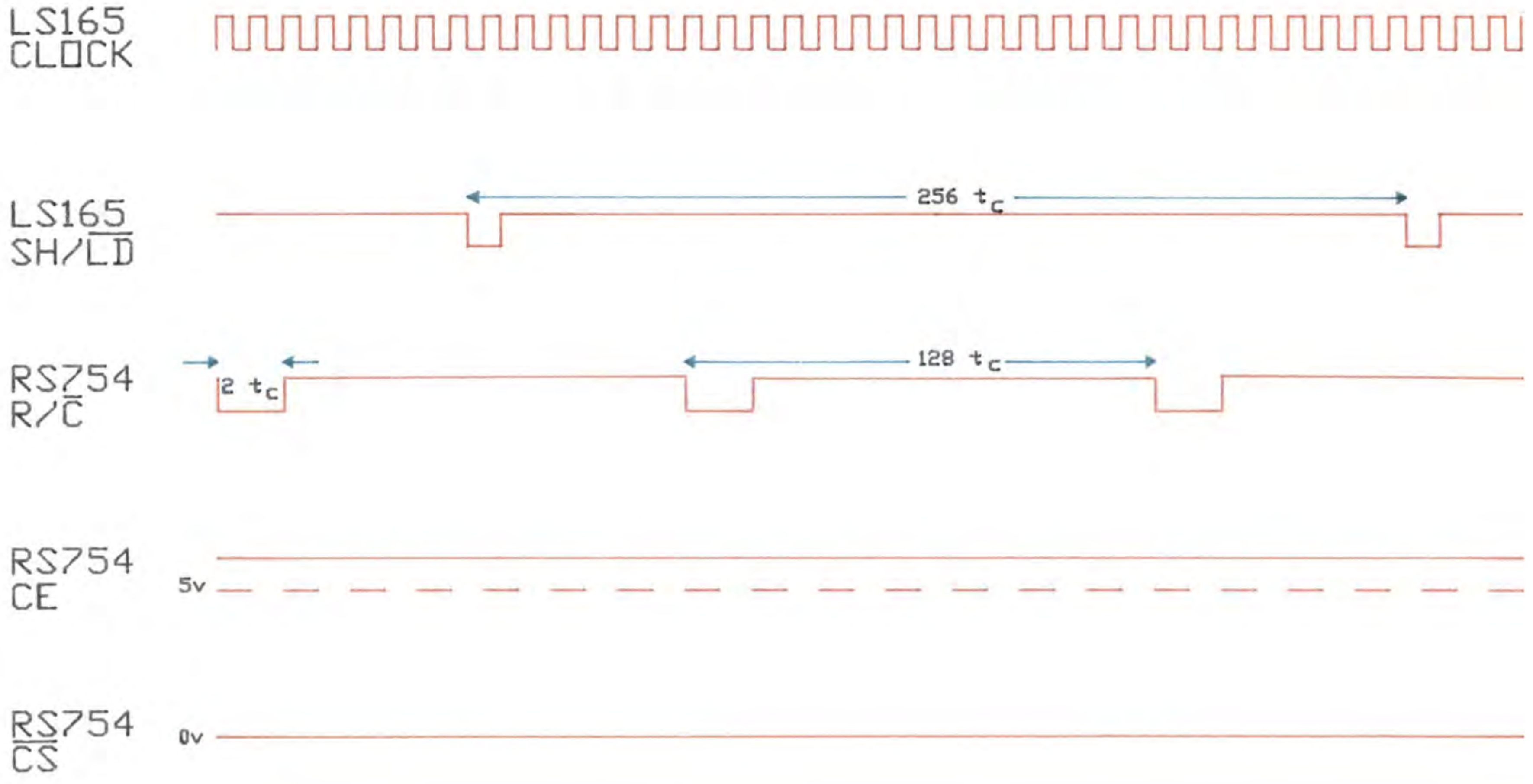


Fig. C-2 Parallel to Serial Conversion Timing Diagram of the Filter Input

APPENDIX D

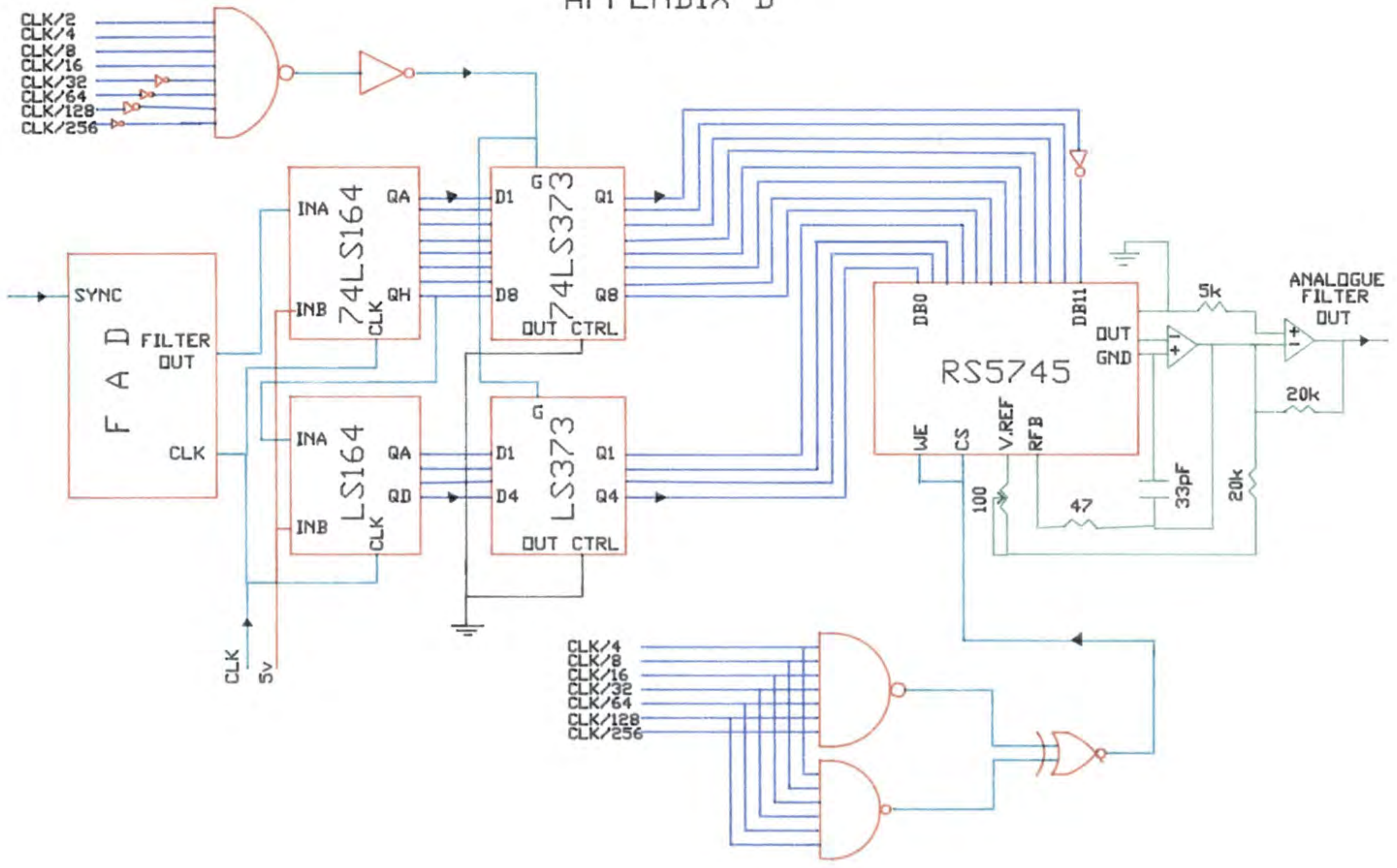


Fig. D-1 Circuit Diagram for Producing the Analogue Filter Output

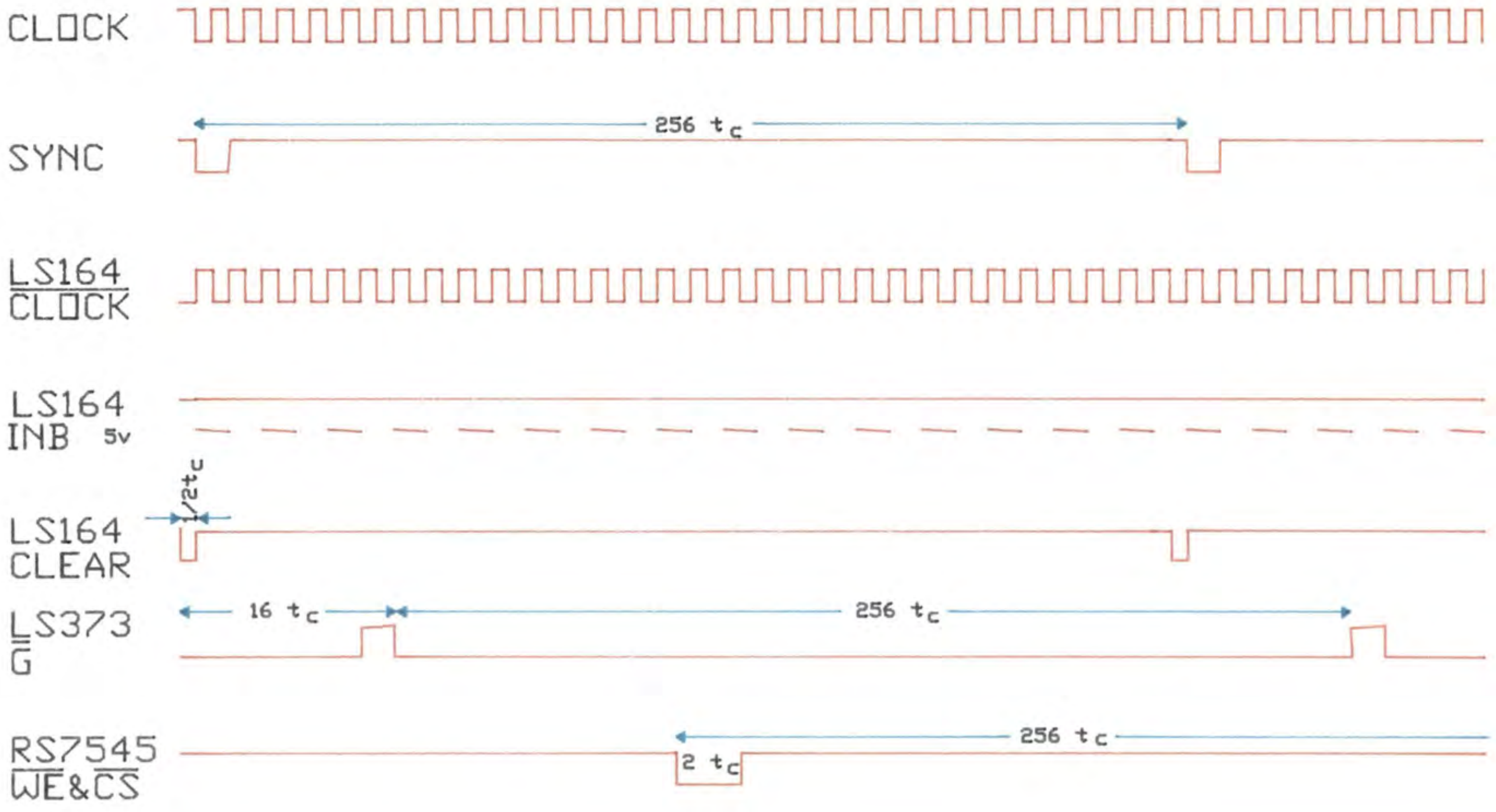


Fig. D-2 Timing Diagram for Producing the Analogue Filter Output

APPENDIX E

```
          ******************************************************
          *                                                    *
          *  68000 ASSEMBLY LANGUAGE PROGRAM FOR UPDATING THE  *
          *    COEFFICIENTS OF THE 16th ORDER RECURSIVE LMS    *
          *                 ADAPTIVE FILTER                    *
          *                                                    *
          ******************************************************

PROGRAM      : EQU  $4000 ;
PIAA         : EQU  $05CEF1 ;  * ADDRESS OF PIA DATA DIRECTION AND DATA REGISTERS A *
CTRLA        : EQU  $05CEF3 ;  * ADDRESS OF PIA CONTROL REGISTER A *
PIAB         : EQU  $05CEF5 ;  * ADDRESS OF PIA DATA DIRECTION AND DATA REGISTERS B *
CTRLB        : EQU  $05CEF7 ;  * ADDRESS OF PIA CONTROL REGISTER B *
INSAMPLE     : EQU  $5000 ;    * STARTING ADDRESS OF ADDRESS REGISTER A7 *
WEIGHT       : EQU  $5300 ;    * STARTING ADDRESS OF MEMORY LOCATIONS USED AS A STORAGE
                                 OF THE UPDATED COEFFICIENTS ADDRESSED BY A5 *
INPUT        : EQU  $5400 ;    * STARTING ADDRESS OF ADDRESS REGISTER A4 *
OUTPUT       : EQU  $5500 ;    * STARTING ADDRESS OF ADDRESS REGISTER A6 *
INSAM        : EQU  $53FE ;
OUTSAM       : EQU  $54FE ;
C            : EQU @$5608 ;
Z            : EQU @$560C ;
CONST1       : EQU @$5610 ;    * TEMPORARY STORAGE OF NRCOEFFS *
CONST2       : EQU @$5614 ;    * TEMPORARY STORAGE OF RCOEFFS
STB1         : EQU @$5618 ;    * STORE μ1 AND μ2 IN MEMORY LOCATIONS
STB2         : EQU @$561C ;      $5618 & $561C RESPECTIVELY *
K            : EQU @$5620 ;

          ******************************************************
          *                                                    *
          *      SET UP DATA DIRECTION, DATA AND CONTROL       *
          *              REGISTERS OF THE PIA                  *
          *                                                    *
          ******************************************************

       ORG  PROGRAM ;               * SPECIFY THE STARTING ADDRESS OF MAIN PROGRAM *

          ********************************************************************************

              MOVEA.L PIAA,A0 ;     * GET PIA BASE ADDRESS *
              MOVEA.L CTRLA,A1 ;    * GET PIA CONTROL REGISTER A ADDRESS *
              MOVEA.L PIAB,A2 ;     * GET PIA DATA DIRECTION AND DATA REGISTERS B ADDRESS *
              MOVEA.L CTRLB,A3;     * GET PIA CONTROL REGISTER B ADDRESS *

          ********************************************************************************

              MOVEA.L INSAMPLE,A7;  * POINTER TO SAMPLED DATA (INPUT, DESIRED
                                      RESPONSE,OUTPUT) BLOCKS START *
              MOVEA.L WEIGHT,A5 ;   * POINTER TO COEFFICIENTS STORAGE START *
              MOVEA.L INPUT,A4 ;    * POINTER TO INPUT SAMPLES START *
              MOVEA.L OUTPUT,A6 ;   * POINTER TO OUTPUT SAMPLES START *
```

```
                    *****************************************************
                    *                                                   *
                    *         INITIALIZE AND SET A & B SIDES TO         *
                    *                   INPUT-OUTPUT                    *
                    *                                                   *
                    *****************************************************

                       CLR.B $0(A1) ;          * CLEAR PIA CONTROL REGISTER (A SIDE) *
                       MOVE.B  #$FF,(A0) ;     * MAKE ALL DATA LINES OUTPUT (A SIDE) *
                       CLR.B $0(A3) ;          * INTIALIZE B SIDE *
                       MOVE.B  #$03,(A2) ;     * DATA LINES 1 & 2 OUTPUTS, 3-8 INPUTS (B SIDE) *
LOOP                 : CLR.L D0 ;
                       CLR.L D1 ;
                       CLR.L D2 ;
                       CLR.L D3 ;
                       CLR.L D4 ;
                       CLR.L D5 ;
                       CLR.L D6 ;
                       CLR.L D7 ;

                    *******************************************************************************

                       MOVE.B  #$14,(A1) ;     * SELECT DATA REGISTER A; CA2 IS TRIGGERED ON LOW TO
                                                 HIGH TRANSITION *
                       MOVE.B  #$14,(A3) ;     *SELECT DATA REGISTER B;CB1 IS TRIGGERED ON HIGH TO LOW
                                                TRANSITION,CB2 IS TRIGGERED ON LOW TO HIGH TRANSITION*
                       MOVE.B (A0),Z ;         * DUMMY READ-CLEARS STATUS BIT IN CONTROL REGISTER A *
                       MOVE.B (A2),Z ;         * DUMMY READ-CLEARS STATUS BITS IN CONTROL REGISTER B *

                    *******************************************************************************

                       MOVE.W  #$10,D4 ;       * NUMBER OF SUCCESSIVE SAMPLES IN EACH BLOCK *

                    *****************************************************
                    *   READ A BLOCK OF SUCCESSIVE SAMPLES OF THE FILTER  *
                    *     INPUT, DESIRED RESPONSE INPUT AND THE FILTER    *
                    *         OUTPUT FROM THE A/D CONVERTER; 6MSBS        *
                    *             FIRST AND THEN THE 6LSBS AND            *
                    *               STORE THEM IN THE MEMORY              *
                    *              LOCATIONS ADDRESSED BY A7              *
                    *****************************************************

HIGH                  : BTST.B  #$06,(A3) ;     * TEST THE STATE OF THE INPUT READY
                        BEQ HIGH ;                   PULSE ON CB2;IF READY DELAY
                        MOVE.B (A0),Z ;                 UNTIL THE NEXT FILTER
                        MOVE.B (A2),Z ;                  INPUT IS AVAILABLE *
LOW                   : BTST.B  #$07,(A3) ;
                        BEQ LOW ;
                        MOVE.B (A2),Z ;

                    *******************************************************************************

DATA                 : MOVE.B (A0),Z ;
                       MOVE.B  #$14,(A3) ;      * CB1 IS TRIGGERED ON HIGH TO LOW TRANSITION *
INPUTM               : BTST.B  #$07,(A3) ;      * TEST STATUS BIT 7 IN CONTROL REGISTER B *
                       BEQ INPUTM ;             * BIT 7=1, GET 6MSBS
                       MOVE.B (A2),(A7)+;         OF THE INPUT SAMPLE *
INPUTL               : BTST.B  #$06,(A1) ;      * TEST STATUS BIT 6 IN CONTROL REGISTER A *
                       BEQ INPUTL ;             * IF BIT 6=1, THEN GET 6LSBS
                       MOVE.B (A2),$20(A7) ;      OF THE INPUT SAMPLE *
DESIREDM             : BTST.B  #$07,(A3) ;      * IF THE STATUS BIT 7 IS SET ON THE HIGH TO LOW
```

```
                BEQ DESIREDM ;            TRANSITION OF THE CB1, THEN GET THE 6MSBS
                MOVE.B (A2),$40(A7) ;     OF THE DESIRED RESPONSE SAMPLE *
                MOVE.B (A0),Z ;         * CLEAR THE STATUS BITS IN CONTROL REGISTER A *
DESIREDL      : BTST.B  #$06,(A1) ;     * IF THE STATUS BIT 6 IS SET ON LOW TO HIGH
                BEQ DESIREDL ;            TRANSITION OF CA2, THEN GET THE 6LSBS
                MOVE.B (A2),$60(A7) ;     OF THE DESIRED RESPONSE SAMPLE *
                MOVE.B  #$06,(A3) ;     * CB1 IS TRIGGERED ON LOW TO HIGH TRANSITION *
OUTPUTM       : BTST.B  #$07,(A3) ;     * GET THE 6MSBS OF THE FILTER OUTPUT SAMPLE *
                BEQ OUTPUTM ;
                MOVE.B (A2),$80(A7);
                MOVE.B (A0),Z ;
OUTPUTL       : BTST.B  #$06,(A1) ;     * GET THE 6LSBS OF THE FILTER OUTPUT SAMPLE *
                BEQ OUTPUTL ;
                MOVE.B (A2),$A0(A7);
                SUBQ.W  #$01,D4 ;       * READ THE NEXT FILTER INPUT, DESIRED
                BNE.L DATA ;              RESPONSE AND FILTER OUTPUT SAMPLES *
                MOVEA.L INSAMPLE,A7 ;   * HAS THE LAST SAMPLED DATA BEEN READ, IF YES THEN
                                          RELOAD THE ADDRESS REGISTER A7 WITH THE FIRST 6MSBS
                                          INPUT SAMPLE'S ADDRESS *
                MOVE.W  #$10,D4 ;       * NUMBER OF SAMPLES IN EACH BLOCK *
                MOVE.W  #$10,K ;

              ******************************************************************************

TOP           : MOVE.B (A2),Z ;         * THE UPDATING OPERATION IS ACCOMPLISHED EVERY 128
                MOVE.B  #$14,(A3) ;               COMPUTATION PERIOD IN ACCORDANCE
LIST          : BTST.B  #$06,(A3) ;                 WITH THE INPUT PULSE ON CB2 *
                BEQ LIST ;

              ******************************************************************************

               MOVE.W D4,K ;            * SAVE THE NUMBER OF SUCCESSIVE SAMPLES *
               MOVE.W  #$0F,D4;

              *******************************************************
              *                                                     *
              *    SET THE SAMPLED DATA IN THE CORRECT POSITIONS    *
              *                                                     *
              *******************************************************

                MOVE.B (A7)+,D0 ;       * LOAD THE DATA REGISTER D0 WITH THE 6MSBS OF THE
                                          INPUT SAMPLE *
                LSL.W  #$06,D0 ;        * ALLOCATE IT IN THE MS BYTE OF D0 *
                MOVE.B $20(A7),D0 ;     * GET THE 6LSBS OF THE INPUT SAMPLE *
                LSR.W  #$02,D0 ;        * SET THE LSB OF THE INPUT SAMPLE IN BIT POSITION 0 IN
                                          D0 *
                LSL.W  #$04,D0 ;        * ALLOCATE THE MSB OF THE INPUT SAMPLE IN BIT
                                          POSITION 15 *
                EXT.L D0 ;              * EXTEND THE SIGN BIT TO BIT 31 IN D0 *
                LSR.L #$04,D0 ;         * SET THE INPUT SAMPLE IN BIT POSITION 0
                                          THROUGH 11 WITH THE EXTENDED SIGN BIT
                                          OCCUPYING BIT 12 THROUGH BIT 15 *


              ******************************************************************************

RET1          : MOVE.B $40(A7),D1 ;     * EXECUTE THE SAME SEQUENCE
                LSL.W  #$06,D1 ;           OF INSTRUCTIONS FOR THE
                MOVE.B $60(A7),               DESIRED RESPONSE
                LSR.W  #$02,D1 ;                INPUT SAMPLE *
                LSL.W  #$04,D1 ;
```

```
                 EXT.L D1 ;
                 LSR.L  #$04,D1 ;

             ********************************************************************************

RET2           : MOVE.B $80(A7),D2 ;   * EXECUTE THE IDENTICAL SEQUENCE
                 TST.B D2 ;                OF INSTRUCTIONS FOR THE
                 BMI.L NEG3 ;                   FILTER OUTPUT
                 LSL.W  #$06,D2 ;                  SAMPLE *
                 MOVE.B $A0(A7),D2;
                 LSR.W  #$02,D2 ;

             ********************************************************************************

RET3           : MOVE.W D0,-(A4) ;     * SAVE THE INPUT AND OUTPUT SAMPLES IN THE MEMORY
                 MOVE.W D2,-(A6) ;       LOCATIONS ADDRESSED BY A4 AND A6 RESPECTIVELY *
                 SUB.W D2,D1 ;         * COMPUTE THE ERROR OUTPUT OF THE FILTER BY SUBTRACTING
                                         THE FILTER OUTPUT FROM THE DESIRED RESPONSE INPUT *

              *******************************************************
              *                                                     *
              *      UPDAT 16-NON-RECURSIVE AND 16-RECURSIVE        *
              *        COEFFICIENTS AND STORE THEM IN MEMORY        *
              *          LOCATIONS ADDRESSED BY THE ADDRESS         *
              *                REGISTER A5 ACCORDING                *
              *                  TO THE EQUATIONS                   *
              *                                                     *
              *             a_{j+1} = a_j + 2*μ1*X_j*e_j            *
              *                                                     *
              *             b_{j+1} = b_j + 2*μ2*Y_j*e_j            *
              *                                                     *
              *******************************************************

                 MOVE.W  #$08,D6 ;     * LOAD D6 WITH THE NUMBER OF THE SECOND ORDER SECTIONS
                                         CASCADED TO FORM THE 16th ORDER FILTER *
                 CLR.L D5 ;            * STARTING ADDRESS OF THE FIRST SET OF THE RAMS *
START          : NOP ;
                 NOP ;
INCURRENT      : CMPA.L  #$541F,A6 ;   * IS IT THE LAST LOCATION OF THE INPUT OR OUTPUT
SAMPLES ?
                 BGE.L COMP1 ;           IF YES, THEN COMP1 *
NONRECUR       : MOVE.W (A4+,D3 ;      * GET THE NEXT INPUT SAMPLE *
                 MULS.W D1,D3 ;        * MULTIPLY e_j BY x_{j-n+1}*
                 DIVS.W STB1,D3 ;      * SCALE THE RESULT BY μ1 *
                 EXT.L D3 ;
                 DIVS.W STB2,D3 ;
                 ADD.W (A5),D3 ;       * ADD 2 μ1e_j x_{j-n+1} TO THE PREVIOUS UPDATED COEFFICIENT
                                         OF THE SAME ORDER *
                 MOVE.W D3,(A5)+ ;     * SAVE THE NRCOEFF BACK INTO THE SAME MEMORY LOCATION*
                 MOVE.W D3,CONST1 ;    * SAVE THE NRCOEFF IN THE TEMPORARY LOCATION ADDRESSED
                                         BY CONST1 *

              *******************************************************************************

                 MOVE.W (A6)+,D7 ;     * EXECUTE SIMILAR  SEQUENCE
                 MULS.W D1,D7 ;              OF INSTRUCTIONS
                 DIVS.W STB1,D7 ;               FOR y_{j-n} *
                 EXT.L D7 ;
                 DIVS.W STB2,D7 ;
```

```
            ADD.W (A5),D7 ;
            MOVE.W D7,(A5)+ ;
            MOVE.W D7,CONST2 ;

         ******************************************************************************


         *******************************************************
         *                                                     *
         *     UPDATE THE NEXT NON-RECURSIVE AND RECURSIVE     *
         *       COEFFICIENTS FOLLOWING SIMILAR SEQUENCE       *
         *        OF INSTRUCTIONS AS IN THE PREVIOUS ONE       *
         *                                                     *
         *                                                     *
         *******************************************************

            CMPA.L  #$541F,A6 ;
            BGE.L COMP2 ;
RECUR     : MOVE.W (A4)+,D3 ;
            MULS.W D1,D3 ;
            DIVS.W STB1,D3 ;
            EXT.L D3 ;
            DIVS.W STB2,D3 ;
            ADD.W (A5),D3 ;
            MOVE.W D3,(A5)+ ;
            MOVE.W (A6)+,D7 ;
            MULS.W D1,D7 ;
            DIVS.W STB1,D7 ;
            EXT.L D7 ;
            DIVS.W STB2,D7 ;
            ADD.W (A5),D7 ;
            MOVE.W D7,(A5)+ ;
            SWAP.L D3 ;             * SWAP THE LSW AND MSW CONTENTS OF D3 *
            MOVE.W CONST1,D3 ;      * OCCUPY THE NRCOEFF IN THE LSW AND IN THE MSW OF D3 *
            SWAP.L D7 ;             * DO THE SAME FOR THE RCOEFF *
            MOVE.W CONST2,D7 ;

         ******************************************************
         *                                                    *
         *  SET THE NON-RECURSIVE AND RECURSIVE COEFFICIENTS  *
         *     CONTROL BITS IN ACCORDANCE WITH TABLE 5.1      *
         *                                                    *
         ******************************************************

            TST.W D3 ;              * TEST THE SIGN BIT OF (A)
            BMI.L THA ;               IF NEGATIVE THEN THA,
            LSR.W  #$03,D3 ;
            BSET.B  #$0C,D3 ;         IF POSITIVE, THEN THE SIGN BIT OF (A0=1, BIT 13)
            BCLR.B  #$0D,D3 ;         AND THE ADD/SUBTRACT CONTROL BIT (AS=0, BIT 14) *
FIRST     : MOVE.W D3,CONST1 ;      * SAVE (A) IN THE MEMORY LOCATION ADDRESSED BY CONST1 *
            SWAP.L D3 ;             * GET THE NEXT COEFFICIENT (B) IN THE LSW OF D3 *
            TST.W D3 ;              * TEST THE SIGN BIT OF (B),
            BMI.L TWSEVB ;            IF NEGATIVE, THEN TWSEVB,
            LSR.W  #$03,D3 ;
            BCLR.B  #$0C,D3 ;         IF POSITIVE, THEN BIT 13=0 AND
            BSET.B  #$0D,D3 ;         C2 =1 FOR THE CASE OF NON-UNITY *
```

```
SECOND      : SWAP.L D3 ;
              MOVE.W CONST1,D3 ;
              LSL.W  #$02,D3 ;       * JOIN THE CURRENT AND THE PREVIOUS
              LSR.L  #$02,D3 ;         UPDATED NRCOEFFS TOGETHER IN D3 *
              TST.W D7 ;             * IF THE RCOEFF (A) NEGATIVE, THEN THa *
              BMI.L THa ;
              LSR.W  #$03,D7 ;
              BSET.B  #$0C,D7 ;      * IF (a ≥ 0), THEN THE SIGN BIT ($a_0$=1, BIT 13) AND
              BCLR.B  #$0D,D7 ;        THE ADD/SUBTRACT BIT ($a_s$=0, BIT 14)
THIRD       : MOVE.W D7,CONST2 ;
              SWAP.L D7 ;            * GET THE NEXT RCOEFF *
              TST.W D7 ;               IF(b < 0), THEN TWSEVb *
              BMI.L TWSEVb ;
              LSR.W  #$03,D7 ;
              BCLR.B  #$0C,D7 ;      * IF ≥ 0, THEN THE SIGN BIT ($b_0$= 0, BIT 14) *
FOURTH      : BSET.B  #$0D,D7 ;      * SET $s_1$ TO 1, $s_2$, $s_3$ TO 0
              BCLR.B  #$0E,D7 ;        (REFER TO TABLE 5.1) *
              BCLR.B  #$0F,D7 ;
              SWAP.L D7 ;            * ALLOCATE THE RCOEFF'S BITS ACCORDING TO TABLE 5.1 *
              MOVE.W CONST2,D7 ;
              LSL.W  #$02,D7 ;       * JOIN a AND b TOGETHERED IN D7 *
              LSR.L  #$02,D7 ;
              BCLR.L  #$1D,D7 ;      * RESET $S_4$ OF THE INPUT SCALING COEFFECIENT *
              CMP.B  #$01,D6 ;       * IS IT THE LAST SECOND ORDER SECTION?
              BEQ.L TEST ;             IF YES THEN TEST,
              BCLR.L  #$1E,D7 ;
              BCLR.L  #$1F,D7 ;        IF NO, THEN THE INPUT SELECTOR CONTROL BIT $C_1$,MSB=0)*

            *****************************************************
            *                                                   *
            *         OUTPUT  NONRECURSIVE AND RECURSIVE        *
            *     COEFFICIENTS BIT BY BIT TO THE EXTERNAL RAM   *
            *                   SIMULTANEOUSLY                  *
            *                                                   *
            *****************************************************

OUT         : MOVE.W D1,C ;          * SAVE THE ERROR OUTPUT IN THE MEMORY LOCATION
                                       ADDRESSED BY C *
              MOVE.W  #$02,D1 ;      * NUMBER OF NRCOEFFS AND (OR) RCOEFFS IN EACH SECOND
                                       ORDER SECTION*

            *****************************************************
            *                                                   *
            *  REVERSE THE ORDER OF THE NONRECURSIVE COEFFICIENT, *
            *    BIT BY BIT, AND STORE IT AS A MOST SIGNIFICANT *
            *           WORD IN THE DATA REGISTER D0            *
            *                                                   *
            *****************************************************

BEGIN       : MOVE.W  #$0F,D2 ;      * NUMBER OF THE NRCOEFF'S BITS-1 *
TRANSFER    : MOVE.W D3,D0 ;         * TRANSFER THE NRCOEFF INTO D0 *
              LSL.W D2,D0 ;          * SHIFT THE COEFFICIENT NUMBER OF BITS CONTAINED IN D2*
              LSL.L  #$01,D0 ;       * SHIFT THE BITS ONE BIT POSITION TO THE LEFT AND
                                       OCCUPY IT IN THE MSW OF D0 *
              SUBQ.W  #$01,D2 ;      * NEXT BIT TO BE REVERSED IN ORDER *
              BNE TRANSFER ;
```

```
                 MOVE.W D3,D0 ;
                 LSL.L  #$01,D0 ;      * REVERSE THE MSB OF THE NRCOEFF *
                 MOVE.W D7,D0 ;        * MAKE THE RCOEFF LOCATED IN THE LSW OF D0 *

               ******************************************************
               *                                                    *
               *   WRITE THE COEFFICIENTS BIT BY BIT, ONE BIT OF    *
               *       NON-RECURSIVE AND ONE BIT OF RECURSIVE       *
               *         SIMULTANEOUSLY THROUGH THE OUTPUT          *
               *               PINS (1&2) OF PORT B                 *
               *                                                    *
               ******************************************************

                 MOVE.W  #$0F,D4 ;     * NUMBER OF BITS-1 FOR EACH COEFFICIENT *
WRITE          : MOVE.B D5,(A0) ;      * ADDRESS OF THE FIRST SET OF RAMS *
                 ADDQ.W  #$01,D5 ;     * ADDRESS OF THE NEXT LOCATION IN RAMS *
                 ROL.L  #$01,D0 ;      * ROTATE TO THE LEFT ONE BIT POSITION TO GET ONE BIT OF
                                         EACH COEFFICIENT TO COINCIDE WITH THE FIRST TWO PINS
                                         OF PORT B *
                 MOVE.B D0,(A2) ;      * WRITE THE BITS INTO THE FIRST SET OF RAMS *
                 LSR.W  #$02,D0 ;      * GET READY FOR THE NEXT SET OF BITS *
                 SUBQ.W  #$01,D4 ;     * NEXT SET OF BITS *
                 BNE WRITE ;           * IS IT THE LAST SET? NO, THEN CONTINUE *
                 ROL.L  #$03,D0 ;      * WRITE THE NON-RECURSIVE AND
                 MOVE.B D5,(A0) ;        THE RECURSIVE LSB INTO THE
                 MOVE.B D0,(A2) ;        FIRST SET OF RAMS *
                 ADDQ.W  #$01,D5 ;     * NEXT ADDRESS OF RAMS *
                 SWAP.L D3 ;           * GET THE NEXT NRCOEFF (B)*
                 SWAP.L D7 ;           * GET THE NEXT RCOEFF (b) *
                 SUBQ.W  #$01,D1 ;
                 BNE BEGIN ;
                 MOVE.W C,D1 ;
                 SUBQ.W  #$01,D6 ;     * HAS THE LAST SECOND ORDER SECTION BEEN WRITTEN?
                 BNE.L START ;           IF NOT GO TO START *
                 MOVEA.L WEIGHT,A5 ;   * IF THE WRITE OPERATION HAS COMPLETED FOR 16 NRCOEFFS
                                         AND 16 RCOEFFS, THEN RELOAD A5 WITH THE STARTING
                                         ADDRESS TO PREVENT THE OVERFLOW *


              *******************************************************************************

                 MOVE.W K,D4 ;
                 SUBQ.W  #$01,D4 ;     * IS IT THE LAST SAMPLED DATA SET, IF NOT
                 BNE.L TOP ;               CONTINUE THE UPDATING OPERATION,
                 MOVEA.L INSAMPLE,A4 ;         OTHERWISE GET NEW BLOCKS
                 BRA.L LOOP ;                    OF THE SAMPLED DATA *

              *******************************************************************************

THA            : LSR.W  #$03,D3 ;      * IF THE NRCOEFF (A < 0) THEN THE
                 BCLR.B  #$0C,D3 ;       SIGN BIT (A0=0, BIT 13) AND THE
                 BSET.B  #$0D,D3 ;       ADD/SUBTRACT CONTROL
                 JMP.L FIRST ;           BIT (AS=1, BIT 14) *
TWSEVB         : LSR.W  #$03,D3 ;      * IF THE NRCOEFF (B < 0),
                 BSET.B  #$0C,D3 ;       THEN BIT 13=0 AND
                 BSET.B  #$0D,D3 ;       IF # 1 THEN
                 JMP.L SECOND ;          BIT 14=1 *
```

```
THa          : LSR.W  #$03,D7 ;      * IF (a < 0), THEN THE SIGN BIT (a0=0) AND
               BCLR.B  #$0C,D7 ;       THE ADD/SUBTRACT BIT (as=1) *
               BSET.B  #$0D,D7 ;
               JMP.L THIRD ;
TWSEVb       : LSR.W  #$03,D7 ;      * IF (b < 0), THE SIGN BIT (b0=1) *
               BSET.B  #$0C,D7 ;
               JMP FOURTH ;
TEST         : BCLR.L  #$1E,D7 ;     * IF IT IS THE LAST SECOND ORDER SECTION, THEN
               BSET.L  #$1F,D7 ;       THE INPUT SELECTOR CONTROL BIT (C2=1, MSB) *
               JMP OUT ;

             ********************************************************************************

COMP1        : MOVEA.L INSAM,A4 ;    * IF IT IS THE LAST SET OF COEFFICIENTS
               MOVEA.L OUTSAM,A6 ;             THEN RELOAD A4 AND A6
               JMP NONRECUR ;                    WITH THE STARTING
COMP2        : MOVEA.L INSAM,A4;                   ADDRESS-2 *
               MOVEA.L OUTSAM,A6 ;
               JMP RECUR ;
```