VLSI Architecture For Adaptive Digital Filtering Utilising Number Theoretic Transform

by

Hamid Amir-Alikhani

A thesis submitted for the degree of

Doctor of Philosophy

at the Electronics Laboratories, University of Kent,

Canterbury, Kent, England.

October 1984

Acknowledgments

My sincere thanks go to my supervisor, Dr O.R.Hinton, for his help, guidance and suggestions which have enabled me to bring this work to a successful completion. Thanks are also due to my colleagues and member of staff of Electronics and Computer and Control laboratories for their friendly assistance.

Finally, I would like to acknowledge the support and encouragement I received from my parents , without which this work could not have been undertaken.

Abstract

This thesis investigates the design of a frequency adaptive digital filter in Very Large Scale Integration using the Number Theoretic Transform. The properties of residue numbering systems are investigated, and particularly the possible advantages occured from parallelism, operations without the need for carry, and the absence of round-off errors. The conclusion is reached that this numbering system is in some circumstances more suitable for high speed processing in Very Large Scale Integration. Transform techniques in a finite field are then examined to determine how they could perform filtering operation more efficiently in terms of the number of arithmetic operations required compared with other techniques such as Fast Fourier Transform.

Adaptive filtering in the frequency domain using the Number Theoretic Transform, and in particular Fermat Number Transform, with appropriate formulae for filter weight adaptation using the Least Mean Square algorithm are presented. Several result show that the frequency mean square error as a performance index results in convergence to an optimal solution. A complexity ratio is used to ascertain that frequency adaptive digital filters need less computational power (number of arithmetic operations) than time domain adaptive filters.

A design of special purpose processor using Very Large Scale Integration technology is described, several structures using pipelining and systolic arrays are presented which support the main Very large Scale Integration design features. A table look-up approach using Programmable Logic Arrays (PLAs) for processing elements and a measurements of system performance regarding time/area complexity are described.

Finally, it is concluded that, with suitable further development, a Very Large Scale Integration architecture for frequency adaptive digital filter using number theoretic transform which has high sampling rate, regular internal structure and

capability to parallel devices could likely be achieved.

Publication

O.R.Hinton, H.A.Alikhani

"A VLSI architecture for adaptive digital filtering utilising the number theoretic transform"

Contribution to VLSI 83, VLSI design for digital systems, edited by F.Anceau and E.Ass, North Hollond publication, pp 237-247

Table of Contents

CHAPTER 1

INTRODUCTION

1.1. OVERALL AIM

Up to five years ago the designer constructed his digital signal processors out of available small scale integration (SSI), medium scale integration (MSI), and large scale integration (LSI) components to implement algorithms using the conventional numbering system, rather than others such as the residue numbering system (RNS) or finite field. This is because designing for the conventional numbering system based on SSI, MSI, and LSI proved to be cheaper and more general purpose. However, the advent of very large scale integration (VLSI) has resulted in the spontaneous growth of interest in designing digital signal processors for both conventional and residue numbering systems. This is not hard to understand given the inexpensive computational power offered by VLSI and the ability to tailor-make circuits. The question naturally arises as to whether VLSI, which is proving so successful in designing digital signal processors based on the conventional numbering system, can be used to effectively exploit the potential of a residue numbering system.

This work investigates these ideas by studying the design of an adaptive digital filter in VLSI using RNS. The major topics covered in this thesis are therefore the principles of RNS and number theoretic transform (NTT) in digital filtering, adaptive digital filtering implemented both in the time and frequency domains, and the criteria involved in "good" VLSI design.

1.2. INTRODUCTION TO DIGITAL SIGNAL PROCESSING

Digital signal processing, a field which has its roots in 17th and 18th century mathematics, has become an important modern tool in a multitude of diverse fields of science and technology.

Digital signal processing is concerned with the representation of signals by sequences of numbers and the processing of these sequences. The purpose of such processing may be to estimate characteristic parameters of a signal or to transform a signal into a form which is in some sense more desirable. Signal processing, in general, has a rich history, and its importance is evident in such areas as filtering,

modulation, coding, etc.

Until recently, signal processing has typically been carried out using analog devices such as inductors, resistors, capacitors, and transistors. However, the limited tolerance and stability of these components on chip has made digital signal processors more practical.

With the early development of digital computers, new opportunities in signal processing applications arose. Because of the flexibility of digital computers, it was often useful to simulate a signal processing system on a digital computer before implementing it in hardware. Such computers offered tremendous flexibility, however, the processing could not always be done in real-time. Consequently, digital computers were being used in the simulation of analog signal processing. In keeping with that style, early work on digital signal processing was very much concerned with ways in which digital computers could programmed, so that with A/D (analog-to-digital) conversion of signal followed by digital signal processing algorithm, followed by D/A conversion the overall system would approximate a good analog signal processing algorithm. However, speed, cost, and size were, of course three important factors in favour of the use of analog components.

As signals were being processed on digital computers, there was a natural tendency to experiment with increasingly sophisticated signal processing algorithms. The development of such signal processing algorithms made the notion of all-digital implementation of signal processing system even more tempting. The importance of this was that it had the effect of reformulating many signal processing concepts in terms of the discrete-time mathematics and these techniques then formed an exact set of relationships in the discrete-time domain. This represented a shift away from the notion that signal processing on digital computers was merely an approximation to analog signal processing.

With the advent of integrated circuit technology and the resulting reduction in cost and size of digital components, together with the increased speed, the range of

applications for digital signal processing has increased enormously. By using a large number of SSI or MSI or even LSI circuits a digital signal processing system can be implemented. However, currently available digital signal processing systems are sometimes inadequate because they are not fast enough to handle the large number of calculations that are usually required. Hence, faster and more powerful systems are needed. Because the speed of components is no longer increasing very rapidly, this augmentation in computing power will have to be mainly attained by using parallel techniques. How parallel the system should be constructed for the signal processing, remains to be determined? Consequently, much effort has been dedicated to the development of algorithms and number systems which makes efficient use of parallelism. Because different algorithms favour different processing structures, it is very difficult to decide on a "good" structure for the class of algorithms used in digital signal processing. However, it is possible to make some general remarks regarding the requirements placed on the number systems. A discussion that follows, will demonstrate some important characteristics.

The input into most digital signal processing systems is an analog signal. This signal is applied to an A/D converter. The result is a sequence of integers with an absolute value smaller than an upper bound which, is usually determined by the resolution of the A/D converter. Next, these integers are transformed by a processing unit into a new set of integers. To accomplish this transformation, the processing unit makes use of additions, subtractions, and multiplications. In general divisions do not occur, although rescaling of the data is usually provided. It is necessary to transform the output sequence of integers back into an analog signal. Again, the finite resolution of the D/A converter imposes an upper bound.

The finite resolution of the system, together with the exclusion of general divisions, and the fact that the number of bits required to carry out the arithmetic operations in the conventional numbering system would increase compared to the required number of input and output bits, and the effect of round-off error, suggests using finite field arithmetic. Calculation on ordinary integers can be translated into

residue arithmetic operations, and any algorithm used for the calculation of certain numeric transformations on elements of the field can also be used for the corresponding transformations on ordinary integers.

The advantages of using finite field arithmetic are:

1) It does not introduce roundoff errors.

2) It makes use of residue arithmetic, which potentially can be implemented relatively cheaply and at high speed, especially in parallel and pipelined systems.

3) The arithmetic operations are performed without the need for carry, which is often the problem in high-speed hardware.

The disadvantages are:

1) Relative-magnitude comparison is not so easy.

2) Overflow detection cannot be mechanised easily.

3) Division is not easily accomplished.

The relative balance of advantages and disadvantages is an interesting question.

A lot of work has been done on finite field algorithms [5,7,11,13,15,18,24,104,108], but relatively little on appropriate hardware.

1.3. ADAPTIVE DIGITAL FILTERS

Typical techniques for estimating signals corrupted by additive noise involve passing them through a filter designed to attenuate the noise, but leave the signal unchanged. Estimation theory provides optimal filtering schemes based upon the work of Wiener, Kalman, Bucy and many others [106]. The most common of the above filters are fixed, meaning that they have a fixed structure and fixed parameters which are designed based upon prior knowledge of both the signal and the noise. Another type of filter is the adaptive filters, which has a fixed structure, but variable parameters. These parameters can be adjusted automatically by a built-in parameter adjustment algorithm which seeks to optimise the filter performance in some sense.

The main advantage of adaptive filters is that little or no knowledge of signal or noise characteristic is required.

Adaptive filtering is by no means a new concept. The technique has been applied successfully to many practical problems, such as channel equalisation, echo cancellers, adaptive antenna array, cancellation of noise in ECG, EEG, and speech, filtering of seismic signals and adaptive control [33,34,40,107].

In general, the adaptation mechanisms are concerned with identifying certain characteristic parameters of the observed data. Based on these parameters an error measure is computed which is used to adjust the signal so as to minimise the error.

The most commonly used error criterion is the Mean-Square-Error. The same is true for adaptive filtering and most adaptive filters use the LMS(Least-Mean-Square) algorithm or a variation thereof. The Widrow-Hoff LMS algorithm [33,34,42] is one of the earliest works on adaptive filters using the mean-square technique in the analysis and updating of filter parameters. It uses a steepest descent approach that iteratively optimises a vector of filter coefficients via a gradient search. Convergence of the least mean square algorithm can be guaranteed when filter parameters are chosen appropriately (e.g by scaling), and the resulting filter approximates the well-known Wiener filter.

Perhaps the most familiar structure for an adaptive digital filter is the tapped delay line or transversal filter. It consists of a tapped delay line connected to an adaptive combiner that adjusts the gain (weights) of the signal derived from the taps of the delay line and combines them to form an output signal.

The transversial adaptive digital filter with the above structure and algorithm can be viewed as attempting to find the best FIR approximation by directly estimating the values of w_j of the impluse response. However, for the following two reason, it is often sensible to attempt the adaptation process in the frequency domain :

1) When the filter length is large, the computational speed becomes a crucial factor in most applications, hence with the aid of transform techniques such as

FFT, or NTT it is often possible to gain some speed advantages and significant reduction in computation.

2) In some applications, the impluse response is not of primary interest but rather the frequency response is desired. In this case, the frequency domain adaptation is of greater interest.

In the attempts to find processors of reasonable cost and performance, numerous designs have been published [20-30]. The earlier versions of these were realisations of an ADF in time domain with various techniques being adopted [69] to circumvent the large number of multiplications required. Later frequency domain realisations [103] utilising the Discrete Fourier Transform or Fast Fourier Transform, reduce this number of multiplications and therefore look particularly attractive. This attraction becomes almost irresistible when it is further combined with the advantages offered by residue numbering systems, and the Number Theoretic Transform. The potential advantages of the latter can be predicted considering the majority of digital signals derived from the A/D converters have a sample word length or dynamic range of between 5 to 10 bits, whilst most FFT realisation require complex floating point arithmetic operations. These arithmetic requirements of the FFT are only an artefact of the FFT algorithm and it is reasonable to suppose that more appropriate algorithms such as NTT, would enable equivalent precision to be obtained. The hardware requirements for such a realisation could be expected to be more efficient and would far better match the commonly known regular cells in VLSI technology such as PLAs.

1.4. VLSI ARCHITECTURE OF DIGITAL SIGNAL PROCESSORS

Up to now many digital signal processors have extensively using SSI and MSI components. There are however some disadvantages:

1) Speed limitation: The maximum bandwidth for a processor to handle the data is determined by the rate at which it can process the samples of data.

2) Complexity: They generally require more components than their analog

counter parts.

3) Interface support: In order that the system would communicate with the outside world, they must be supported by A/D and D/A converters. This would increase the hardware complexity.

These disadvantages made digital signal processors impractical a decade ago. However, the great advent in Very Large Scale Integration technology has made it possible for their use in practical applications.

There are two general approaches of using VLSI technology for digital signal processors:- we can use a general purpose, or design a special purpose processor. The choice between the general and special purpose processor depends on many factors. To make the maximum use of technology, we should restrict the actual operations performed to essential arithmetic, that decoding of instructions and the storing and fetching of signal values to and from memory represent "wasted resources" in terms of area and power. The design and successful fabrication of an integrated circuit chip is difficult and expensive enough so that it is desirable to develop a flexible general purpose device for mass production. This is bound to change with the development of better tools for chip design.

Because many digital signal processing algorithm can be partitioned, it is possible to distribute the processing functions over several chips so this has the advantage of modularity for growth.

Combining a modular structure and the technological advances offered by VLSI leads to enhancement in residue numbering system implementation for high speed digital signal processing applications. The VLSI approach is promising as RNS supports two main VLSI design features:

1) RNS has a parallel nature where the arithmetic operations are performed independently for each module which supports distributed processing. This can minimise the execution time and results in higher data throughput.

2) Because of the finite number of states in RNS systems, a memory intensive architecture is suitable and this is also appropriate for VLSI implementation.

With recent advances in VLSI technology, it has become evident that digital signal processing can be implemented in cost-effective technology. Such implementations operate faster, consume less power, and are more reliable than their predecessors built from SSI, or MSI. In order for a system to be realisable in VLSI, it should have the following characteristics:

1) Simple and regular design

In order that the system be cost-effective, it should be implemented using only a few different types of simple cell. In some cases, it may require more processing power, to do this it is necessary to expand the number of processing units. This is only possible if the interconnections between processing units are regular.

2) Pipelining and concurrency

There are essentially two ways to build a fast processor. One is to use fast components and therefore high clock rate, and the other is to make use of concurrency. In applications which is limited by the component speed, the only solution that can achieve higher system complexity would be the use of parallelism and high degree of pipelining which appears appropriate especially with VLSI technology.

1.5. ORGANISATION OF THE THESIS

The rest of the thesis is organised as follows:

Chapter 2: Finite number system and convolution algorithms

An overview of finite number system is outlined and importance of Euler's and Fermat and Chinese theorem has been presented. The convolution algorithm in the finite field is also presented.

Chapter 3:Convolution using Number Theoretic Transform

The theory of Number Theoretic Transforms and their usage in conjunction with the calculation of the convolution algorithm is explained. The computational complexity of various Number Theoretic Transform techniques are reviewed in order to show the advantages of one transform technique over the others.

Chapter 4:Adaptive filtering

The frequency adaptive digital filter algorithm utilising the Number Theoretic Transform is discussed. The formulae for the convergence and adaptation speed are derived and compared with the LMS algorithms and it is shown that under certain condition the algorithms are equivalent. Some simulation results are presented in order to verify the convergence of the FLMS. The computational complexity of the LMS and FLMS are presented and a complexity ratio is analysed. It has been shown that the frequency adaptive filter is ten times more efficient (in terms of arithmetic complexity) than time domain adaptive filters.

Chapter 5:Special vs general purpose signal processors

A survey of designs of special and general purpose processors are reviewed including the single chip, array and multiprocessors, and systolic arrays.

Chapter 6:VLSI architecture for adaptive digital filter

The design of a VLSI architecture for ADF is described. It

includes the design of processing elements based on the table look-ups using the Programmable Logic Arrays. The system performance for a number of structures based on the Time/Space complexity has also been presented.

Chapter 7: Conclusion

The achievements of the work are reviewed. A number of improvements are proposed which should allow efficient implementation of the frequency adaptive digital filter utilising the VLSI technology.

CHAPTER 2

NUMBERING SYSTEM AND CONVOLUTION ALGORITHMS

2.1. FINITE NUMBERING SYSTEM AND CONVOLUTION ALGORITHM

In this chapter we will present an overview of finite number systems or residue number systems (Finite Fields) . Residue Numbering Systems (RNS) have been useful in coding theorem and digital system theory because they are structured with a finite number of states on which addition and multiplication form a closed operations. Many authors [1-6] have been investigating the use of these numbering system for digital signal processing algorithms. Therefore, in this chapter we shall investigate the properties of such a system, when used for filtering and other signal processing algorithms.

The material covered here is divided into two main parts ; elementary number theory and convolution algorithms.

In the elementary number theory section, we introduce briefly the concepts of rings and fields [1-6] which play an important part when used in conjunction with Number Theoretic Transform. In part two we introduce the fast algorithm for convolution which is commonly used for digital signal processing.

2.2. ELEMENTARY NUMBER THEORY

Definition : A Finite Number Residue System is a triple ;

$$FNRS = (S, I, F)$$

Where S is the symbol set (set of n-tuple digit vectors), I is the interpretation set, and F is the evaluation function which maps S on to I .

Figure 1.1 shows the sets associated with a FNRS.

The Residue Number System (RNS) as opposed to the fixed-radix system, where the number system is completely specified by stating the radix, is described by stating the base. However, for the RNS, this base does not consist of a single radix but of an N-tuple of relatively prime integers m_1, m_2, ..., m_N where m_i is called " MODULO ".

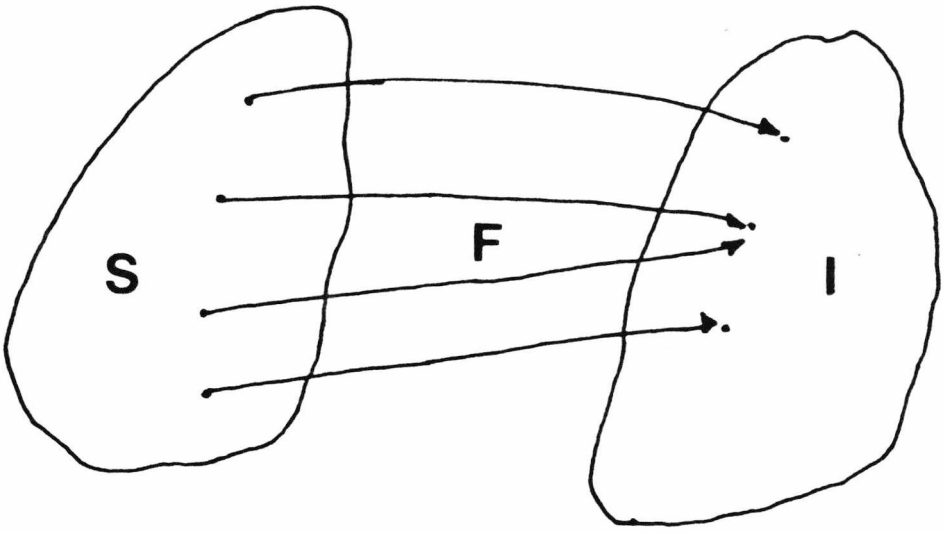


Figure 1.1 FNRS representation of a set

For any given base, the residue representation of an integer X is another N-tuple { x_1, x_2, ..., x_i, ... x_N }, where x_i is defined as :

$$x_i = X \bmod M \quad or \quad |X|_{M_i}$$

where $M = \prod_{i=1}^{N} m_i$.

The residue representation of an integer is unique ; i.e. each integer has only one representation, since there exists only one least positive remainder for any number. However, the converse of the statement is not true. This ambiguity of residue representation is avoided if the integer X lies between 0 and M-1.

The algebraic operation in the RNS consists of operating on the residue digit pairwise with modular arithmetic. A number of arithmetic relationships are been presented here which follow from the definition of the residue system [4].

a) Additive inverse modulo M

$$|-X|_M = |M - X|_M$$

b) Addition and subtraction modulo M

$$|X + Y|_M = ||X|_M + |Y|_M|_M$$

c) multiplication modulo M

$$|X.Y| = ||X|_M \cdot |Y|_M|_M$$

So far, only methods for converting to the residue numbering system and performing arithmetic operations within this systems have been discussed. We shall now discuss the conversion from residue systems back into weighted numbering system.

Given the residue representation $\{ x_1, x_2, \ldots\ldots, x_N \}$of X, it is possible to determine X by Chinese Remainder Theorem provided that the pairwise modulo are relatively prime.

Definition : The Chinese Remainder Theorem (CRT) can be defined as follows;

$$X = |\sum_{i=1}^{N}(\frac{M}{m_i})\cdot x_i \cdot T_i|_M$$

where $(\frac{M}{m_i})\cdot T_i \equiv 1 modulo \ \ M$

Since the arithmetic operation is done on the various residues, without any carry from one residue to another one can chose $M = \prod_{i=1}^{N} m_i$ to be the product of many small relatively prime modulo. Hence, the computation can accommodate large numbers, although actual computations are performed on a large set of small residues. Thus, the RNS are quite attractive for high speed multiplications and additions. But this advantage is offset by many practical difficulties such as, wordlength etc. which make them rarely used. However, we shall see in later chapters that RNS can play an important role in digital discrete transforms such as Number Theoretic Transform.

2.2.1. FINITE FIELD

In order to understand fully the concept of discrete transform over finite field it is necessary to further develop the mathematical model for Number Theoretic Transform. Material related to Residue arithmetic system is developed in a previous section and a review of RNS properties is included in Appendix C.

1) RING : A non-empty set R is said to form a ring with respect to the binary operation + and × provided, for any x,y,z ∈ R, the following properties hold ;

a) $(x + y) + z = x + (y + z)$

b) $x + y = y + x$

c) there exist $0 \in R$ such that $0 + x = x$

d) for each $x \in R$ there exists $-x \in R$ such that:

$$x + (-x) = 0$$

e) $(x . y) . z = x . (y . z)$

f) $x . (y + z) = x . y + x . z$

g) $(y + z) . x = y . x + z . x$

If the following properties hold, then R is a field.

h) there exist non-zero element $1 \in R$ such that $x . 1 = x$

i) for each non-zero element $x \in R$ there exist x^{-1} R such that $x . x^{-1} = 1$

j) $x . y = y . x$

2) ORDER of a FIELD : The order of a field is the number of elements it contains. The field is said to be finite if its order is finite . Finite fields are often called Galios fields and denoted by GF (M) where M is the order .

3) CHARACTERISTIC : The characteristic of a field F is define to be the smallest positive integer m so that $m . x = 0$ for every $x \in F$. If no such integer exists, the field is said to have characteristic zero.

4) SUBFIELD : A subfield of a field F is any subset G of F which is itself also a field . e.g field of real numbers is a subfield of the field of complex numbers.

5) PRIME FIELD : A field is prime if it does not contain any proper subfield

Let M be a prime, then Z(M) is the same as GF(M) since all finite fields of the same order are equivalent.

The following definitions and theorems relate specially to the properties of the finite set of elements of the field Z(M) . It is these properties which will determine how the kernel for discrete transforms is to be selected.

All integers "a" which give the same remainder when divided by M can be thought as belonging to the same equivalence class relative to the equivalence relation (see appendix C) . Among these classes those corresponding to integers which are relatively prime to M play a particularly important role in defining Primitive Roots. We shall need to know how many integers smaller than M and relatively prime to M exist. This quantity is defined by the following [1-3]:

EULER'S FUNCTION : The function $\Phi(M)$ is called Euler's function and is defined as the number of integers in Z(M) that are relatively prime to M. If M is prime then:

$$\Phi(M) = M-1$$

If $M = P^c$, the only numbers less than M and not prime to P are the multiples of P . Therefore :

$$\Phi(M) = \Phi(P^c) = P^{c-1}(1-P) = P^c(1-\frac{1}{p})$$

If "a" and "b" are two mutually prime integers then :

$$\Phi(a.b) = \Phi(a).\Phi(b)$$

If the integer M is given by its prime factorisations $M = m_1, m_2, \ldots\ldots, m_i$ then :

$$\Phi(M) = M.\prod_{i=1}^{K}(1-\frac{1}{m_i})$$

However, in order to define the primitive root we need to define the order of an element of the field Z(M) . To do this let us consider an integer x_n defined by :

$$x_n \equiv a^n \ \ modulo \ \ M$$

If n takes successively the values 0,1,2,..... then x_n will takes the value $x_0, x_1, \ldots\ldots$. Since x_n can only take M distinct values 0 to M-1 , x_n will necessarily repeat a previously computed value x_r , with $r > n$. Let r be the smallest value of n for which such repetition occurs. Therefore x_n repeats itself cyclically with a period of "r" elements (integers) . The condition for this case is given by EULER'S THEOREM :

EULER'S THEOREM Definition :

If $M > 1$ and (a,M) = 1 then ;

$$a^{\Phi(M)} \equiv 1 \ \ modulo \ \ M$$

When M is prime and M does not divide "a" and $\Phi(M) = M - 1$ then, the Euler's Theorem reduces to FERMAT THEOREM .

FERMAT THEOREM Definition :

If M is a prime, then for every integer "a" ;

$$a^{M-1} \equiv 1 \ \ modulo \ \ M$$

The main interest of these theorems lies in the specification of the order of an element (integer).

We have seen that the sequence $x_n \equiv a^n \ \ modulo \ M$ repeats itself with a periodicity "r". If $a^n \equiv 1 \ \ modulo \ M$ for some value "r" of "n", the sequence will repeat itself from the beginning. Hence, if "r" is the smallest positive integer such that $a^r \equiv 1$ modulo M, the complete sequence of integer a^n modulo M (residue) will be periodic with period (order) "r" .

Let us now determine the maximum vale of "r" for a given M . From Euler's theorem if (a,M) = 1 then $r = \Phi(M)$. If $(a,M) \neq 1 = d$ where d is the greatest common divisor of a and M . We have $(\frac{a}{d}, \frac{M}{d}) = 1$ and $r' = \Phi(\frac{M}{d})$. Since $\Phi(\frac{M}{d}) < \Phi(M)$ then the period "r" is maximum for (a,M) = 1 .

We shall call the element α which generates a sequence of order (length) $r = \Phi(M)$ as PRIMITIVE ROOT . An element α generating a shorter cyclic sequence of order $r < \Phi(M)$ will simply be called a ROOT of order r.

2.3. CONVOLUTION ALGORITHMS

The adaptive digital filtering is concerned with two major operations ; a) convolution of two sequences to produce an output and b) the correlation of the input sequence with the error to update the filter coefficients in order to minimise the mean-square error. In this section we shall look at some of the algorithms which exist to evaluate the convolution algorithm faster, since correlation is the same as convolution but the sequence is time reversed.

The circular convolution y_k of two sequences x_n and h_n of the length N can be defined as :

$$y_k = \sum_{n=0}^{N-1} x_n . h_{k-n} \qquad (1)$$

for k= 0,1,2,.....,N-1

It can be seen quite clearly from (1) that, it would required a number of multiplications and additions of the order of N^2 . For large convolution the corresponding processing load becomes rapidly excessive and therefore considerable effort has been devoted to devising faster computation methods or algorithms [8-11]. Most fast convolution algorithms are either based on the transform techniques, and shall be dealt with in a later chapter, or on the techniques of replacing the large convolution with a large number of small convolutions [Nussbaumer, 6, 1981]. In fact, the number of such algorithms is so large that an exhaustive presentation would be

impossible or not totally related to this thesis. Moreover, many seemingly different algorithms are essentially identical and differ only in the formalism used to develop a description [Nussbaumer, 6,1981].

Most convolution algorithms which have been developed both theoretical and practical levels, treat data as a quantised quantity. That is the data which is discrete in amplitude [Rabiner and Gold, 12, 1975] . There the point of view was to treat each sample as a sum of continuous quantity and an error. The goal was to estimate or minimise it.

Here, we present a different point of view, namely that all data can only take quantised (integer) values. We shall present a method to evaluate the convolution of such quantities. Namely, the convolution in the finite ring of integers, or finite field, utilising the residue arithmetic operations.

Knuth [1,1981] and Szabo [4,1967] have investigated and demonstrated the use of the Residue numbering system to speed up the operation of multiplication on general purpose computers. Although the technique never achieved widespread usage because of hardware complexity and cost, it is now feasible to achieve this with the aid of Very Large Scale Integration (VLSI).

Recall equation (1) and rewrite it as follows;

$$y_l' = \sum_{m=0}^{N-1} x'(m) \cdot h'(l-m) \quad modulo \quad M \tag{2}$$

where $l = 0,1,.....N-1$ $x'(m)$ and $h'(m)$ are the residues of x(n) and h(n) respectively and define as;

$$x'(m) \equiv x(n) \quad modulo \quad M$$

$$h'(m) \equiv h(n) \quad modulo \quad M$$

and addition, multiplication are define modulo a prime M.

Since the arithmetics modulo a prime M are exact, multiplication and addition of integers do not require rounding. Hence, operations in the Residue number system are

free from round off error, and the only source of error occurs when we quantise the sample of input signal, i.e. in the encoder.

Because of this lack of round off, the major limitations of Residue Number System are magnitude and overflow detections. The later disadvantage can be prevented by using a large modulo (M) or dynamic range . However, performing arithmetic on a large dynamic range can be very costly and complex . In order to overcome this difficulty we can use several small modulo which are pairwise relatively prime and paralleled to obtain a sufficient dynamic range [Jenkins, 13, 1977] .

Let ; $M = \prod_{i=0}^{L} m_i$ in this case equation (2) becomes:

$$y_{l_i}' = \sum_{m=0}^{N-1} x_i'(m) \cdot h_i'(l-m) \quad modulo \quad m_i$$

where ;

$$x_i'(m) \equiv x'(m) \quad modulo \quad m_i$$
$$h_i'(m) \equiv h'(m) \quad modulo \quad m_i$$

Therefore y_l' is obtained by computing the L product of $x_i'(m) \cdot h_i'(m) \quad modulo \quad m_i$ .Hence ;

$$y_l' = \sum_{j=0}^{L-1} y_{l_i}' \tag{3}$$

However, the aim was to obtain the convolution given by equation (1) namely y_l rather than y_l' . Therefore one needs to translate the result calculated by equation (3) which are in residue form, back into the natural representation.

Using Chinese Remainder Theorem, which was investigated by Szabo and Tanaka [4]) we can reconstruct y_l from y_l' . The question then arises that, although the computation is free of round-off error it is not likely that the full precision of the final output will be retained . This is because of the lack of sign detection in the Residue Numbering System . Szabo [4,1967], Jenkins [13,1977] argued that by altering the decoding process it is possible to produce an output that has been correctly scaled.

They have suggested that if the input data are within certain boundaries, then the

result calculated in the Residue Numbering System and properly decoded is identical to the result obtained using conventional arithmetic. This boundary is defined as :

$$I \quad [-W, W]$$

where I is any integer and $W = \frac{M-1}{2}$.

Thus integers between { 0, W } would be accounted as positive quantities and integers belonging to { -W, 0 } would be negative quantities . On the other hand, integers between { 0, $\frac{M-1}{2}$ }and { $\frac{M-1}{2}$, M-1 } are positive and negative quantities respectively .

The only drawback with the technique is as long as the quantisation of input signal does not cause large negative quantities to be mistaken for large positive values or vice-versa . Such a draw back can be prevented by choosing m_i large enough in relation to dynamic range (M) . This latter modification will increase the hardware complexity further . Taylor [15, 1981] suggested three methods for various memory / throughput / dynamic range tradeoffs, for encoding and decoding the natural integers to and from Residue representation.

The principle of convolution using Residue Arithmetic can be illustrated by a simple example . Let us suppose that it is desirable to calculate the convolution of two sequences x(n) and h(n) described by the following equation ;

$$y(n) = h_0 \cdot x(n) + h_1 \cdot x(n-1)$$

Where $h_0 = 127$ and $h_1 = -61$.

Whenever x(n) = 25 and x(n-1) = 83, y(n) = -1888 . Let M = (19,23,29,31) be the modulo set on which the Residue Numbering System will be based . Since $M = \prod_{i=0}^{3} m_i = 329863$ the dynamic range of this system is 196431 or $\frac{M-1}{2}$. Using equations (2.2) and (2.3) h(n) and x(n) becomes

$$h_0 \equiv (13,12,11,3) \quad h_1 \equiv (15,8,26,1)$$

$$x(n) \equiv (6,2,25,25) \quad x(n-1) \equiv (7,14,25,21)$$

respectively.

The residue encoded output $y'(n) \equiv y_0'(n), y_1'(n), y_2'(n), y_3'(n)$ would be computed in parallel by using equation (3) :

$$y_0'(n) = |13\times6 + 15\times7|_{19} = 12$$

$$y_1'(n) = |12\times2 + 8\times14|_{23} = 21$$

$$y_2'(n) = |11\times25 + 26\times25|_{29} = 26$$

$$y_3'(n) = |3\times25 + 21\times1|_{31} = 3$$

Using the Chinese Remainder Theorem it is possible to decode the residue outputs in order to obtain y(n) . Therefore y(n) becomes;

$$y(n) = \left|\sum_{i=0}^{3} m_i \left|m_i^{-1}\cdot y_i'(n)\right|_{m_i}\right|_M$$

where m_i^{-1} is the multiplicative inverse of m_i .

These constants can be computed as follows ;

$$m_0 = 20677 \quad m_1 = 17081 \quad m_2 = 13547 \quad m_3 = 12673$$

and

$$m_0^{-1} = 4 \quad m_1^{-1} = 20 \quad m_2^{-1} = 22 \quad m_3^{-1} = 5$$

Therefore

$$y(n) = |20677\,|4\times12|_{19} + 17081\,|20\times21|_{23} + 13547\,|22\times26|_{29} + 12673\,|3\times5|_{31}|_{329867}$$

$$= |206770 + 102486 + 284487 + 190095|_{392863}$$

$$y(n) = 390975$$

Since y(n) lies between $\frac{M-1}{2}$ and M, therefore it is a negative integer . To find the true value of y(n), we should compute the complement of it . Hence y(n) = -1888 which is the same as if we were using the conventional system.

SUMMARY

In this chapter we presented a number of theorems and definitions related to finite fields . These Theorems and definitions present a broad scope, as regards the usage of these, when used in conjunction with the Number Theoretic Transforms . In the second part of this chapter we presented an algorithm for performing convolution in the ring of integers . The advantage of computing the convolution in the Finite Field compared to the conventional number system is that it is exact and free of errors and that it occurs due to register limitation as in the case of the conventional system . As well as error free computation, it is possible to perform the convolution much faster in the Finite Field rather than in the conventional system. This is because arithmetic is performed in parallel on the relatively small prime integers . It is this parallelism and hence the speed, which greatly influences the use of these algorithms in conjunction with today's technology for example Very Large Scale Integration .

One of the limitations of using the technique described in the previous section is, as the sequence length increases the number of arithmetic operations will increase accordingly, and therefore it is not suitable for many practical cases where the sequence length is high . In order to overcome this difficulty, it is possible to perform the convolution in the frequency domain with the aid of the well-known algorithms such as DFT, FFT and NTT . However these techniques have their own limitations, and will be discussed in detail in the next chapter.

CHAPTER 3

NUMBER THEORETIC TRANSFORMS

3.1. CONVOLUTION USING NUMBER THEORETIC TRANSFORM

In chapter 2, we introduced the concept of Finite Number Theory and its usage in conjunction with the calculation of the convolution of two sequences. There , we showed how modulo arithmetic and the Chinese Remainder Theorem can be used in order to compute the cyclic convolution of two sequences x(n), h(n) in time domain. We have argueed that the number of arithmetic operations (multiplications , additions)are directly proportional to the sequence length (N) . Therefore, direct calculation of the convolution for long sequences would not be very efficient.

Many authors [1-6] have shown that convolution in the frequency domain with the use of discrete transforms, with the convolution property, would reduce the number of arithmetic operations, and hence, it is more efficient. In this chapter we begin by presenting the definition and basic conditions for the existence of discrete transforms, with the emphasis on the Number Theoretic Transform possessing the convolution property . Then, we shall introduce the two most important Number Theoretic Transforms, namely MERSENNE TRANSFORM and FERMAT TRANSFORM and further we have generalized our definition of the Number Theoretic Transform to include Complex Transforms . Finally, we concluded the chapter by discussing the constraints and computational complexity for these transforms.

3.2. DISCRETE TRANSFORM

The discrete transform possessing the convolution property can be used to efficiently compute the convolution of two long sequences, which is the requirement of most practical cases. This computation is done under certain conditions which will be described later.

All discrete transforms have the same general form, with conditions for an inverse transforms, and the convolution property being determined by the number system over

which the transform is defined . In the following discussion the convolution property is a focal point in the development of the NUMBER THEORETIC TRANSFORM . In a later section the definition and the conditions of the general Number Theoretic transform, not requiring the convolution property, will be given .

The discrete transform of a sequence x(n) where n = 0,1,2,3,.....,N-1 is defined as :

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot C_f \ (n,k) \quad k=0,1,2,.....,N-1$$

and with inverse transform defined as :

$$x(n) = \sum_{k=0}^{N-1} x(n) \cdot C_i \ (n,k) \quad n=0,1,2,.....,N-1$$

Where $C_f \ (n,k)$ and $C_i \ (n,k)$ are the forward and inverse kernel coefficients respectively.

The discrete transform is called the Discrete Fourier Transform (DFT) when the kernels of the transforms are defined as complex exponential i.e. $C_f = W^{nk}$ where $W = \exp(\frac{-j2pi}{N})$ for a sequence of length N . The DFT is the only transform in the complex number field which has the convolution property . The convolution implemented is the cyclic convolution property if the output sequence y(n) of two sequences, x(n) and h(n), each with a period of N, can be related as :

$$T[y(n)] = T[x(n)] \cdot T[h(n)] \qquad (3.1)$$

Then we can say that the transform has Cyclic Convolution Property (CCP) . Linear convolution can be calculated with cyclic convolution by augmenting the sequence x(n) and h(n) with a sufficient amount of zero to prevent aliasing .

The Transform approach to performing convolutions is useful when there exists a fast algorithm, such as Fast Fourier Transform, to reduce the number of arithmetic

operations . However, the disadvantages of using FFT to calculate convolution are significant amounts of rounds of errors and considerable number of multiplications .

Using kernels defined over a Finite Field all calculations are performed exactly . The transforms with kernels defined over a Finite Field GF(M) are called NUMBER THEORETIC TRANSFORM [Pollard, 26, 1971] . The Number Theoretic Transform (NTT) of a sequence x(n) where n=0,1,2,....,N-1, is defined as :

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot \alpha^{n.k} \quad k=0,1,.....,N-1 \tag{3.2}$$

With inverse transform defined as :

$$x(n) = N^{-1} \sum_{k=0}^{N-1} x(n) \cdot \alpha^{-n.k} \qquad n=0,1,...N-1 \tag{3.3}$$

Where the " α " is a root of order N, N^{-1} represents the multiplicative inverse in the field in which the arithmetic is carried out, and all operations performed modulo M .

The relationship between the modulo M, the transform length N , and the kernel " α " is reviewed in definitions 3.1.1 and 3.1.2 [Agarwal,23,1975 : Agarwal,22,1974],establishing the conditions for the existence of a Number Theoretic Transform possessing the convolution property .

Definition 3.1.1 : A Number Theoretic Transform of length N over Z(M), having the DFT structure, will implement cyclic convolution property if and only if there exists N^{-1} and an element " α " , which is a root of unity of order N.

Definition 3.1.2 : A Number Theoretic Transform of length N over Z(M) having the DFT structure will implement cyclic convolution modulo M if, and only if,

$$N \ \ divides \ \ O(M)$$

Where O(M) is the order of M .

With the aid of the above definitions we can summerise the conditions for the existence of the Number Theoretic Transforms as follow ;

1) $\alpha^N \equiv 1$ Modulo M

2) $N.N^{-1} \equiv 1$ Modulo M

3) $gcd[(\alpha^t - 1), M] = 1 \quad t = 1,2,\ldots N-1$

Immediate consequences of the conditions 1,2,3 are that, the inverse transform defined by (3.3) is indeed the inverse transform, N must be relatively prime to M and α must be a root of order N modulo M respectively. Example : let N=4 $\alpha = 2$ M = 5

1) $2^4 = 16 \equiv 1$ modulo 5

2) N=4 $N^{-1} = 4$ hence $N.N^{-1} \equiv 1$

3) [2, 1] = 1 [3, 5] = 1 [7, 5] = 1

In order for the Number Theoretic Transform with the conditions sets above and convolution property to be more attractive in comparison with other transforms or convolving directly, they should be computationally efficient and easy to implement using the available technology. The requirements for the efficiency are summerised as follow [Agarwal,23,1975];

1) N should be highly composite (preferably a power of 2)

2) N should be large enough for practical cases.

3) α should have very few bits in binary representation in order that the multiplication of power of α be simple operation.

4) M needs to be large enough to avoid overflow, and also a few binary representation.

So far, we have shown the conditions and requirements for an Number Theoretic Transforms with the cyclic convolution property to exist. Let us now investigate the good choices of M, α for which the maximum Transform length $N_{\max} = O(M) = \Phi(M)$ is not too small . From (4) the most obvious choice of M is 2^p. However, in this case the maximum transform length is 1 and therefore of no practical interest . Similarly, when M is even, one of the factors of M is 2 and from definition 3.1.2 the maximum transform length is equal to 1. Thus, the only case of interest is when M is odd.

Now we shall examine the most important Number Theoretic Transform and their properties, when M is either a Mersenne number ($M_p = 2^p - 1$) or a Fermat number ($M_f = 2^k + 1$) . Further information about the properties of these number can be found in [Vinogradov,3,1955 : Nussbaumer,6,1981].

3.2.1. MERSENNE NUMBER TRANSFORMS

The most interesting case for Mersenne numbers M_p corresponds to p being a prime number, that is p = 3,5,7,9,.... . From definition 3.1.2 and Fermat's theorem and the fact that M_p is a prime number, the possible transform length can be given by ;

$$N/(M_p - 1) \quad \textit{that is} \quad N \quad \textit{divides} \quad M_p - 1$$
$$N \,/\, (2^p - 2) \qquad (3.1.1.1)$$

It can be concluded from the above equation, that 2 and p are the obvious divisor of $2^p - 2$, hence we can define a Number Theoretic Transform of length p and 2p modulo a Mersenne Number . In order to complete the definition of Mersenne Transforms, we must find the root α of order p and 2p . Since $M_p - 1$ is a prime, an obvious root of order p is 2, since the p^{th} first powers of 2 are all distinct . Mersenne transforms can also be defined with transform lengths of 2p . In this case (-2) is a root of order 2p modulo a Mersenne number, since the 2p first powers of

-2 are all distinct .

For $\alpha = 2$ the Mersenne Number Transform and its inverse can be defined respectively as follow :

$$X_m(k) = \sum_{n=0}^{p-1} x(n) \cdot 2^{nk} \qquad mod M_p$$

$$x(n) = P^{-1} \sum_{k=0}^{p-1} X_m(k) \cdot 2^{-nk} \quad mod M^p$$

Where n,k = 0,1,...,p-1 and $P^{-1} = M_p - (M_p - 1)/p$

Thus a length-p circular convolution can be computed as shown in figure 3.1 by three Mersenne Transforms plus p multiplications in the transform domain . However, if one of the input sequences is fixed, its transform can be precalculated and therefore only two transforms are needed to evaluate the convolution .

Similarly, we can define a Mersenne Transform of length 2p with root $\alpha = -2$. It is also possible to define Mersenne transform with a dimension larger than 2p, since the maximum transform length is $2^p - 2$. However , the root α of these transforms is no longer a power of two, and need some general multiplications . Therefore, in practice only the transforms of length 2 and 2p are called Mersenne Transforms . The relationships between transform length, root, and word length are summerised in table 3.1 .

From the definition of Mersenne numbers, we can deduce that any integer can be represented by p-bits word . Thus, additions modulo a Mersenne number is performed by using a binary full adder of p-bits and folding the MSB output back into LSB . This is similar to the One's Complement addition . Similarly, multiplications modulo M_p can be performed by forming the 2p-bit product of the two words and adding the pth-MSB to the pth-LSB which is again Similar to the One's Complement operation .

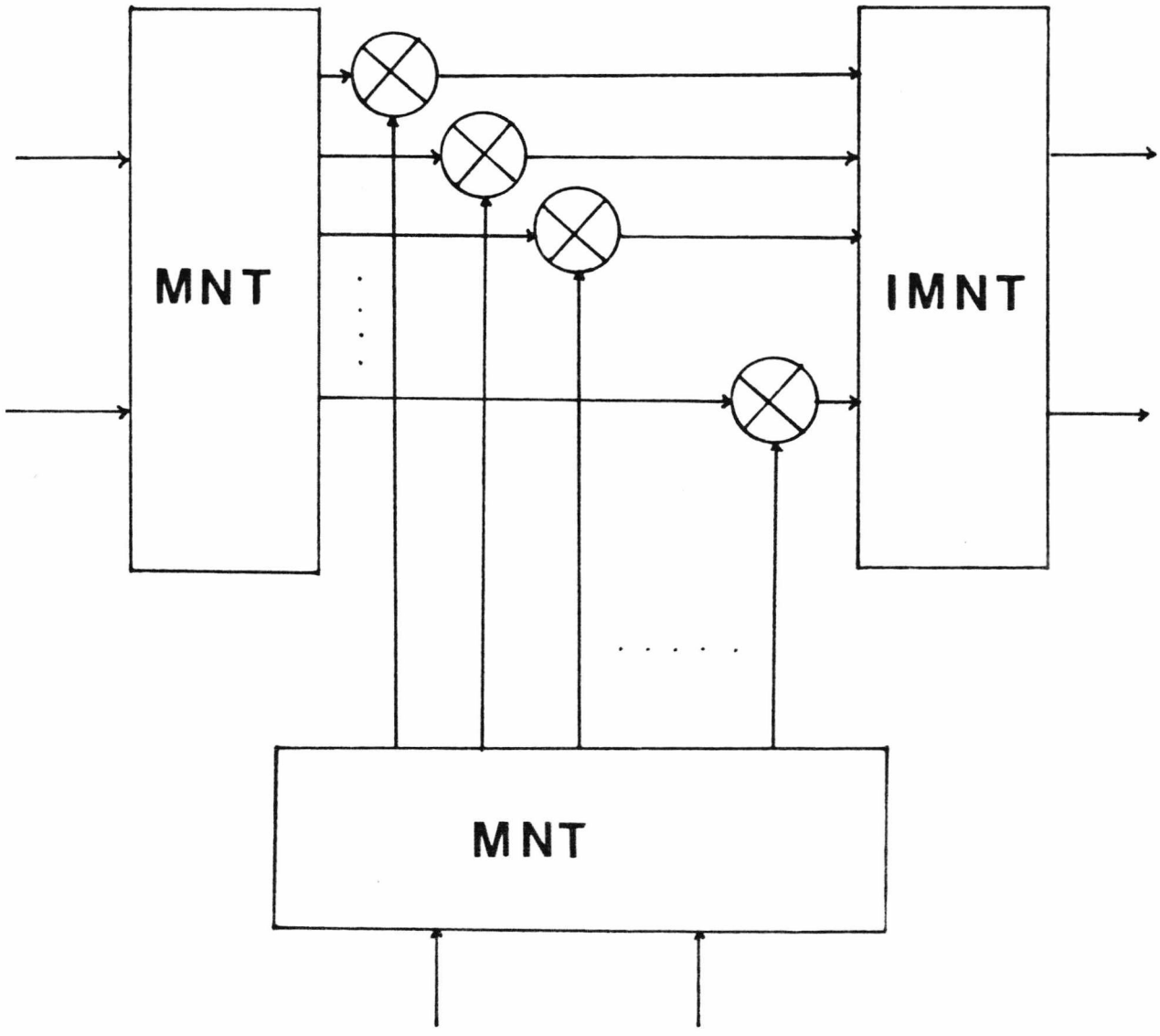


Figure 3.1 Block diagram of Mersenne Number Transform

However, multiplication by powers of two is particularly simple, and only amounts to shifts.

In summary, a Mersenne Transform requires p(p-1) additions and $(p-1)^2$ shifts and p general multiplications in transform domain . Apart for the advantages explained above, the principal deficiencies of Mersenne Number Transform are due to firstly, the

lack of ability to perform a fast transform algorithm and secondly, Mersenne Number Transform is not attractive in the real-time signal processing (e.g Image processing, Speech, Filtering etc) , this is because there is a rigid relationship between word length and transform length .

In order to ease some of these limitations we can define another transform having the cyclic convolution modulo a Fermat number .

3.2.2. FERMAT NUMBER TRANSFORMS

Fermat numbers are defined as $M_f = 2^k + 1$. Let us assume that k is an odd number, therefore 3 will divide M_f and hence we can say that M_f is a composite, and 3 is at least one of the factors, that is $M_f = m_1, m_2, \ldots\ldots, m_i$. Using the Euler's Theorem (chapter 3) , we can define the order of M_f . That is

$$O(M_f) = GCD[(m_1-1),(m_2-1),\ldots..] = \Phi(M_f) \qquad (3.1.2.1)$$

Therefore $O(M_f) = 3-1 = 2$

In order that the transform supports the convolution property ;

$$N_{\max} = O(M_f) = 2$$

Hence, the possible transform length when k is odd, will be equal to 2 . Thus, we consider only k as an even number . Let $k = s2^t$ where s is an odd integer, then M_f divides $2^{sk} + 1$ and hence, the possible transform length will be governed by the possible length for M_f . Therefore only the integers of the form $2^k + 1$ are of interest .

The Number Theoretic Transforms defined modulo a Fermat number are called Fermat Number Transforms . Let us now consider transform lengths possible in arithmetic modulo various Fermat Numbers .

Since the first five Fermat numbers are prime, we can define the order of M_f as follows:

$$O(M_f) = M_f - 1$$

Therefore we can have a maximum transform length of 2^k . For these Fermat primes the integer 3 is an " α " of order $N = 2^k$ allowing the largest possible transform length . With 3 as a root of order N , it is no longer possible to achieve the powers of 3 with simple shift operations and therefore need more general multiplications . However, it is still possible to find roots that only need bit shift operation, but will not satisfy the maximum transform length criteria . The integer 2 is a root of order $N = 2^{k+1}$ $moduloM_f$, since 2^i takes N distinct values for i=0,1,2,.....,2^{t+1} -1 [Agarwal,23,1975] . This means that when M_f is prime we can define Fermat Number Transform and its inverse of length N = 2k with root 2 as follow :

$$X_f(k) = \sum_{n=0}^{2k-1} x(n) \cdot 2^{nk} \qquad (3.1.2.2)$$

$$x(n) = F \sum_{k=0}^{2k-1} X_f(k) \cdot 2^{-nk} \qquad (3.1.2.3)$$

Where $F = -2^{k-t-1}$

It is also possible to double the transform length . Agarwal[27,1973] showed that with $\alpha = \sqrt{2}$ is a root of order N = 4k and multiplications by powers of $\sqrt{2}$ are simply a word shift . Table 3.2 lists possible parameters for Fermat Number Transform .

In computing the Fermat transform, arithmetic is done modulo $2^k + 1$ Arithmetic modulo a Fermat number is significantly more complex than arithmetic modulo Mersenne numbers . In this case a k-bit word is used to represent the integers from 0 to $2^k - 1$. The problem then arises of how we shall represent 2^k modulo M_f . There are two ways to overcome this difficulty, one is to allow for an error to occur in the calculations, that is when 2^k is encountered in the data it is rounded to 0 or -2 . As k increases the probability of such occurrence is small, in fact Agarwal[22,1974] has discussed in detail the hardware implementation of modulo arithmetic for Fermat transforms using the above idea. The second approach is by using k+1-bits and

p	Mp	Length	Root	Max Length	Word length
3	7	3	2	6	3
3	7	6	-2	6	5
5	31	5	2	30	5
5	31	10	-2	30	5
7	127	7	2	126	7
7	127	7	-2	126	7
9	511	9	2	510	9
9	511	18	-2	510	9

TABLE 3.1 *Parameter for Several Mersenne Number Transform*

t	k	M_f	Length	Root	Max Length	Root/Max Length	Word Length
2	4	$2^4 + 1$	8	2	16	3	5
2	4	$2^4 + 1$	16	$\sqrt{2}$	16	$\sqrt{2}$	5
3	8	$2^8 + 1$	16	2	256	3	9
3	8	$2^8 + 1$	32	$\sqrt{2}$	256	3	9
4	16	$2^{16} + 1$	32	2	65536	3	17
4	16	$2^{16} + 1$	64	$\sqrt{2}$	65536	3	17
5	32	$2^{32} + 1$	64	2	128	$\sqrt{2}$	33

TABLE 3.2 *Parameter for various Fermat Number Transform*

various data code translation to simplify the practical implementation . Mc Clellan[28,1976] has described a new binary code for the integers modulo M_f . Given a binary representation of k+1-bit s for a data as $A = [a_{k,\ldots,a_0}]$, the new code is described as follow :

$$If \quad a_k = 1 \qquad then A = 0$$

$$\text{if } a_k = 0 \text{ then}$$

$$A = d_{k-1} \cdot 2^{K-1} + d_{k-2} \cdot 2^{k-2} + \ldots\ldots$$

where

$$d_j = \begin{cases} 1 & \text{if } a_j = 1 \\ -1 & \text{if } a_j = 0 \end{cases}$$

This number representation provides a binary arithmetic modulo M_f for negation, addition and multiplication by integer power of 2 . This new arithmetic is shown to be similar to and as complex as One's Complement arithmetic . One problem with this coding technique is that performing arithmetic with this additional bit is difficult . Leibowitz[29,1976] has discussed another coding technique of less complexity than that of McClellan . In order to overcome the problem mentioned above, a modified binary number system was used . In order to avoid arithmetic with extra bit, he allowed the additional bit to be a "1" only, when the number to be represented is "0" . This is achieved by subtracting "1" from the normal binary representation .

In summary, using any techniques described above for arithmetic modulo a Fermat number is similar to the one's complement and that of Mersenne numbers . However, we shall describe a different approach for performing modulo arithmetic on these numbers in later chapters .

In general, the Fermat Number Transform have two principle advantages over Mersennen Number Transform :

1) The Fermat Number Transforms permits much more flexibility in selecting the transform length as a function of word length than Mersenne transforms.

2) It is possible to evaluate the Fermat transforms with a fast transform type algorithm similar to that of FFT .

3.3. OTHER NUMBER THEORETIC TRANSFORMS

Mersenne and Fermat transforms were presented in the previous section, and have been used for several digital signal processing algorithms especially the convolution of two sequences . Each transform possesses the circular convolution property and has the arithmetic operations of addition, shift, and complement . The primary advantages of these transforms is the exact calculation . All operations are performed in a Finite Field of integers with the arithmetic carried out modulo a prime integer M . This structure causes a rigid relationship between the transform length and the word size . However, Fermat transforms have the advantages of utilising a fast transform algorithm, where Mersenne transforms have the advantages of very efficient arithmetic . Several methods have been proposed to overcome some of the difficulties while retaining the advantages of these transforms . Agarwal and Burrus [30,1974] showed that the word length restriction and hence transform length will be reduced by using 2-dimensional convolution . In this case, cyclic convolution of length $N = N_1 \cdot N_2$ can be implemented using a 2-dimensional Fermat transform defined similarly to the equations (3.1.2.2, 3.1.2.3) :

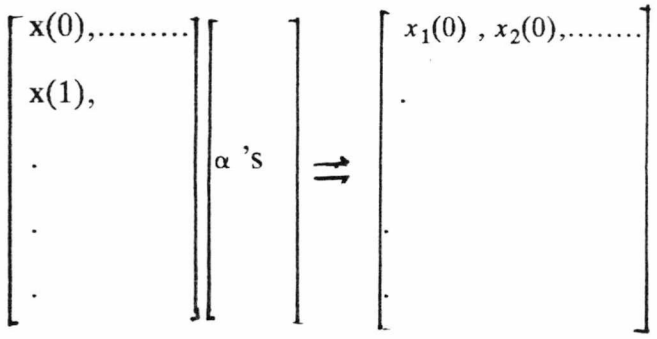
$$X_f\ (k_1\ ,\ k_2) = \sum_{n_1}^{N_1} G(n_1 k_1)\ .\ \alpha^{n_1 k_1}$$

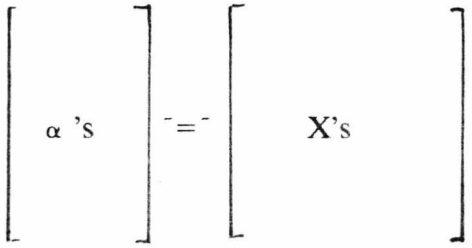
$$Where\quad G(n_1,k_1) = \sum_{n_2}^{N_2} x(n_1,n_2)\ .\ \alpha^{(n_2 k_2)}$$

$$x(n_1 , n_2) = N^{-1}\sum_{k_1}^{N_1}-1 \; F(n_1 k_1) \cdot \alpha - n_1 k_1$$

$$Where \quad F(n_1,k_1) = \sum_{n_2}^{N_2}-1 X_f(k_1,k_2) \cdot \alpha^{-n_2 k_2}$$

The 2-dimensional transform and its inverse can be taken in either order, however for computational advantages it is best to take the transform along the column of the input matrix and then for the final result along the rows . This procedure can be shown graphically as follow ;





A consequence of this technique is, the word length required now is proportional to the square root of the transform length rather than the transform length itself . Unfortunately, this technique will achieve the result at the expense of increased requirements for computation and storage .

In order to improve some of the difficulties which have arisen in transforms defined so far, we shall introduce other transforms which will achieve a higher computational efficiency and an increase in transform lengths in the next sections .

3.3.1. COMPLEX NUMBER THEORETIC TRANSFORM

Complex convolution arises in many areas e.g. radar, modems ,etc , in this case it

is necessary to define complex transforms .It also greatly influences the transform length when real convolution is used . Reed and Truong [24,1975] have investigated the general case of complex Number Theoretic transform and in specific Complex Mersenne Transform over GF(M_p^2) . They have shown that complex transforms which support the circular convolution can be defined modulo $M_p = 2^p - 1$ for any length N such that ;

$$N / (q^2 - 1)$$

For any transform of length N, with $N = 2^{p+1}$, the roots are given by ;

$$\alpha = a + Jb$$

This specifies relatively large transform lengths with arithmetic operative in One's complement and a fast transform type algorithm . However, the root are not simple and need some general complex multiplications .

Nussbaumer [31,1976] further introduced the concept of the complex transform defined in the ring, modulo M_p . In this ring with $M_p = 2^p - 1$, 2 and -2 are roots of order p and 2p respectively, corresponding to transform length of p and 2p . Since M_p is a prime , 2^d and and -2^d are also roots of order p and 2p provided "d" is not a multiple of M_p . This implies that 2J and 1 + J are roots of order 4p and 8p respectively . Table 3.3 shows some data relates to Complex Mersenne Transform . Under these conditions, a Complex Mersenne transform of length 8p and word length of p-bits which supporting circular convolution can be defined by ;

$$X_m(k) = \sum_{n=0}^{8p-1} x(n) \cdot (1+J)^{nk}$$

$$x(n) = R \sum_{k=0}^{8p-1} X_m(k) \cdot (1+J)^{-nk}$$

Where $R \cdot 8p \equiv 1 \ modulo M_p$.

Since the transform length is no longer a prime, the calculation of these transforms can be partly simplified by fast transform type algorithm . Using either decimation in time

p	M_p	Trans.Length/α =2J	Trans.Length/α =1+J	Word Length
3	7	12	24	3
11	2047	44	88	11
13	$2^{13}-1$	52	104	13
15	$2^{15}-1$	60	120	15
17	$2^{17}-1$	68	136	17
19	$2^{19}-1$	76	152	19
21	$2^{21}-1$	84	168	21
23	$2^{23}-1$	92	184	23

TABLE* 3.3 *Parameters related Complex Mersenne Number Transform

or decimation in frequency, the transform defined above can be decomposed into p eight point transforms :

$$X_m(k) = \sum_{n=0}^{7} x(p_n) \cdot (1+J)^{p_n k} + (1+J)^k \sum_{n=0}^{7} x(p_n+1) \cdot (1+J)^{p_n k}$$

$$+ \ldots\ldots + (1+J)^{(p-1)k} \sum_{n=0}^{7} x(p_n+p-1) \cdot (1+J)^{p_n k}$$

This decomposition technique will reduce the number of real operations . Therefore, we conclude that a complex transform can reduce the number of operations and at the same time provide a fairly large transform length . A similar argument is also true in the case of Complex Fermat Transforms .

3.3.2. PSEUDO TRANSFORMS

Another alternative to ease the question of transform length and word length is the use of another concept known as Pseudo Transforms . The idea of Pseudo Transform is similar to the idea explained in chapter three for computing the convolution using the residue arithmetic .

In this section we first explain the Pseudo Fermat Transform and expand the idea further to include the complex pseudo transforms . As explained, one of the restriction of using the FNT's is their rigid relation to the word length . Various solutions to this problem have been considered such as segmenting words into smaller blocks, or using two different modulo M_1 , M_2 which the final output can be obtained with the aid of CRT [Agarwal,23,1975] . However a more direct solution could be obtained if Number Theoretic Transform could be defined for $M_f = 2^k + 1$ and $k \neq 2^t$. In this case, let us assume that the prime factorisation of M_f is given by ;

$$M_f = M_1 , M_2 , ... , M_i$$

Thus, an N-point transform having the circular convolution can be defined in the ring of integer modulo M_f , provided the N-point transform can be defined separately in the field of M_1 , ... , M_i . This leads to the condition for the existence of N-point transform , such that N must simultaneously divide $M_1 - 1$, $M_2 - 1$, ... , $M_i - 1$. In this case, if k is even the maximum transform length would be equal to 2 , therefore , k must be a power of 2 to provide better transform length [Agarwal,23,1975] . Hence, it seems that we managed to reduce the word length, but we are faced with the fact that the transform length is sill small .

Nussbaumer [32,1977], used the same idea , but changed it completely to facilitate the long transforms as well as short word length . In this case the Number Theoretic Transform is defined in a ring of submultiple of M_p rather than M_p , and calling them Pseudo Fermat Number Transforms . We will restrict our definition of these transforms with a root of power of 2 [Nussbaumer,6,1981] as follow ;

$$X_f(k) = \sum_{n=0}^{N-1} x(n) \cdot 2^{wnk} \quad modulo \quad M_f / M_i$$

$$x(n) = Q \sum_{k=0}^{N-1} X_f(k) \cdot 2^{-wnk} \quad modulo \quad M_f / M_i$$

Where $Q \cdot N \equiv 1 \qquad$ modulo M_f / M_i

It can be seen that these transforms have a similar structure to the one of the Fermat Number Transforms . But the corresponding word length for a specific transform length has been reduced . One limitation or rather one can say a practical deficiency is that, performing transform modulo M_f / M_i could be difficult and more importantly the corresponding arithmetic circuit could be more complex than the case of arithmetic modulo M_f . To circumvent this difficulty, it is possible to operate the transforms modulo M_f , with the final result obtained by performing the last operation modulo M_f / M_i . Table 3.4 shows the transform lengths and roots for various k , when k is even .

k	Modulo	Trans.Length	Root	Word Length	Prime Factor
20	$\frac{2^{20}+1}{17}$	40	2	16	17.61681
22	$\frac{2^{22}+1}{5}$	44	2	19	5.397.2113
24	$\frac{2^{24}+1}{257}$	48	2	16	97.257.673
26	$\frac{2^{26}+1}{5}$	52	2	24	265.157.1613
34	$\frac{2^{34}+1}{5}$	68	2	32	685.953.26317
38	$\frac{2^{38}+1}{5}$	76	2	36	1145.457.525313
44	$\frac{2^{44}+1}{17}$	88	2	40	17.353.2931542417

TABLE 3.4 Parameters for various Pseudo Fermat Number Transform

Similar transform pairs can be defined for the Mersenne Number Transforms . Again the PMNT holds the same conditions as have been set up for the PFNT . However, if one studies the transform lengths, roots and word length, they are not much different to those of PFNT and in some cases even worse .

The idea of Pseudo Transforms can go further to develop the Complex Pseudo Transforms . The existence of Complex Pseudo Fermat Transform can be demonstrated by considering an N terms PFNT defined in the ring of M_f / M_i , with a root 2^w of order N .

If N and w are odd, then the condition N . w = 2k implies that N is even and N / 2 is odd . That is ;

$$((-2)^{w})^{N/2} \equiv ((-2)^{wd})^{N/2} \equiv 1 \quad modulo \quad M_f / M_i$$

provided d and k have no common factors [Nussbaumer,32,1977] . In this instance $(2J)^{w}$ is a root of order 2N and $(1+J)^{w}$ is a root of order 4N . The Complex Pseudo Fermat Number Transform pair can be defined as :

$$X_f(k) = \sum_{n=0}^{4N-1} x(n) \, . \, (1+J)^{wnk} \quad modulo \quad M_f / M_i$$

$$x(n) = (4N)^{-1} \sum_{k=0}^{4N-1} X_f(k) \, . \, (1+J)^{-wnk} \quad modulo \quad M_f / M_i$$

It is also possible to find other values for root α , but these values have no simple structure . Various options for CPFNT are listed in table 3.5 [Nussbaumer,32,1977]. It can be seen from table 3.5 that Fast Fourier Transform algorithm does not necessarily increase the efficiency of the transform, because N is not highly factorisable .

3.4. WALSH TRANSFORMS

In previous sections we introduced several Number Theoretic Transforms modulo Fermat and Mersenne numbers which possess the convolution property . The major

problem with these types of transforms has been that the transform length is dependent upon the modulo and the root . There are variations as indicated in tables 3.1-5, to circumvent these problems . But in each variation there was always a desirable feature which was compromised .

If all the desirable features of each number theoretic transform were considered, then the ideal number theoretic transform could be characterised by the following ;

1) Arithmetic modulo a Mersenne number .

2) Operations consist of additions and bit shift .

3) Having a fast type algorithm .

4) The convolution property .

The major obstacle in simultaneously satisfying all the above criteria has been the convolution property . It is this property which determines the dependency of the transform length, the modulo , and the root of unity [def.3.1.1] . If the convolution property is considered to be optional , then another class of transforms can be defined for which the kernel will be of a rectangular nature . This class of transforms are better known as Walsh Transform . A finite Walsh Transform pair can be defined as follow [Elliott,25,1982] ;

$$X_w(k) = \sum_{n=0}^{N-1} x(n) \; . \; wal(n,j) \quad moduloM$$

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X_w(k) \; . \; wal(k,j) \quad modulo \quad M$$

where wal(n,j) and wal(k,j) are the Walsh functions with the property of ;

$$\sum_{j=0}^{N-1} wal(n,j) \; . \; wal(k,j) = \begin{matrix} 1 & n=k \\ 0 & otherwise \end{matrix}$$

Because the Walsh functions are binary valued, their generation and implementation are simple . These functions can be rearranged in several ways a) sequency order b) natural order and c) dyadic order [Elliott,25,1982] in order to provide the kernel for the Finite Walsh Transform. Because of the characteristic of Walsh functions, and the

k	Modulo	Trans.Length	Root	Prime Factor
15	$\frac{2^{15}+1}{9}$	40	2(J-1)	99.331
25	$\frac{2^{25}+1}{33}$	200	J+1	33.251.4051
27	$\frac{2^{27}+1}{81.19}$	216	J+1	81.19.87.211
33	$\frac{2^{33}+1}{9}$	88	2(J-1)	603.683.20857
35	$\frac{2^{35}+1}{33}$	56	-4(J-1)	33.43.281.86171
45	$\frac{2^{45}+1}{171}$	40	16(J+1)	11.171.331.18837001
49	$\frac{2^{49}+1}{129}$	392	J+1	129.4363953127297

TABLE 3.5 *Various Parameters for Complex Pseudo Fermat Transform*

fact that the elements of the kernel are 1 , M-1 , it is implied that

the α is a root of order 2 . Hence the maximum transform length is equal to two in order to support circular convolution property . Therefore one can say that, the Finite Walsh Transform does not preserve the convolution property . However, an arbitrary transform length Finite Walsh transform can be defined which possess the dyadic convolution property [Elliott,25,1982], and cannot be used to perform linear convolution . Hence, the Finite Walsh Transform over finite field produces a structure similar to the traditional Number Theoretic Transform except it cannot possess the cyclic convolution property. The only constraint imposed on the Finite Walsh Transform is that the transform length must be relatively prime to M in order to take fully the advantages of modular arithmetic and Eular's Theorem .

3.5. COMPUTATIONAL COMPLEXITY

In previous section we presented a number of discrete digital transforms each capable of performing the convolution of two sequences in the frequency domain . In this section we shall study some of the advantages and disadvantages of these transforms over its rival Fast Fourier Transform .

For many practical purposes, comparing various algorithms of such complexities is very tedious and almost impossible in the loose sense . However , it is somewhat possible to compare these algorithms with the aid of complexity theory . The idea of complexity theory is due to A.L.Toom who used it to show how fast we can multiply?, and later S.A.Cook showed how Toom's method can be used to calculate the minimum computation time of a function . This is known as Toom-Cook complexity algorithm [Knuth,1,1981] . We shall use their idea to evaluate the computational complexity corresponding to the transform implementation of convolution of two sequences see figure 3.1 and in a later chapter we shall show how these can be extended further to include the implementation of digital adaptive filters . For the ease of the argument at the moment, we assume that the hardware cost of arithmetic modulo Mersenne and

Fermat numbers are the same .

In this thesis we shall refer to computational complexity as the number of arithmetic operations performed by these transforms for an arbitrary sequence length. The number of arithmetic operations related to computation of a specific transform depends essentially on two factors, addition and multiplications . However there might be other factors involved e.g. word length, but these are somewhat proportional to the number of addition and multiplication . Below, we summerise a number of formulae for the evaluation of the addition and multiplication for a number of transforms . Also we assume that one of the input sequences has a length L, which is true for the case of digital filters .

1)Mersenne Number Transform

These transforms suffer from the lack of fast transform algorithms, and therefore the number of additions and multiplications (shifts when $\alpha = 2$) are proportional to the transform length N [21].

$$\frac{3 \, . \, N \, . \, (N - 1)}{N - L}$$

real additions

$$\frac{3 \, . \, (N - 1)^2 + N}{N - L}$$

real multiplications(shifts)

2) Fast Fourier Transform

the number of additions and multiplications for FFT is given by [12] ;

$$\frac{3 \, . \, (k + 1) \, . \, (r - 1) \, . \, N \, . \, \log_r^N + 2 \, . \, N}{N - L}$$

real additions

$$\frac{3 \, . \, k \, . \, (r - 1) \, . \, 4 \, . \, N \, . \, \log_r^N + 4 \, . \, N}{N - L}$$

real multiplications

Since the Fermat Number Transform supports the fast transform type algorithm, it is possible to use the FFT formulae for the derivation of the complexity formulae for FNT. A modification is done to FFT and from that the required formulae for number of additions and multiplications for the FNT, CFNT and CPFNT are developed.

3) Fermat Number Transform

Unlike MNT's, Fermat transforms will support the fast transform algorithms such as FFT . Let $N = r^n$ be the transform length, where r is the base (radix) and n is any integer . We shall evaluate the number of additions and multiplications as follows;

$$\frac{3 \cdot (r-1) \cdot N \cdot \log_r^N}{N-L}$$

real additions

$$\frac{3 \cdot k \cdot (r-1) \cdot N \cdot \log_r^N + N}{N-L}$$

real multiplications

Where k depends on the symmetries of the root of unity α e.g. k=1 if r is odd and $k = \frac{1}{2}$ if r is equal 2 . If however, $\alpha = 2$ the multiplication in transform will be reduced to only bit shift ;

$$\frac{3 \cdot (r-1) \cdot N \cdot \log_r^N}{N-L}$$

real additions

$$\frac{N}{N-L}$$

real multiplications

$$\frac{3 \cdot k \cdot (r-1) \cdot N \cdot \log_r^N}{N-L}$$

shifts

For $\alpha = \sqrt{2}$, the number of shifts and multiplication is the same , but the number of additions will increase . This is due to the fact that multipling by odd powers of α need

an extra addition $\sqrt{2} = 2^{k/4} \cdot (2^{k/2} - 1)$, hence ;

$$\frac{3 \cdot (r-1) \cdot N \cdot (\log_r^N + \frac{1}{4}}{N - L}$$

Let us assume that our two sequences are complex values rather than real values therefore, Using the FNT with α is not a power of two we need ;

$$\frac{6 \cdot (r-1) \cdot N \cdot \log_r^N}{N - L}$$

real additions

$$\frac{6 \cdot k \cdot (r-1) \cdot N \cdot \log_r^N + 4 \cdot N}{N - L}$$

real multiplications

The reason for the extra arithmetic operations is that, we compute the transforms of real and imaginary of the sequences separately . However , with the kernel being a power of two, then ;

$$\frac{6 \cdot (r-1) \cdot N \cdot \log_r^N}{N - L}$$

real additions

$$\frac{6 \cdot (r-1) \cdot N \cdot \log_r^N}{N - L}$$

shifts

$$\frac{4 \cdot N}{N - L}$$

real multiplications

4) Complex and Pseudo Fermat Transforms

In the case of complex and pseudo Fermat transforms with complex roots, we note that ;

$$2^k = M_f - 1 \equiv -1$$

Hence, $J = \sqrt{-1}$ can be represented in the Fermat number ring by $2^{k/2}$. Since for both

the cases the transform length is composite , the number of additions and multiplications is given by ;

$$\frac{3 \cdot N \cdot (\log_2^{N_1} + N_2 - 1)}{N - L}$$

$$\frac{3 \cdot N \cdot (\log_2^{N_1} + N_2 - 1 + \frac{1}{4}}{N - L}$$

real additions

for the case of $\alpha = 2J$, $1+J$ respectively .

$$\frac{3 \cdot N \cdot (\frac{1}{2} \cdot \log_2^{N_1} + N_2 - 1}{N - L}$$

shifts

$$\frac{N}{N - L}$$

$$\frac{N + N \cdot N_2^2}{N - L}$$

real multiplications

for $\alpha = 2J$, $1+J$ respectively.

Where N_1 and N_2 are the factors of N and equal to 4p and 8p for $\alpha = 2J$, $1+J$ respectively . The similar formulae are true for Complex and Pseudo Mersenne Transforms .

Summary

It can be concluded from the formulae that, for a given transform length N , the optimum computing efficiency can be achieved for $L = N/2$ in many practical cases . Table 3.6 and 3.7 shows the computational complexity for various Number Theoretic Transforms . A first indication on the relative number of additions and multiplications of various transforms shows that the largest arithmetic operations correspond to MNT

and at the lowest limit lies the FNT and CPFNT . It can also be seen that the FNT and CPFNT compares very favourably to that of the Fast Fourier Transform . However, each transform has its own drawbcak . In case of the FFT one of the disadvantages is that it needs memory spaces for storing the values of the trigonometric functions for the multiplications, and as the transform length increases so does the number of storage . Where as in the case of NTT the values of kernel is a power of two and hence, the general multiplications which greatly influences the performance of a system will be reduced to only shifts and therefore there is no need for storage of such quantities .

For the case of MNT, even though the multiplications are reduced to only shifts , the number of arithmetic operations is still too large for a given transform length compared with other transforms . This is because it dose not support the fast type algorithm . Where as the cases of CMNT and CFNT and PMNT and PFNT can be partly computed by a fast algorithm technique . Hence, they have a moderately small number of arithmetic operations compared to MNT but larger than FNT at the expense of using complex kernel for a given length, but the transform length is still small .

Hence, the most promising of all are the FNT and CPFNT . However , these transforms have their own limitations . The major disadvantages of using the FNT's is the word length . The number of bits required to represent the M_f is equal to N/2 and N/4 for $\alpha = 2$, $\sqrt{2}$ respectively, hence as N increases so does the word length . In this respect, the CPMNT and CPFNT provide a significant improvement in the word length . The reason for this descent is because the M_p or M_f can be factorised into small prime numbers. And each modulo can be represented by only a few bits . The advantages of reducing the word length while retaining the modest transform length and greater flexibility and some cases the speed, is offset by one major disadvantage . Because various operations are performed modulo M_p / M_i or M_f / M_i, the corresponding arithmetic circuits are obviously much more complex than arithmetic modulo M_p *or* M_f . This difficulty can be circumvented by computing modulo M_p *or* M_f and obtain the final result by performing the last operations modulo M_f / M_i *or* M_p / M_i . This

obviously increases the computational complexity of the algorithm as well as increasing in the word length .

However, it is possible to reduce the number of bits used in the Fermat Number Transform further, by taking the argument used in the PFNT and CPFNT . This way by choosing $2^{k-1} < M < 2^k$ where M is prime, we can define another set of Fermat Transforms which will support the Euler's theorem for achieving the maximum transform length while having smaller number of bits for processing . The advantages in this instance are highly composite and therefore can be implemented more efficiently by using the mix radix technique . As an example Let M = 163 which is a prime number, with a word length of 8 bits and $N_{max} = 162 = 2 . 3^4$. Therefore a 162 point radix 2 and 3 Fermat Number Transform with 8-bit wordlength can be achieved . A longer transform length also can be achieved by the same mechanism . E.g M = 64153, a 16-bits 13 stage Fermat Transform of length $N = 2^{10} . 3^2 . 7$ is possible . However, in this case the root α may not be a power of two and hence need a more general multiplications . So , in choosing an algorithm for transformation , there is an element of compromise between multiplications, transform length ,word length and shifts .

In general, there are two factors which greatly influence the choice between these transforms

1) Applications

2) Technology

In most application areas where digital adaptive filters are used, the required word length is between 8-16 bits . Therefore the later case seems to be more favourable than other techniques such as CPFNT where the number of bits are still too large .

Transform	ADD	MULT.	ADD	SHIFT	ADD	ADD $\alpha=\sqrt{2}$	SHIFT	MULT. $\alpha=$
Length	FFT	FFT	MNT	MNT	FNT	FNT	FNT	FNT
32	49	34	186	180	30	32	17	17
64	58	40	390	372	36	38	20	20
128	67	46	762	756	42	44	23	23
256	76	52	1530	1524	48	50	26	26
512	85	58	3066	3060	54	56	29	29
1024	94	64	6138	6132	60	62	31	31

TABLE 3.6 *Number of arithmetic operation per output sample for various Transforms*

Transform	ADD	SHIFT	ADD	SHIFT	ADD	SHIFT
Length	CMNT	CMNT	PFNT	PFNT	CPFNT	CPFNT
40	•	•	42	33	•	•
44	72 $\alpha=2J$	66	72	66	•	•
68	108 $\alpha=2J$	102	108	102	•	•
88	79 $\alpha=1+J$	69	78	69	79 $\alpha=1+J$	69
76	120 $\alpha=2J$	114	120	114	•	•
136	115 $\alpha=1+J$	105	•	•	•	•
200	•	•	•	•	163 $\alpha=1+J$	153
392	•	•	•	•	307 $\alpha=1+J$	297

TABLE 3.7 *Number of arithmetic operations per output sample for various Transforms*

• not given

CHAPTER 4

FREQUENCY ADAPTIVE DIGITAL FILTERING

INTRODUCTION

Frequency-domain adaptive filters have been considered by many authors [100-103], however, the approach varies considerably. For example, Clark [100,1981] has used the transform technique mainly to decrease the amount of computation needed in the convolution and correlation stages of the adaptive filters. There the approach was to update the filter weights in the time domain. Another approach was taken by Mansour [101, 1982], where the filter weights are updated in the frequency domain, while the error is obtained in the time domain. That is, after obtaining the error sequence in the time domain, it then converted the error sequence back into the frequency domain by using an FFT. The more direct approach of implementing the adaptive filters in the frequency domain was taken up by Reed [102, 1981] after the work of Dentino [103, 1978]. There, the calculation of the error and the updating of the filter coefficients is all done in the frequency domain stage rather than the previous work of Clark and Mansour.

The approach taken in this thesis is after the work presented by Dentino and Reed. However, the Dentino and Reed work does not cover complete mathematical analysis of the frequency adaptive filters, and does not at all consider the use of Number Theoretic Transforms of frequency adaptive filters and in particular when the NTT is used.

A block diagram of the frequency domain adaptive digital filter is shown in figure 4.2. The input signal x(n) is accumulated in a buffer memory to form an N-point data block, which can then be transformed. The result of each bin in the transform domain (which is the multiplication of the filter coefficient and the input data) is then subtracted from the desired response coefficient to produce an error which is then used to update the filter coefficient.

4.1. ADAPTIVE FILTERING

The adaptive digital filter discussed here is of the Least-Mean-Square (LMS) type presented by Widrow et.al [33,34,41] for which the performance index is the Mean-

Square-Error. All inputs are assumed to be real. The adaptive filter of Widrow is a Finite Impulse Response (FIR) digital filter of order N-1, for which the output, y, at a discrete time instant, k, is given as the convolution sum of the input, x, and the filter weights w :

$$y_k = \sum_{i=0}^{N-1} w_i \cdot x_{k-i} \tag{4.1.1}$$

The Widrow-Hopf LMS algorithm adjusts the filter weights in accordance with equation (4.1.2) :

$$w_{k+1} = w_k + 2\mu e_k \cdot x_k \tag{4.1.2}$$

where μ is the convergence constant, and x, w are the input vector and the weight vector respectively :

$$w_k = [w_{0,\ldots\ldots,} w_{N-1}]$$

$$x_k = [x_{0,\ldots\ldots,} x_{N-1}]$$

and e is the error at the k-th instant given by the difference between the desired output d and the actual output y :

$$e_k = d_k - y_k \tag{4.1.3}$$

From an implementation point of view, the adaptive filter consists of two major operations, a convolution to produce the outputs y, and the LMS algorithm to adjust the filter weights, as illustrated in figure 4.1.

The main purpose of this thesis is to present the analysis of the frequency adaptive filters.

Therefore, in this respect, the error in the frequency domain can be defined as follows :

$$|E^J|_M = |D^J|_M - |Y^J|_M \tag{4.1.4}$$

Where D^J is the desired response in the transform domain and Y^J is the output of the

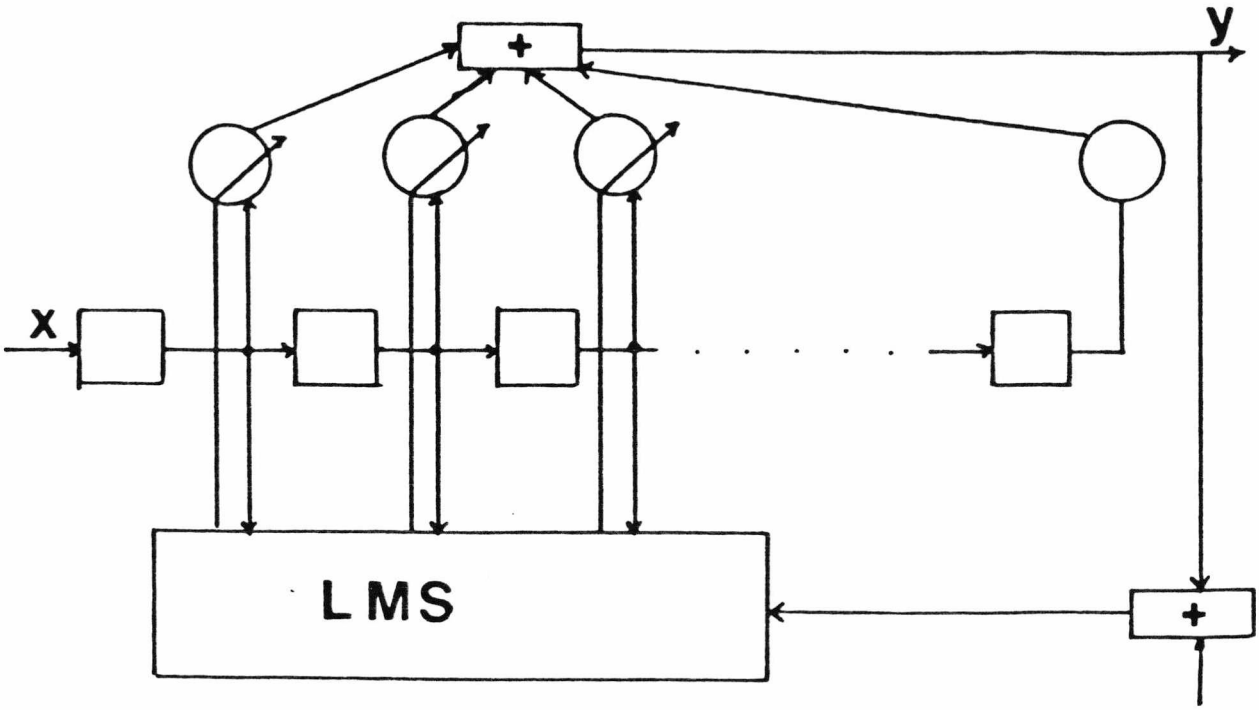


Figure 4.1 Time domain adaptive digital filter

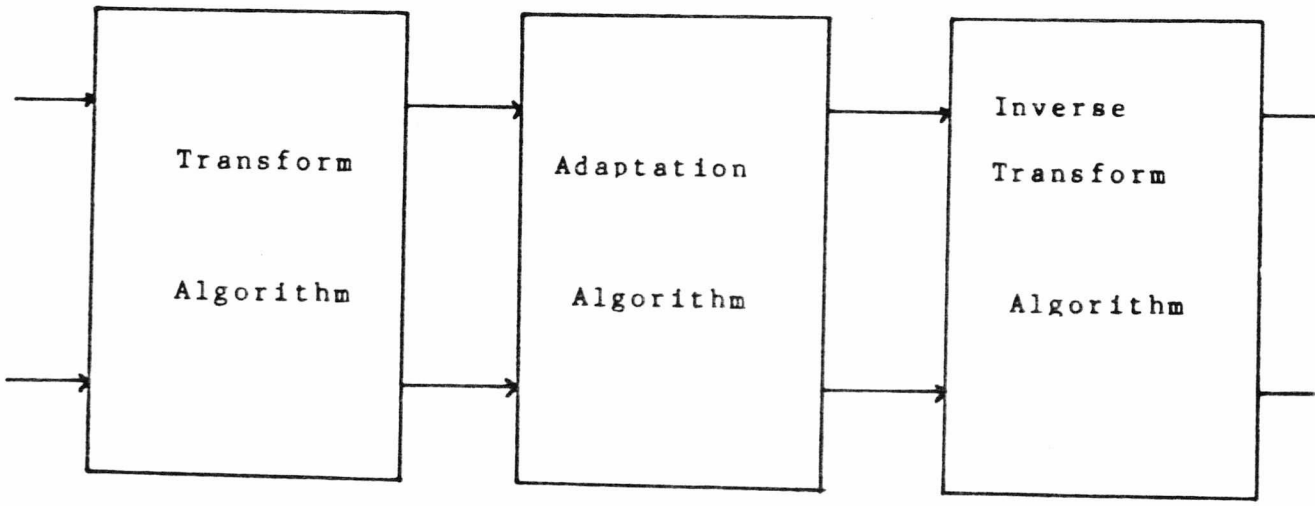


Figure 4.2 General configuration of frequency ADF

filter.

4.2. FREQUENCY WIENER FILTERING PROBLEM

Wiener filtering is, of course, the basis of the LMS adaptive filtering, as the adaptive filter converges in the mean to the Wiener solution[33,34,42]. Wiener filtering can be done in frequency, see Fig 4.2, using the following definition, along with the assumption that all the inputs are stationary. Let:

$$|D^j| = [|D^0|_M, \ldots\ldots, |D^{N-1}|_M] \tag{4.2.1}$$

be the Nx1 vector of desired response and let,

$$|E^j| = [|E^0|_M, \ldots\ldots, |E^{N-1}|_M] \tag{4.2.2}$$

be the Nx1 vector of errors where E is defined in Eq(4.1.4).

4.2.1. FREQUENCY MEAN SQUARE ERROR

The key element in the analysis of the LMS in the time domain is the mean-square error. The same is true for the LMS in the frequency domain, but with a difference. As for the time domain analysis, the square of the expectation of the error is used for the analysis, however, this can be calculated in the frequency domain and is equivalent to the multiplication of the frequency component by its conjugate value, as for the case of DFT (Parseval's Theorem). This is clearly not true for the NTT as it is purely real. However, using the properties of the NTT [Agarwal,22], it is possible to define an equivalent square of the error in the NTT domain.

Frequency mean square error (FMSE) is defined by:

$$FMSE = \Xi = \epsilon |E^{\tau j}|_M \cdot |E^j|_M \tag{4.2.1.1}$$

where τ denotes the transpose inverse, and ϵ represents the statistical expectation or ensemble average operation [35], and $|\ |_M$ represent the residue arithmetic operations. Clearly, the FMSE is the expected value of a smoothed estimate of the square error over one block. The following analysis is acceptable to both the NTT and DFT except for the DFT case, τ represents the complex conjugate and all the arithmetic is done in the conventional numbering system.

Using the Eqs (4.1.4),(4.2.2), Eq (4.2.1.1) becomes:

$$\Xi = \epsilon[\,|E^{\tau^j}|_M \cdot |E^j|_M = \epsilon[(|D^{\tau^j}|_M - |Y^{\tau^j}|_M)(|D^j|_M - |Y^j|_M)]$$

$$= \epsilon[\,|D^{\tau^j}|_M \cdot |D^j|_M] - \epsilon[\,|D^{\tau^j}|_M \cdot |X^j|_M \cdot |W^j|_M]$$

$$- \epsilon[\,|X^{\tau^j}|_M \cdot |W^{\tau^j}|_M \cdot |D^j|_M] + \epsilon[\,|W^{\tau^j}|_M \cdot |X^{\tau^j}|_M \cdot |X^j|_M \cdot |W^j|_M] \qquad (4.2.1.2)$$

The following correlation matrices are now defined:

$$|\phi_1|_M = \epsilon[\,|X^{\tau^j}|_M \cdot |X^j|_M]$$

$$|\phi_2|_M = \epsilon[\,|X^{\tau^j}|_M \cdot |D^j|_M] \qquad (4.2.1.3)$$

where $|\phi_1|_M$ and $|\phi_2|_M$ are the NxN input auto correlation and the Nx1 cross-correlation between the input and desired response respectively.

The matrix $|\phi_1|_M$ has got the following properties;

a) $|\phi_1|_M$ is symmetric

b) $|\phi_1|_M$ is positive definite

c) $|\phi_1|_M$ has N linearly independent eigenvalues and can always be reduced to diagonal form by a similarity transform.

Using these definitions, the FMSE, Eq (4.2.1.2) can be written as:

$$\Xi = \epsilon[\,|D^{\tau^j}|_M \cdot |D^j|_M - |\phi_2^{\tau}|_M \cdot |W^j|_M$$

$$- |W^{\tau^j}|_M \cdot |\phi_2|_M + |W^{\tau^j}|_M \cdot |\phi_1| \cdot |W^j|_M] \qquad (4.2.1.4)$$

Using the vector inner product, the third term in Eq, (4.2.1.4) can be written as $|\phi_2^{\tau}|_M \cdot |W^j|_M$, therefore Eq(4.2.1.4) becomes :

$$\Xi = \epsilon[\,|D^{\tau^j}|_M \cdot |D^j|_M] - 2 \cdot |\phi_2^{\tau}|_M \cdot |W^j|_M + |W^{\tau^j}|_M \cdot |\phi_1|_M \cdot |W^j|_M \qquad (4.2.1.5)$$

This is the same as the MSE [33] . Thus, FMSE can give similar results as the MSE, when all inputs are stationary. It follows, then, that the optimal set of filter weights W^{*j} for the frequency Wiener filter is similar to as it is for the Wiener filter [33,34,42] i.e:

$$W^* = |\phi_1^{-1}|_M \cdot |\phi_2|_M$$

The above development shows that the frequency Wiener filtering formulation can serve as a unifying framework for the Wiener problem, and the optimum solution W^{*j} of the frequency Wiener problem can give similar results to corresponding solutions of the Wiener problem.

4.3. FREQUENCY ADAPTIVE FILTERING AND FLMS ALGORITHM

In analogy to LMS adaptive filtering a frequency algorithm can be derived to sequentially solve for the Wiener weight vector, in real-time, by an implementation of the method of the steepest descent.

Figure 4.2 shows the general configuration. Because it is desired to keep the weights constant, while each block of data sequence is being processed, let the weight vector be adjusted once per data block sequence, rather than once per data sample, as with LMS algorithm. The algorithm then becomes:

$$|W^{j+1}|_M = |W^j|_M - \mu_F \cdot |\nabla_F^j|_M \tag{4.3.1}$$

where diagonal matrix μ_F is the convergence constant, ∇_F^j

is the Nx1 FMSE gradient, and W^j is the Nx1 weight vector. The gradient is taken with respect to the weights as follows:

$$|\nabla_F^j|_M = \frac{\epsilon[\,|E^{\tau^j}|_M \cdot |E^j|_M\,]}{W} \tag{4.3.2}$$

Because the computation of an ensemble average is difficult and impractical for this problem, an estimate of the gradient $|\nabla_F^j|_M$ is used in place of $|\nabla_F^j|_M$.

Define the frequency mean-square error gradient estimate as:

$$|\nabla_F^j|_M = \begin{bmatrix} \frac{\partial}{W_1}[\,|E^{\tau^j}|_M \cdot |E^j|_M\,] \\ \frac{\partial}{W_2}[\,|E^{\tau^j}|_M \cdot |E^j|_M \\ \vdots \\ \frac{\partial}{W_3}[\,|E^{\tau^j}|_M \cdot |E^j|_M\,] \\ \frac{\partial}{W_4}[\,|E^{\tau^j}|_M \cdot |E^j|_M \\ \vdots \end{bmatrix} \tag{4.3.3}$$

$$\begin{bmatrix} \vdots \\ \frac{\partial}{W_r}[\,|E^{\tau^j}|_M \cdot |E^j|_M] \\ \frac{\partial}{W_N}[\,|E^{\tau^j}|_M \cdot |E^j|_M \end{bmatrix}$$

Taking advantage of Eqs (4.2.1.3) and (4.2.1.5) and ignoring the expectation, the gradient estimate can be written as:

$$|\nabla_F^j|_M = -2 \cdot |D^{\tau^j}|_M \cdot |X^j|_M + 2 \cdot |W^{\tau^j}|_M \cdot |X^{\tau^j}|_M \cdot |X^j|_M$$

$$= -2 \cdot |X^j|_M \cdot [\,|D^{\tau^j}|_M - |W^{\tau^j}|_M \cdot |X^{\tau^j}|_M] = -2\,|X^j|_M \cdot |E^{\tau^j}|_M$$

or

$$|\nabla_F^j|_M = -2 \cdot |X^{\tau^j}| \cdot |D^j|_M + 2 \cdot |X^{\tau^j}|_M \cdot |X^j|_M \cdot |W^j|_M = -2 \cdot |X^{\tau^j}|_M \cdot |E^j|_M$$

Using this block gradient estimate in the weight adjustment algorithm Eq (4.3.1) gives the frequency least mean square (FLMS) algorithm:

$$|W^{j+1}|_M = |W^j|_M + 2 \cdot \mu_F \cdot |X^{\tau^j}|_M \cdot |E^j|_M \qquad (4.3.5)$$

Clearly, the weight update term is a correlation, implemented either with a parallel or with a serial processor.

4.4. CONVERGENCE PROPERTIES OF FLMS ALGORITHM

The convergence properties of interest in frequency adaptive filtering are the required bounds on the μ_F (convergence constant) and adaptation speed. The convergence constant must take on values in a particular range in order to insure convergence of the algorithm. Adaptation speed refers to how fast the FLMS is reduced to its lowest level ($\Xi_{\min}$). These convergence properties are examined below, one-by-one, for frequency adaptive filters, and compared with the convergence properties of LMS filters.

4.4.1. BOUNDS ON μ_F TO GUARANTEE CONVERGENCE

First, it must be prove that the FLMS algorithm converges. This proof is given in

Appendix (A). The approach taken is to show, that as the weight index j approaches infinity, the expected value of weight vector ($\epsilon[|W^{j+1}|_M]$) approaches the Wiener weight vector, under the assumption that $|X^j|_M$ and $|D^j|_M$ are ergodic. The proof also shows that the requirements on the convergence constant (μ_F for FLMS and μ for LMS) are similar. That is, μ_F and μ must take on values in a similar range in order to ensure the convergence.

The bounds on the convergence constants are:

For FLMS:

$$0<\mu_F<\frac{1}{|\lambda_{\max}|_M} \tag{4.4.1.1}$$

For LMS:

$$0<\mu<\frac{1}{\lambda_{\max}} \tag{4.4.1.2}$$

Where $\lambda_{\max}$ is the largest eigenvalue of matrix $|\phi_1|_M$.

The convergence proof in Appendix A is carried out in order to establish the mathematical model for FLMS convergence.

4.4.2. ADAPTATION SPEED

Adaptation speed is given in terms of a, " FMSE time constant " which indicates how fast the weight vector converges to the Wiener weight vector [33,34,42]. There are N time constants T_{pMSE}, one for each pth " mode " of the different equations describing the adaptation process. These time constants are derived in Appendix B. The derivations follow the form of the corresponding derivations for the LMS algorithm, but with a few differences. The convergence constant μ (LMS) is replaced by μ_F (FLMS). But more important, the time unit for adaptation for LMS is one time sample, whereas, the time unit for adaptation of (FLMS) is the data block sequence. Thus, the equation for the two different algorithms has the same form but different meaning. This has been resolved in Appendix B. The time constant t_{MSE} for LMS and T_{FMSE} for FLMS is given as follows:

$$t_{mse} = \frac{1}{[4 \cdot \mu \cdot R]} \quad T_{FMSE} = \frac{1}{[4 \cdot N \cdot \mu_F \cdot tr(|\phi_1|_M)]} \qquad (4.4.2.1)$$

Where R is the input power in time domain and $tr(|\phi_1|_M)$ is the input power in the transform domain .

4.4.3. FILTER SIMULATION

In order to verify the convergence properties of the FLMS algorithm, derived in previous sections, a computer simulation has been used .

The goal of the simulation is to apply the LMS and FLMS to two examples, and see that the mean square error will converge to an optimum solution .

In the first example the input signal which is a sinusoid of frequency f (sin(wt)), embedded in the white noise as shown in figure 4.3 is applied to an Frequency Adaptive Digital Filter (FADF). The output is compared with a desired response which is a sinusoid of the same frequency but different phase (sin(wt+T)), which is not coherent with the input signal. The error obtained is then used to update the filter coefficients according to equation 4.3.5.

In the second example the input signal is represented as a sinusiod of frequency f and 2f (sin(wt)+sin(2wt)) [Cowan,40,1980] this is shown in figure 4.4, is applied to the same FADF as used in the first example. The desired response is a sinusiod of frequency f but different phase (sin(wt+T)). The error obtained is then used for the weight update.

The adaptive filter which is used in the above two examples, is of length 4 and implemented using the configuration of figure 4.2 . Since the filter is implemented using the transform technique described in chapter 3, certain conditions regarding the choice of modulo and the root of unity must hold . In this simulation the Fermat Number Transform is used for the conversion of the input data into its transform domain . A number of steps were taken for the simulation;

1) The modulo used is M=17, so as to be in line with the VLSI design of the filter presented in chapter 6.

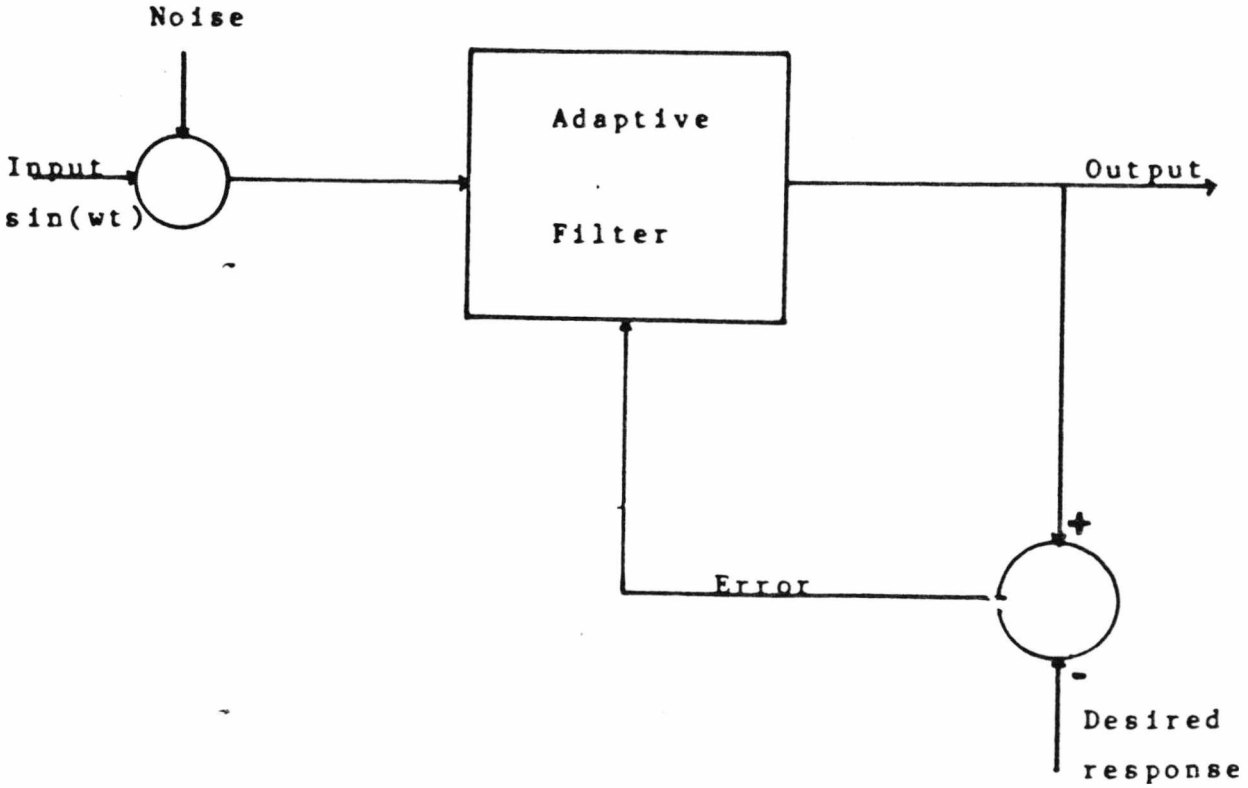


Figure 4.3

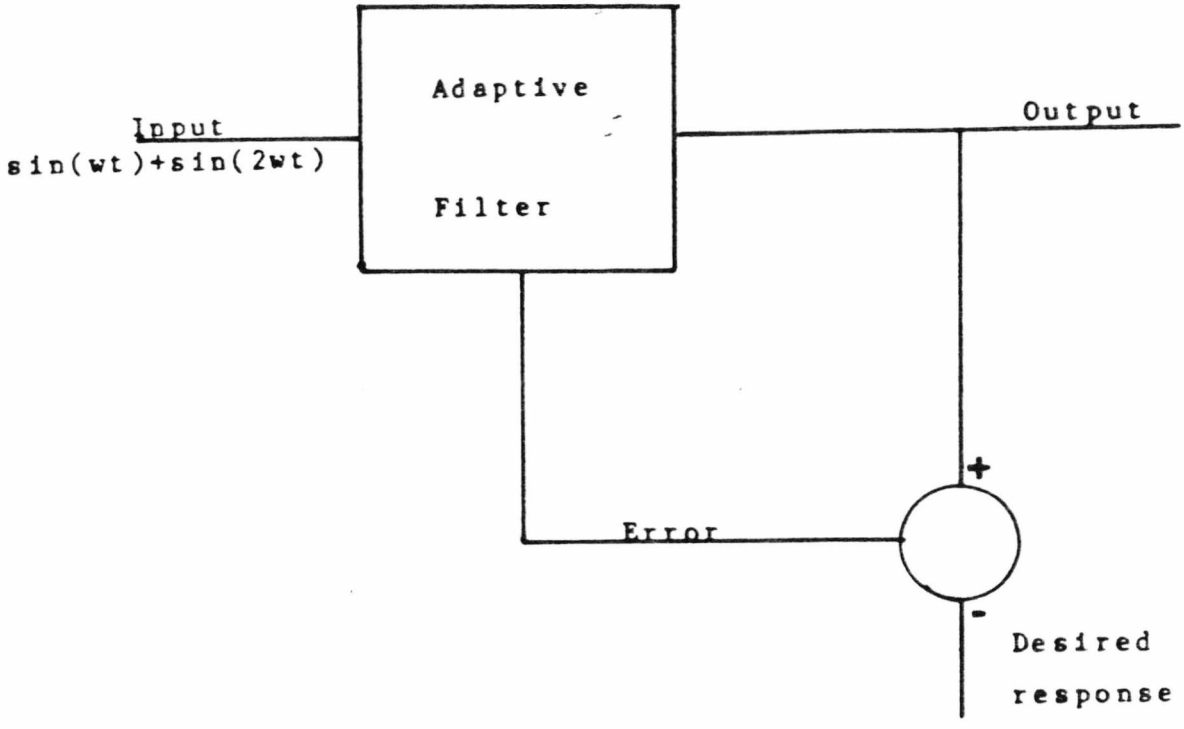


Figure 4.4

2) A 4-bit wordlength is used for the data.

3) The filter coefficients are originally set to an initial value of zero.

The simulation results are done for both frequency and time domain filters for variety of phases and various values of convergence constant μ_F. These results are shown in figures 4.5 to 4.18 for the first 200 algorithm iterations.

4.4.4. SIMULATION RESULTS

The results shown in figures 4.5-4.18 verifies that the Frequency-Mean Square Error (FMSE) measured at the filter output does converge to an " optimum solution" . The convergence takes place at different rates. This is because as the convergence constant is changed, so does the adaptation speed. This becomes apparent when we compare them with the TADF case. The above results verify that the FLMS is not in general, equal to the LMS. Examination of the above curves shows a cyclic behaviour of the FLMS values. This may be due to the quantisation. It was also noted that when the input and reference signals are 90 degrees out of phase [fig 4.8], the result was different to the others. These two may be due to the usage of short transform and word lengths simulation.

4.5. COMPUTATIONAL COMPLEXITY OF LMS AND FLMS ADAPTIVE FILTERS

The main computational efficiency issues involved in the algorithm implementation are the storage, time (machine cycle, I/O) and computational complexity measured by the required number of multiplications and additions . The first two issues are processor architecture dependent, and will be discuss in later chapters . This section will concentrate on the computational complexity required when using a standard processor .

4.5.1. COMPUTATIONAL COMPLEXITY OF LMS ADAPTIVE FILTERS

The convolution operation is done in the standard form ;

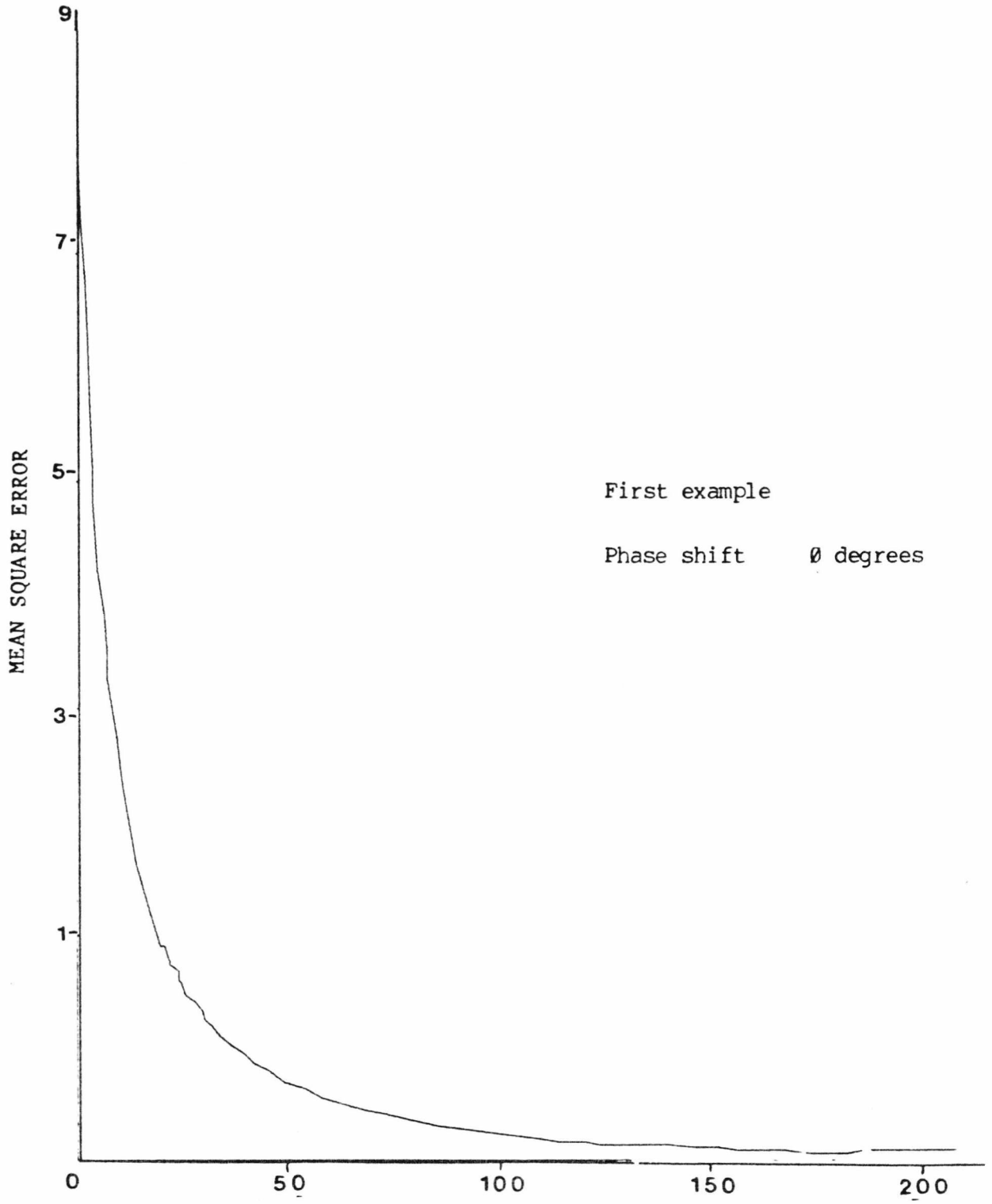


Figure 4.5 Mean Square Error vs No. of iterations for FADF

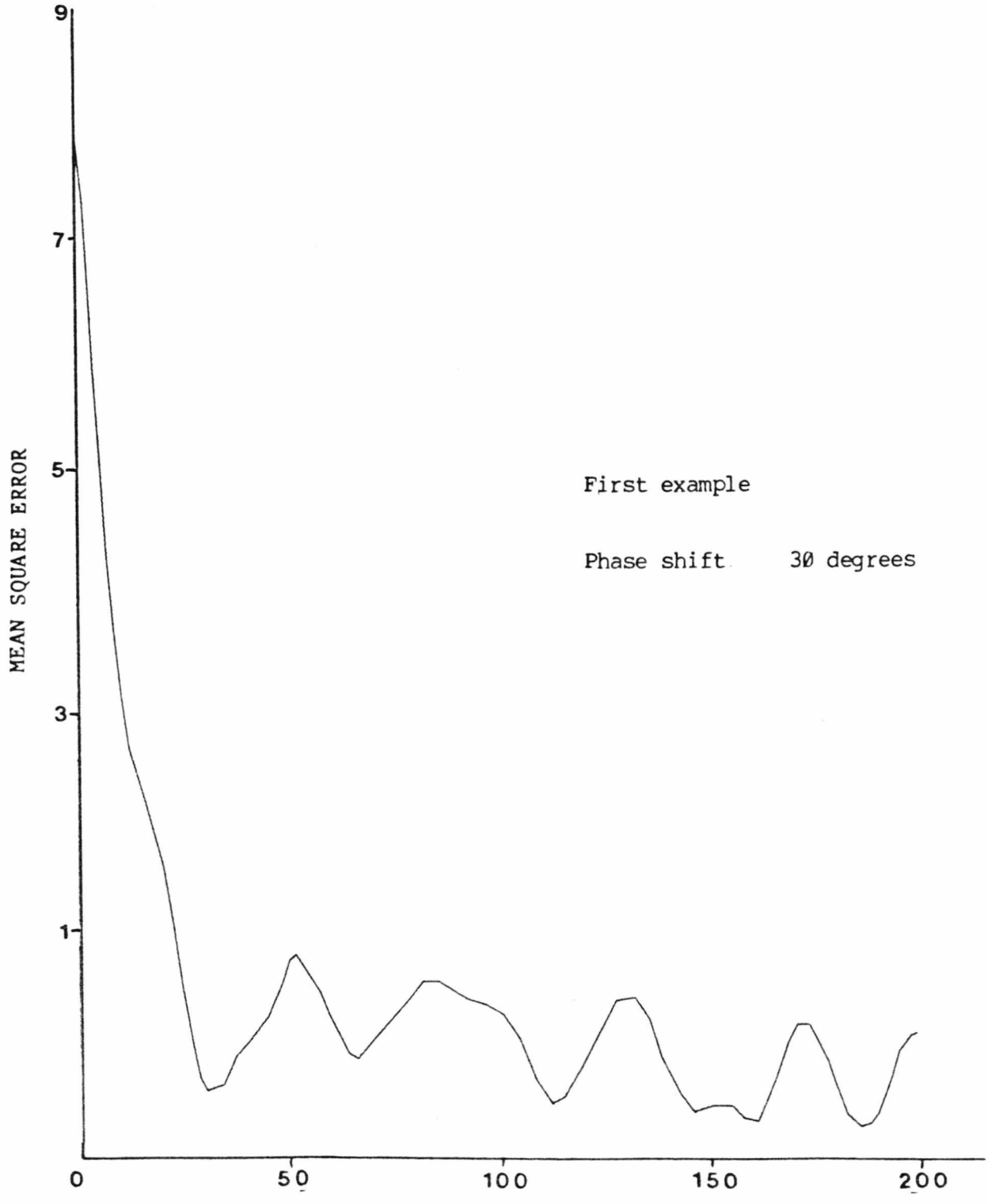


Figure 4.6 Mean Square Error vs No. of iterations for FADF with 30 degrees phase shift

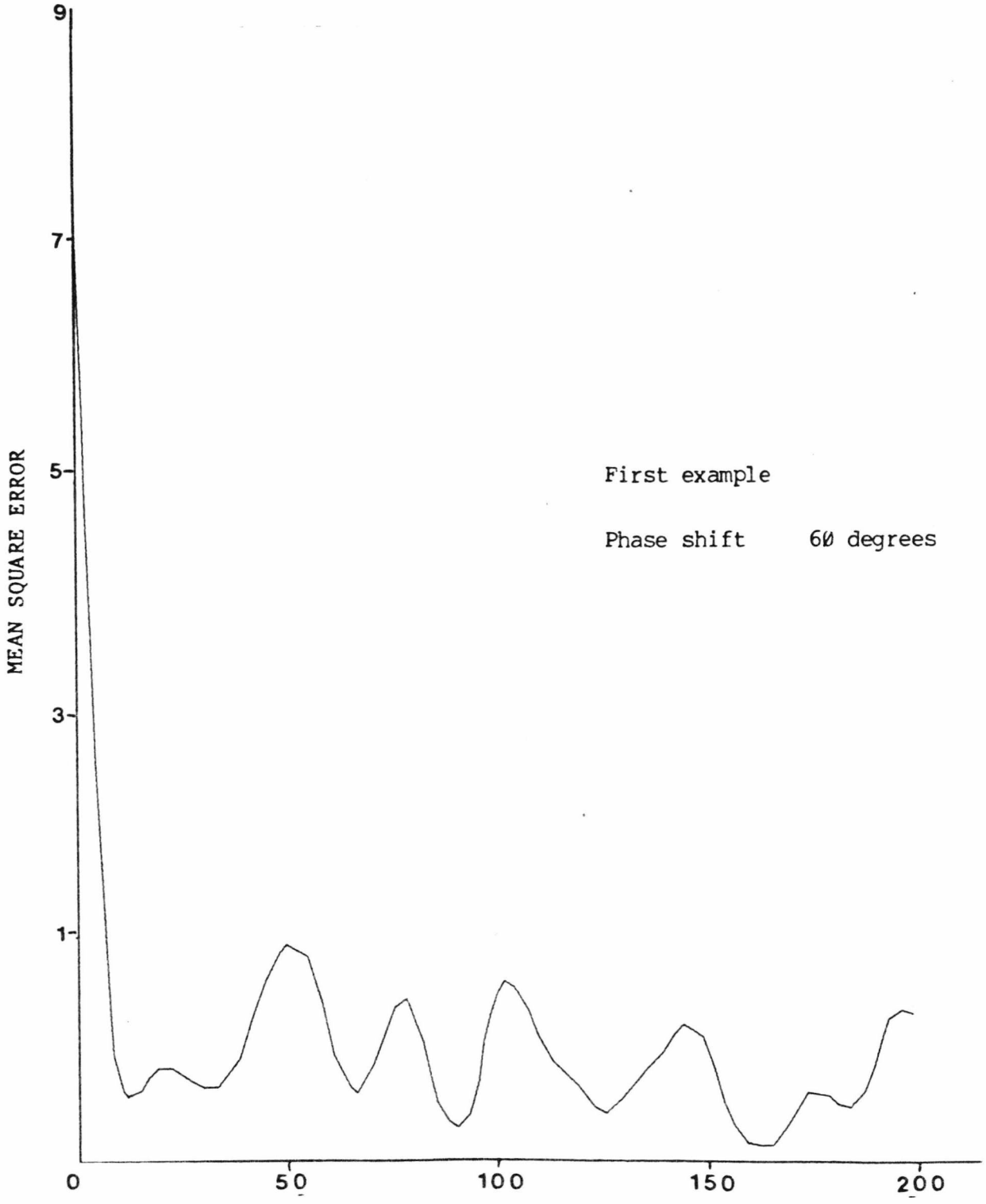


Figure 4.7 Mean Square Error vs No. of iterations for FADF with 60 degrees phase shift

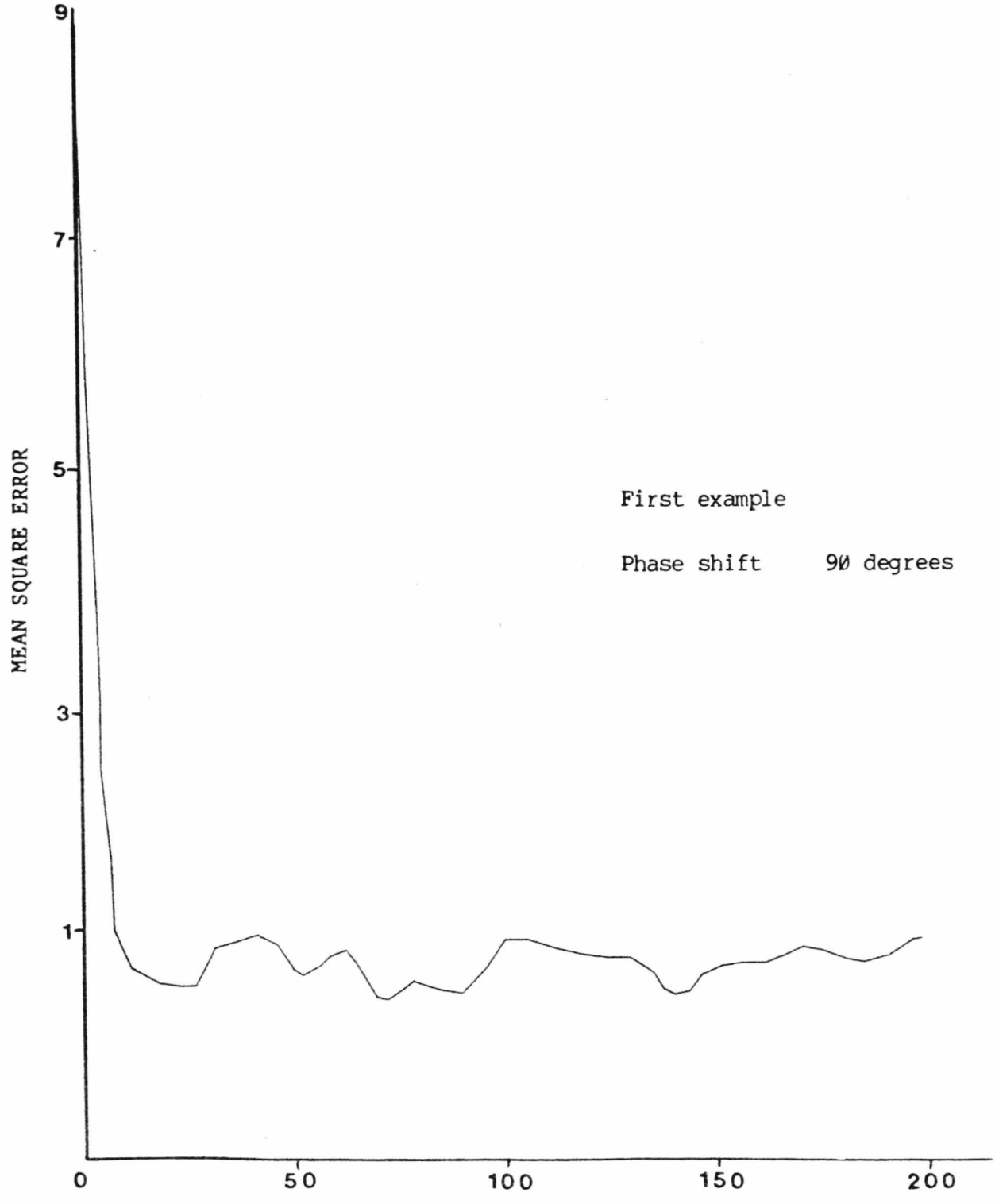


Figure 4.8 Mean Square Error vs No. of iterations for FADF with 90 degrees phase shift

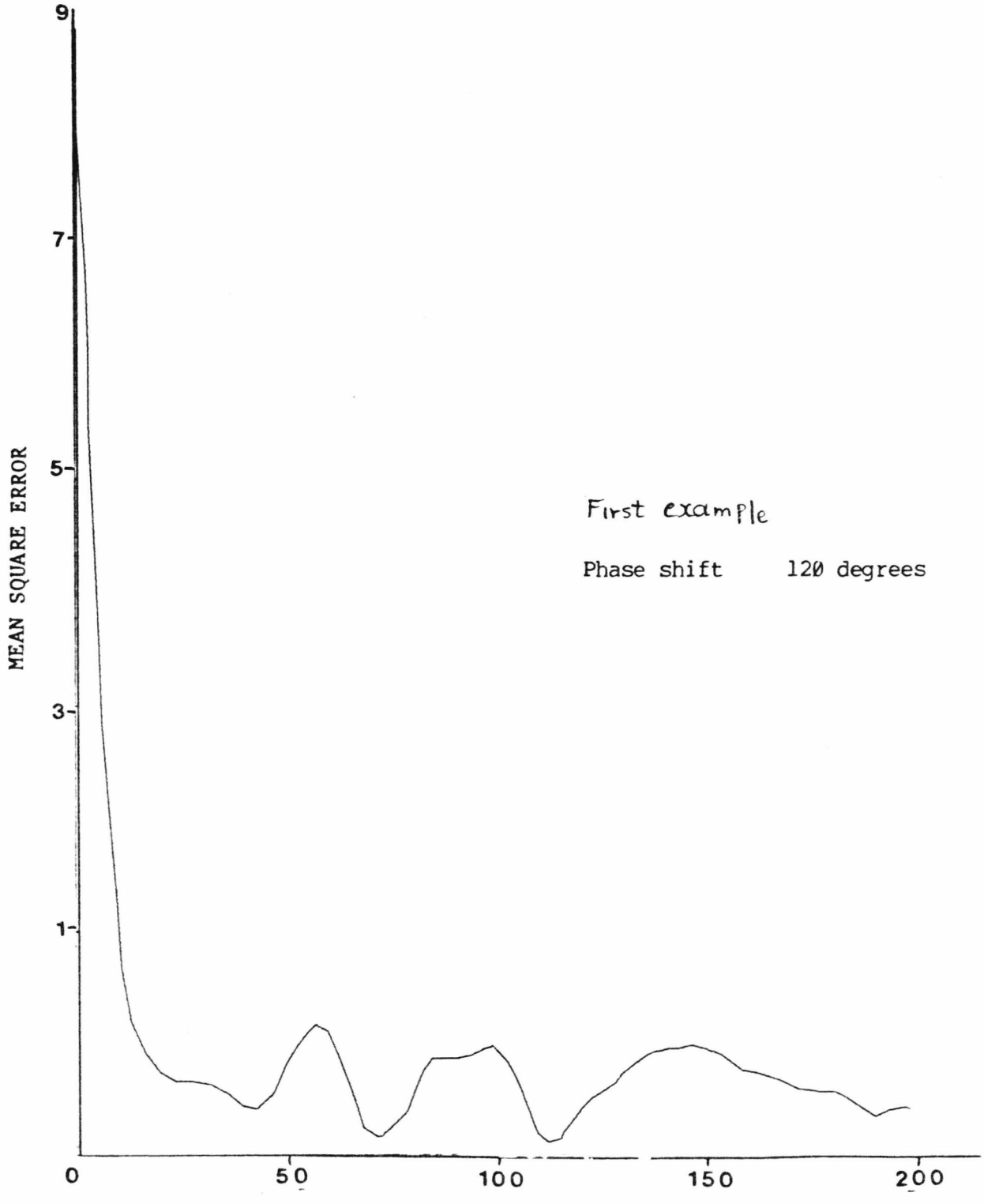


Figure 4.9 Mean Square Error vs No. of iterations for FADF with 120 degrees phase shift

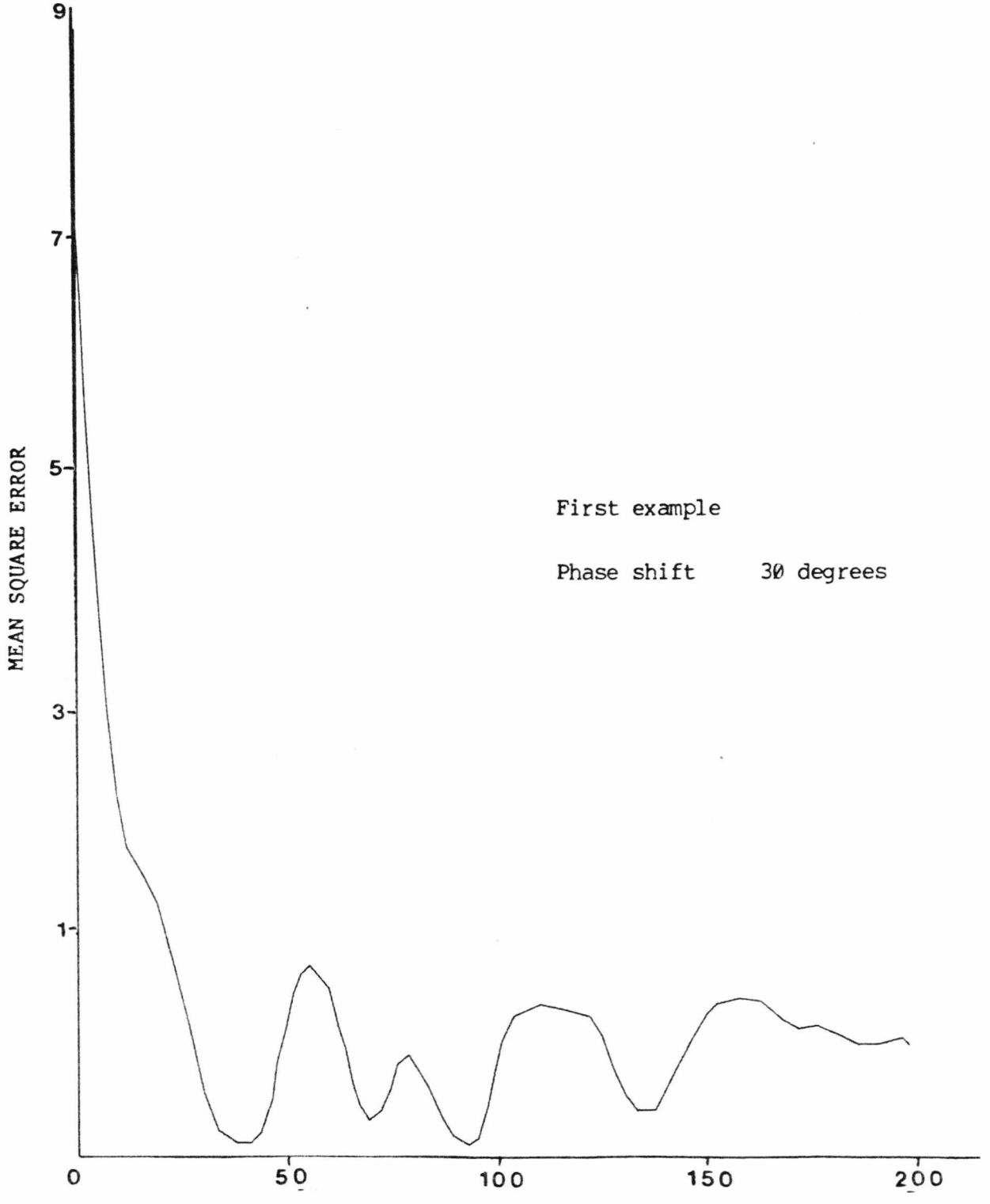


Figure 4.10 Mean Square Error vs No. of iterations for FADF with 30 degrees phase shift and differ in input and reference signal amplitude

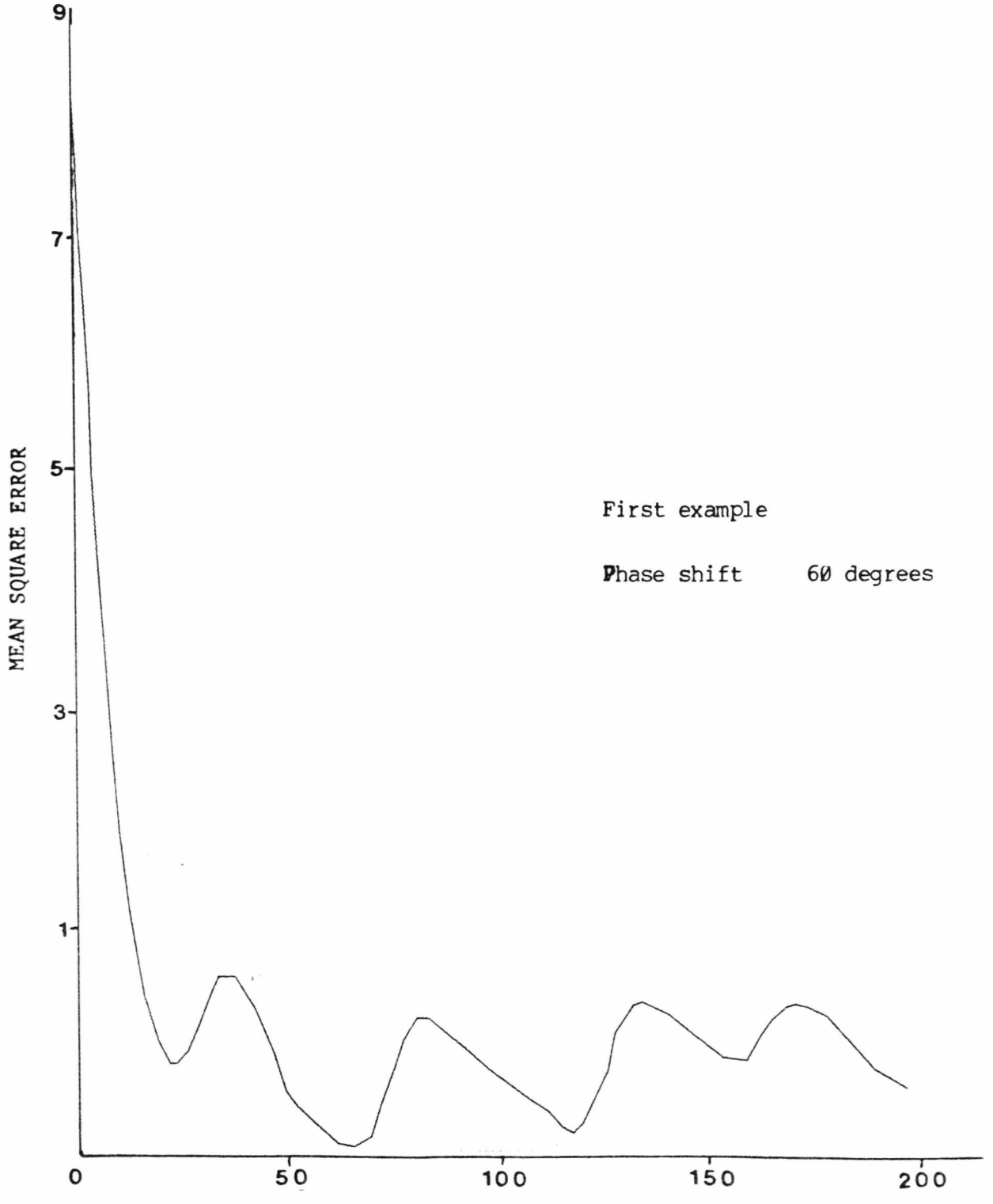


Figure 4.11 Mean Square Error vs No. of iterations for FADF with 60 degrees phase shift and differ in input and reference signal amplitude

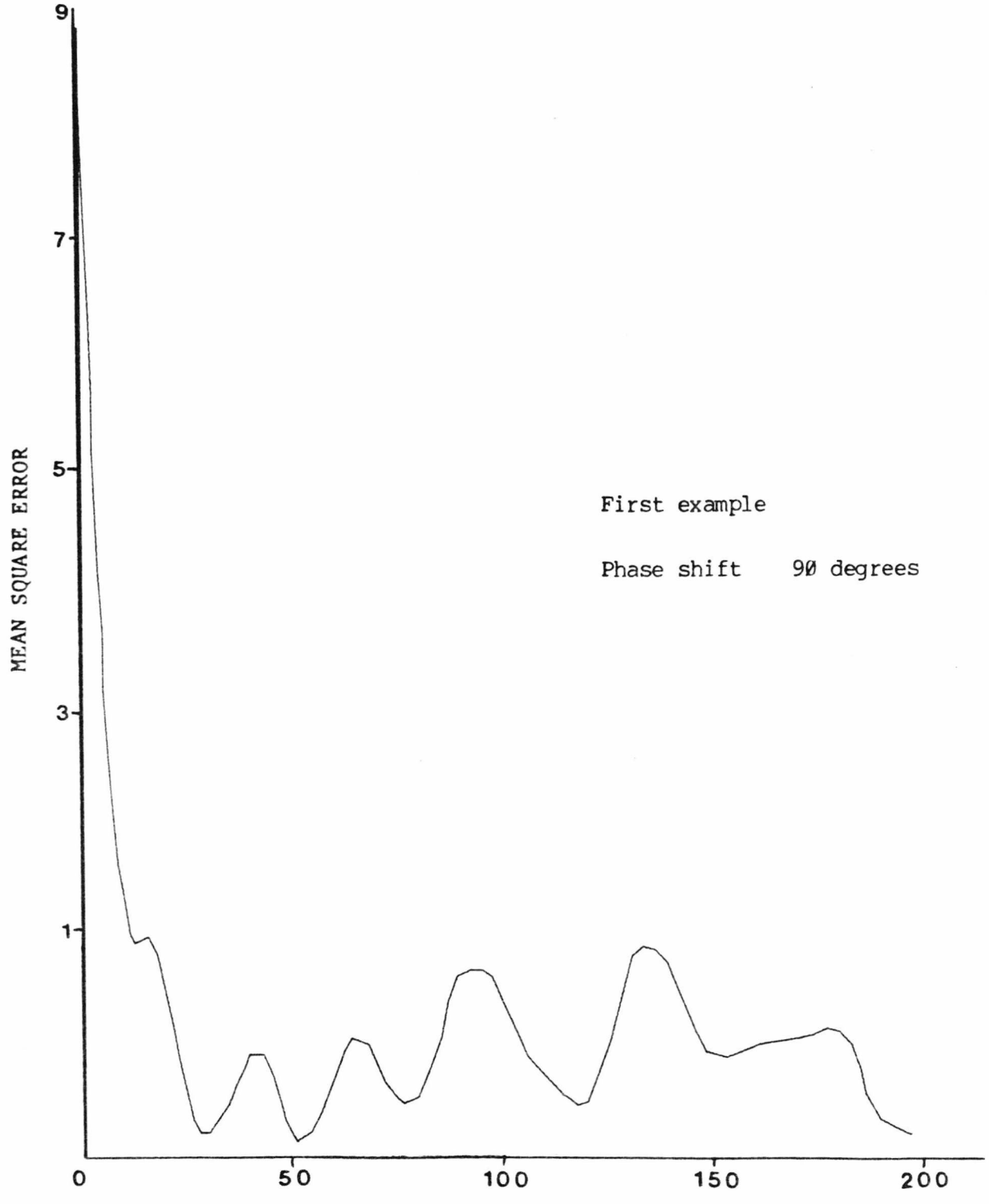


Figure 4.12 Mean Square Error vs No. of iterations for FADF with 90 degrees phase shift and differ in input and reference signal amplitude

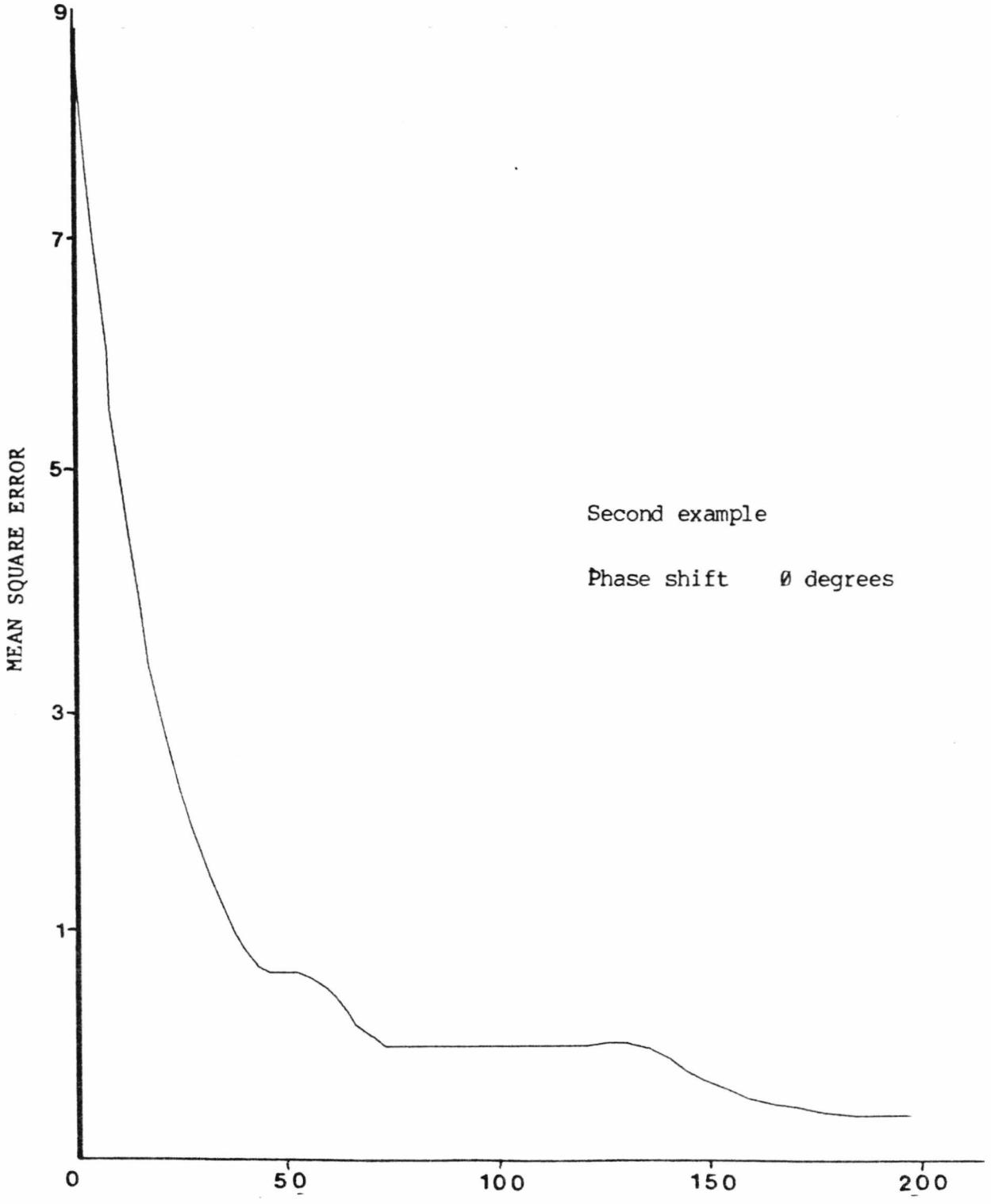


Figure 4.13 Mean Square Error vs No of iterations for FADF in the second example

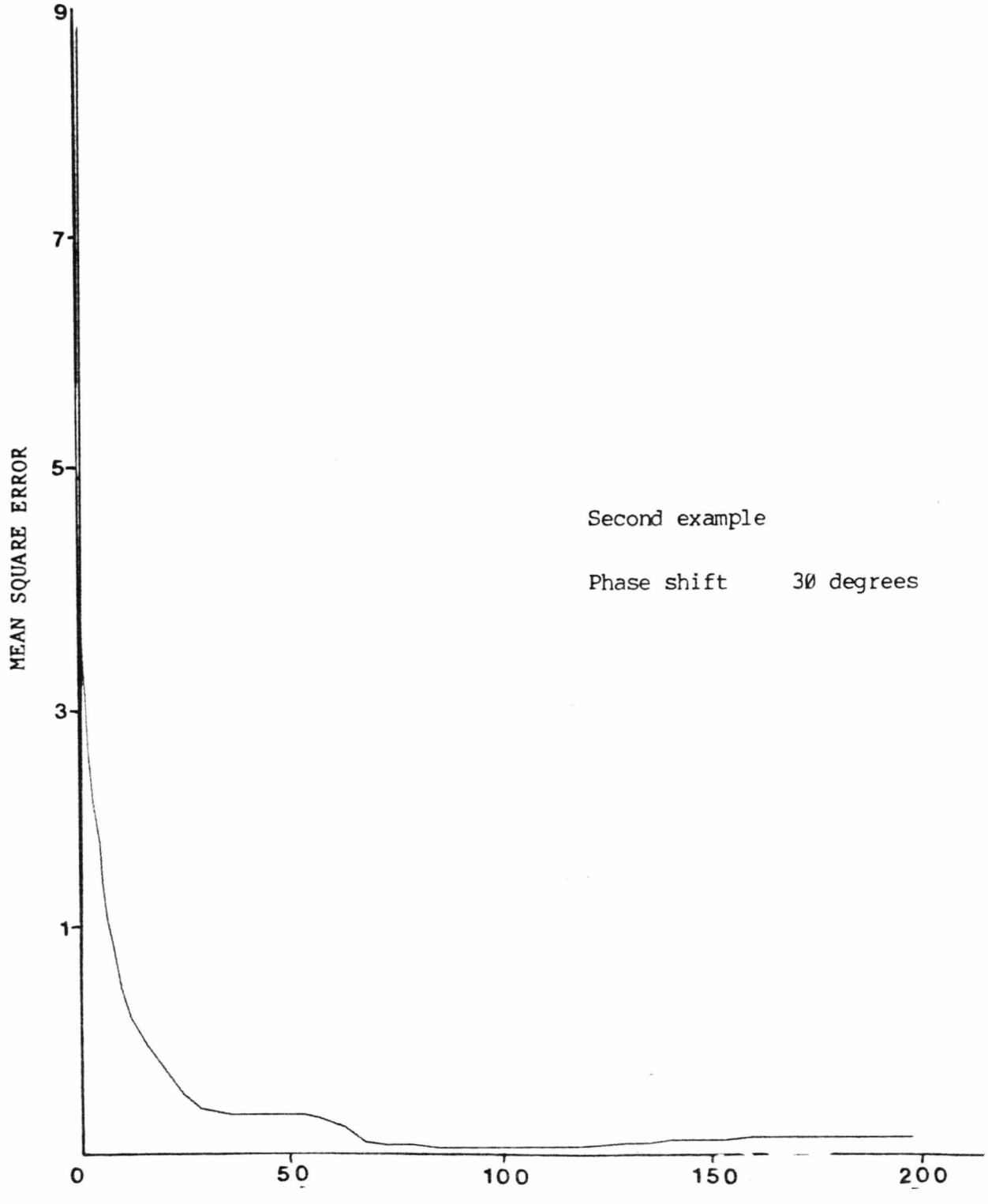


Figure 4.14 Mean Square Error vs No. of iterations for FADF in the second example with 30 degrees phase shift

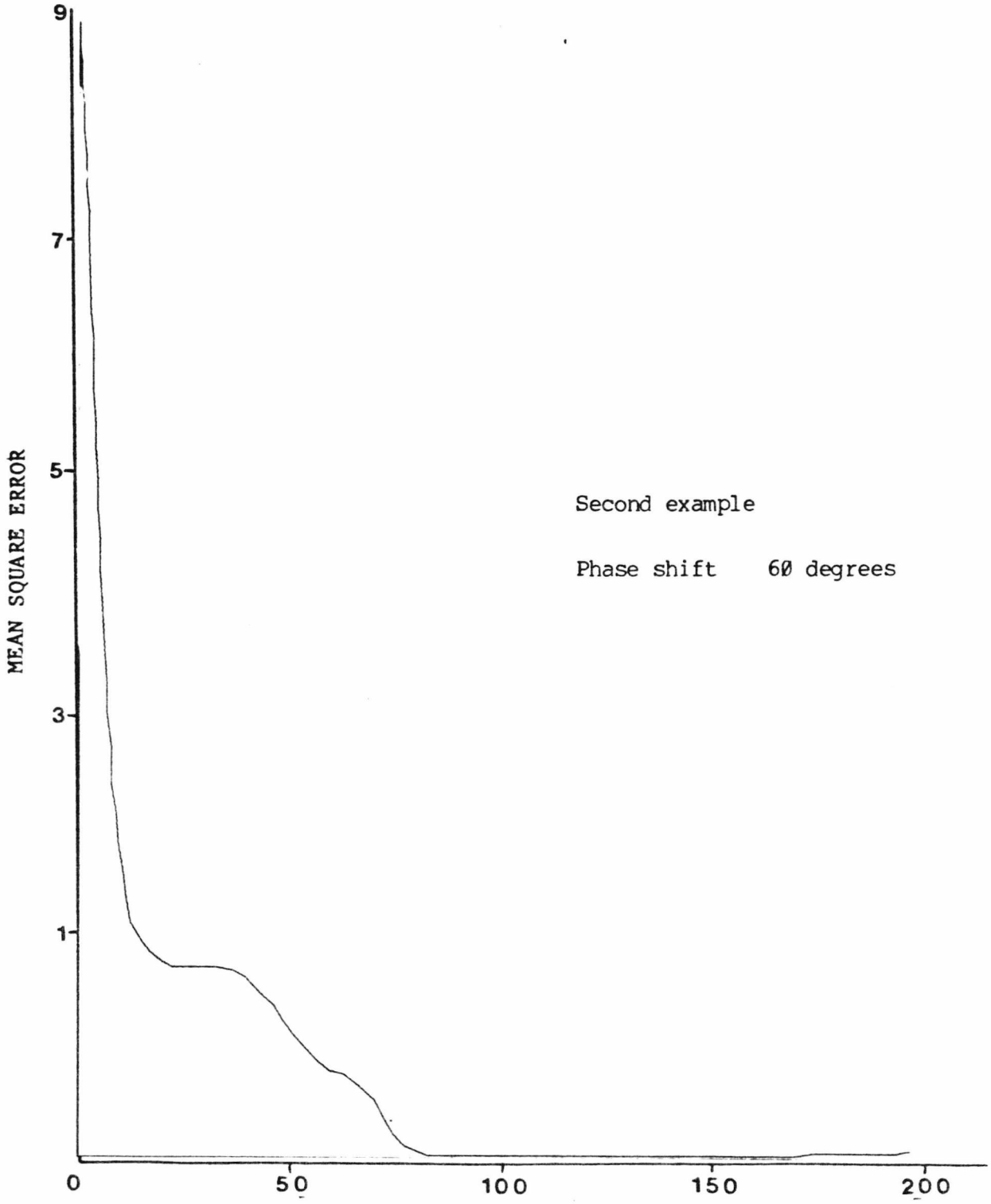


Figure 4.15 Mean Square Error vs No. of iterations for FADF in the second example with 60 degrees phase shift

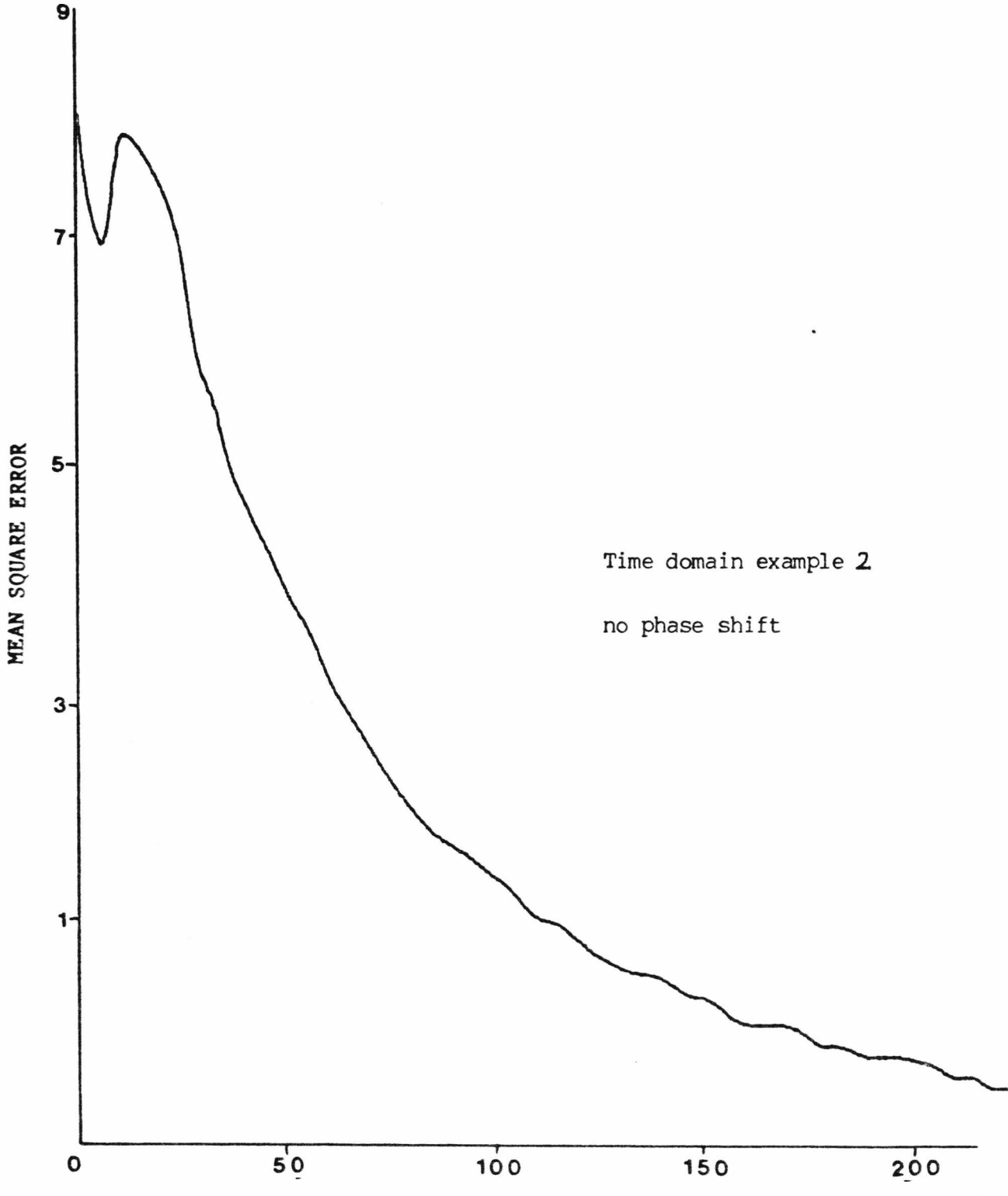


Figure 4.16 Mean Square Error vs No. of iterations for TADF with convergence constant = 0.0023

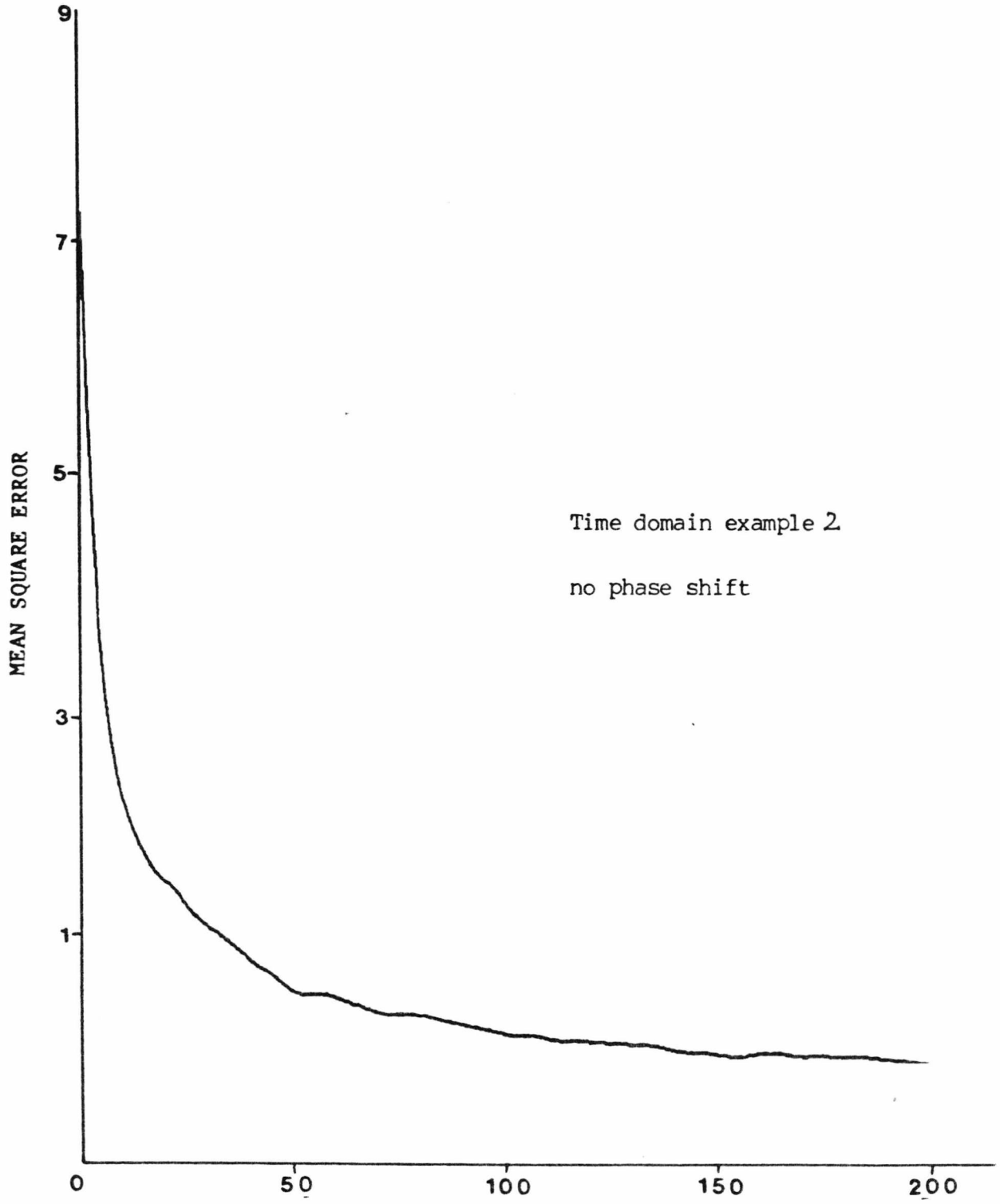


Figure 4.17 Mean Square Error vs No. of iterations for TADF with convergence constant = 0.023

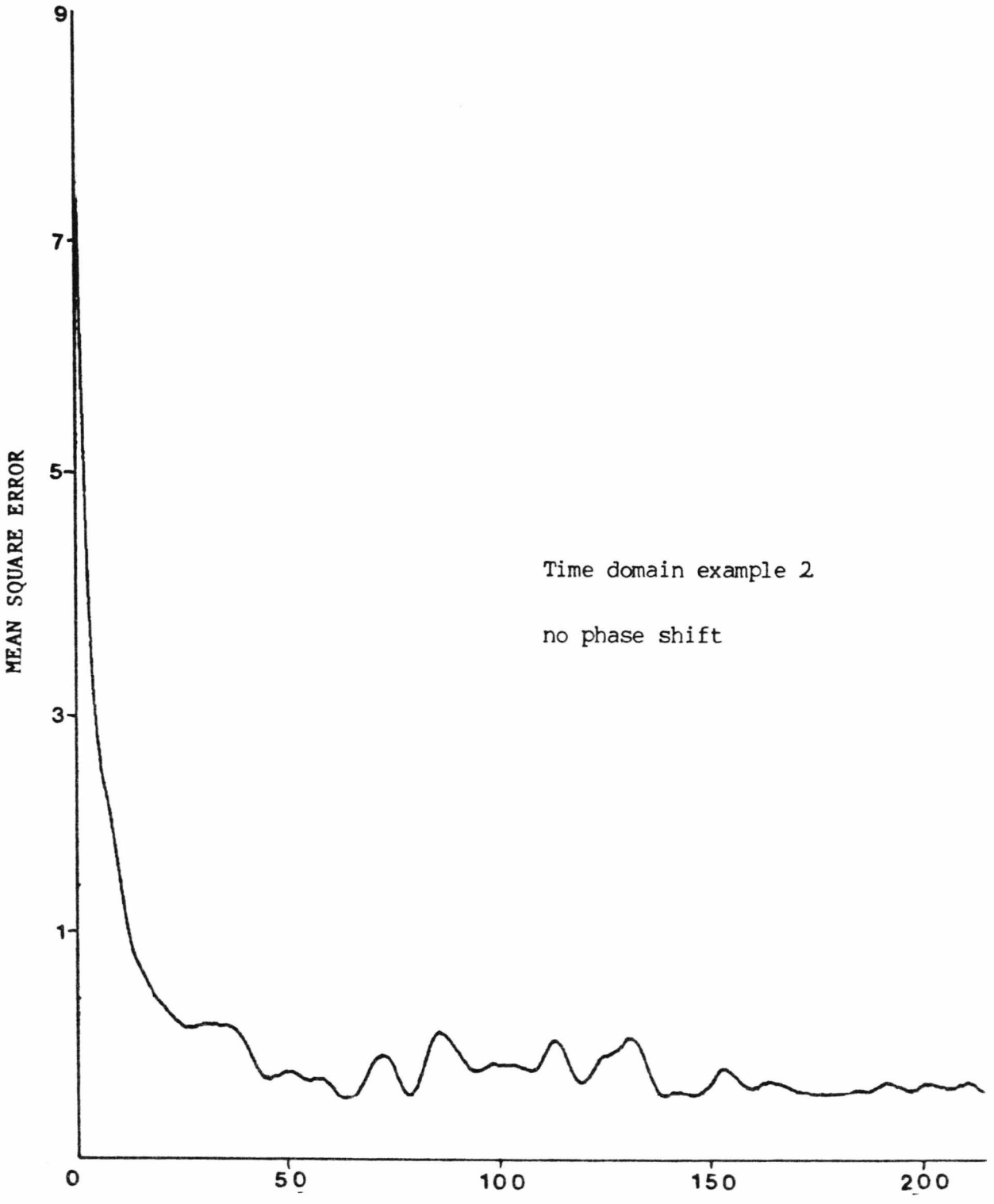


Figure 4.18 Mean Square Error vs No. of iterations for TADF with Convergence constant = 0.23

$$y_k = \sum_{n=0}^{N-1} x_n \cdot h_{k-n}$$

To produce one output point requires N real multiplications and N - 1 additions. Therefore, to produce N outputs requires N_2 multiplication and N (N - 1) additions .

Recall LMS algorithm ;

$$w_{k+1} = w_k + 2 \cdot \mu \cdot e_k \cdot x_k$$

To produce N outputs require N^2 adaptations . The second term requires N (N + 1) real multiplications . The addition operations requires N^2 real add . The cost of computing $e_k = d_k - y_k$ is N real add . The total arithmetic operation per output block for the LMS algorithm requires N (N + 1) real add and N (N + 1) real multiplications . Thus, the total computational complexity for LMS adaptive filter is N (2N + 1) real multiplication and $2N^2$ real additions .

4.5.2. COMPUTATIONAL COMPLEXITY OF FLMS ADAPTIVE FILTERS

It is well known that the digital filtering of an N_1-length sequence with an N_2-length sequence can be achieved by linear convolution . Since, in many applications such as speech, seismic processing etc., the input sequence (N_2) is large , perhaps infinitely large . Thus , the linear convolution would be hampered by two obstacles . First the memory would be large , and secondly the result would be subject to enormous delay . These difficulties have been relieved by a scheme known as sectioning [Gold,41,1969] . Sectioning can be performed in two ways overlap-add and overlap-save . Using either of this techniques, the linear convolution yields a sequence of length ;

$$N' = N_1 + N_2 - 1$$

This fact is used to find the minimum transform size to use, when performing the transform implementation of convolution with sectioning method . As an example let us define a convolution of two sequences x_n and h_n by its circulant matrix ;

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_3 & x_2 & x_1 \\ x_4 & x_3 & x_2 \end{bmatrix} \cdot \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} \qquad \text{(a)}$$

In order to fully use the properties of the transform, for the implementation of the linear convolution, the dimension of the above circulant matrix must be of N' *by* N' . Equation (a) can be augmented as follow to form a circular matrix ;

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} x_3 & x_2 & x_1 & x_4 \\ x_4 & x_3 & x_2 & x_1 \\ x_1 & x_4 & x_3 & x_2 \\ x_2 & x_1 & x_4 & x_3 \end{bmatrix} \cdot \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ 0 \end{bmatrix}$$

A careful comparison of the above development and the development described by Oppenhiem and Schafer [ ] shows that they are equal . Thus, by appending appropriate numbers of zero -valued, the linear convolution, by sectioning, can be achieved by the transform techniques . Because of the sectioning procedure used in the transform implementation of linear near convolution, it causes more than the necessary number of values to be computed . Since the weight vector has only N_1 weights, all but the first N_1 values must be discarded before using the FLMS algorithms .

In this respect, the computational complexity of the FLMS adaptive filters in the transform domain (fig 4.2), using one the transforms having an FFT type algorithm described in chapter 4, can be summerised as follow :

The transform implementation of convolution requires $3 \cdot \frac{N'}{2} \cdot \log_2^{N'}$ multiplications and $3 \cdot N' \cdot \log_2^{N'}$ additions, provided that N' is a power of two .

Recall FLMS algorithm ;

$$W_{k+1} = W_k + 2 \cdot \mu_f \cdot |E_k|_M \cdot |X_k^T|_M$$

To produce an N' out put it requires $3N'$ real multiplications and $2N'$ real additions . Thus the total arithmetic operations per N' output for the FLMS adaptive filters require $3 \cdot \frac{N'}{2} \cdot \log_2^{N'} + 3N'$ real multiplications and $3 \cdot N' \cdot \log_2^{N'} + 2N'$ real additions . If however, the kernel of the transform is a power of two, the number of multiplications will reduce to only $3N'$.

4.5.3. COMPLEXITY RATIO

In previous sections we presented the number of arithmetic operations for the LMS and FLMS adaptive filters . A comparison is made between LMS and FLMS filters described above . A computer program was used to compare the above complexities. A complexity ratio (CR) which is the ratio of the number of additions and multiplications of the time and frequency adaptive filters is used for the comparison. A complexity ratio, "CR", is plotted, and tabulated, versus the filter length . For these implementations see figures 4.19 and 4.20 and table 4.1 .

The complexity ratio (CR) is defined as ;

$$CR = \frac{complexity\ of\ LMS\ filters}{complexity\ of\ FLMS\ filters}$$

We define the addition and multiplication complexity ratio as follows;

$$CR = \frac{2 \cdot N^2}{3 \cdot N' \cdot \log_2^{N'} + 2N'} \qquad \text{addition}$$

and

$$CR = \frac{N(2N + 1)}{3 \cdot \frac{N'}{2} \cdot \log_2^{N'} + 3N'} \qquad \text{multiplication}$$

This is only the case where filter length = N is analyzed . As discussed in previous sections, the convolution implementation requires a sequence length of $N' = N_1 + N_2 - 1$. Because N' must be the power of two, for simplicity we assume that $N' = 2N$. The complexity is analyzed for N = 2 to 1024 . A line is drawn at CR = 1 in fig 4.19 and 4.20 for reference . It can be seen clearly from the graphs, that the frequency implementation of adaptive filters are attractive for most practical cases . For large filter lengths the complexity improvements are dramatic , reaching a factor of ten or greater .

4.6. CONCLUSION

In this chapter we presented the development and proof of FLMS algorithm for both DFT and NTT. It was shown that the weight update is proportional to the

correlation of the transform components of both the input signal and error achieved at the filter output. The proof also shows that the frequency adaptive filters can serve as a unifying framework for the Weiner problem.

The development of the FLMS algorithm was supported by a simple simulation of two applications. The results show that the FLMS does converge to an optimum solution, but the speed of convergence is dependent upon the convergence constant. However, in order to observe the convergence of the FLMS algorithm a complete simulation was presented.

We also showed that by analysis of a complexity ratio that frequency adaptive filters require fewer operations than time domain adaptive filters.

N	N'	CR radix-2 implementation of FLMS	
		real multiplication	real addition
2	4	0.42	0.25
4	8	0.60	0.36
8	16	0.94	0.57
16	32	1.6	0.94
32	64	2.7	1.6
64	128	4.8	2.8
128	256	8.6	4.9
256	512	16	8.8
512	1024	28	16
1024	2048	53	29

Table 1

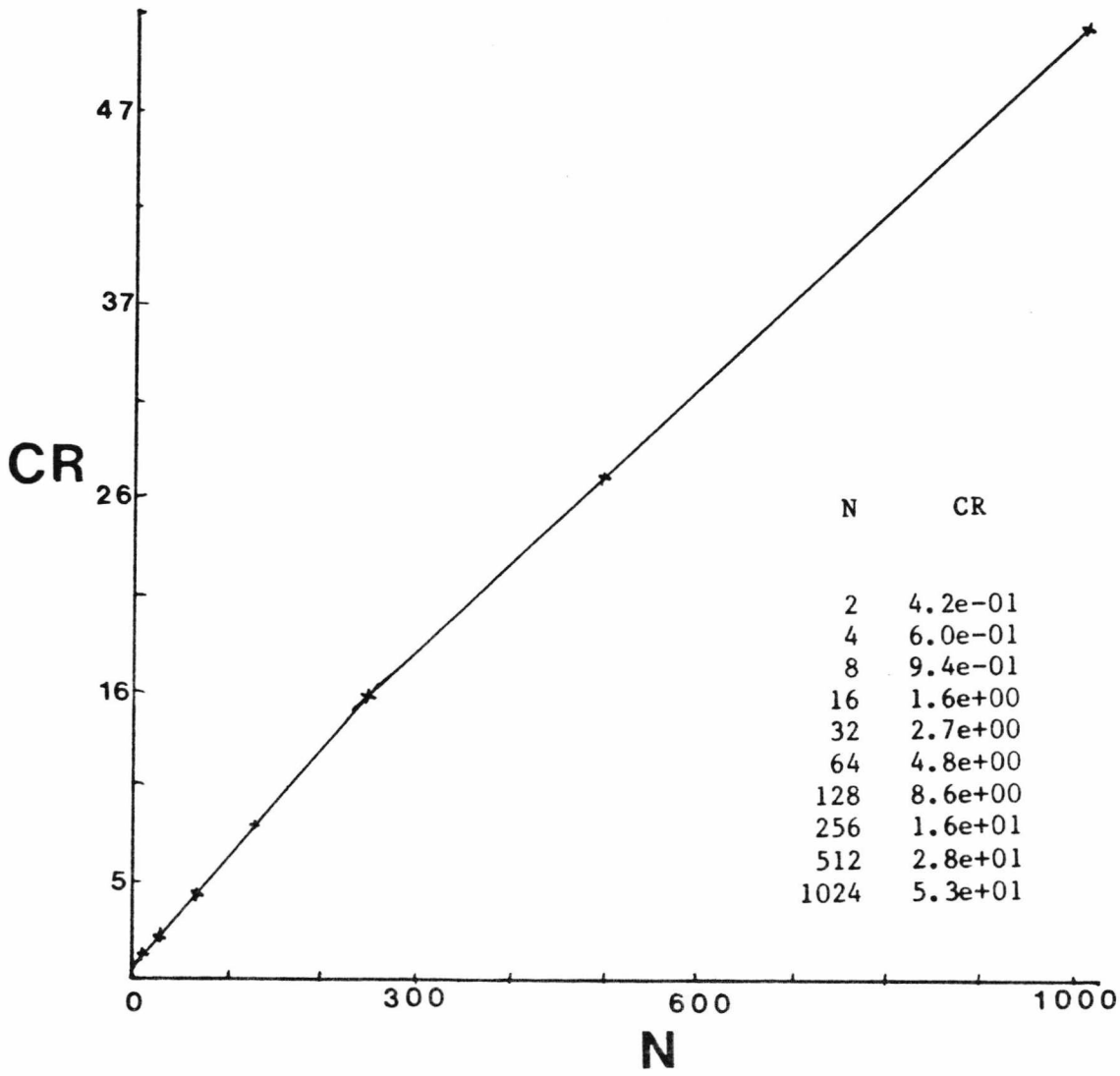


Figure 4.20

Complexity Ratio vs filter length for multiplication

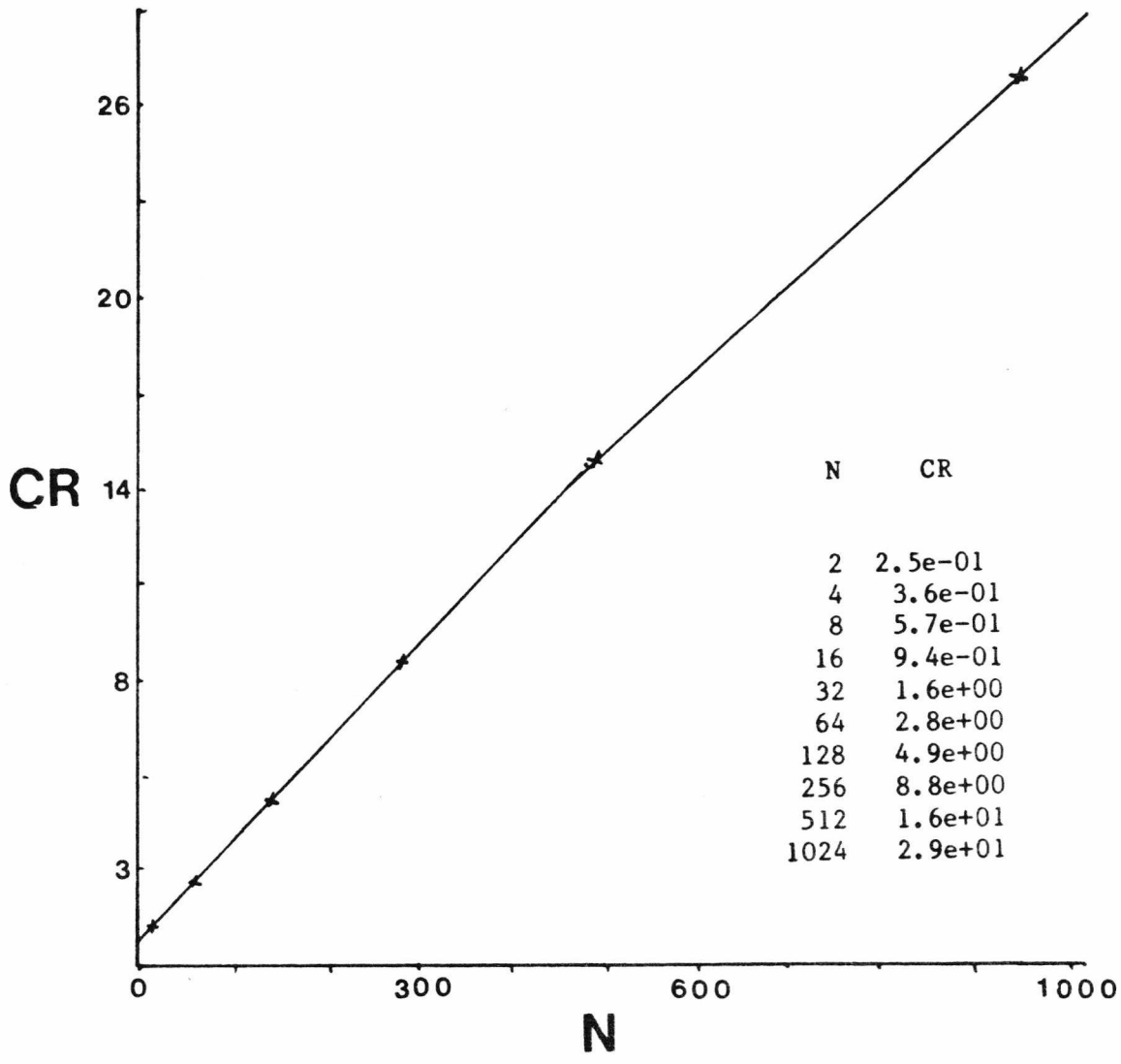


Figure 4.21

Complexity Ratio vs filter length for addition

CHAPTER 5

A SURVEY OF GENERAL AND SPECIAL PURPOSE SIGNAL PROCESSORS

5.1. INTRODUCTION

In order to specify the architecture of a processor, it is necessary to examine the application areas, and to abstract from them the character and structure of processing requirements.

The concept and techniques of digital signal processing discussed in the literature and previous chapters have been, and continue to be, applied to a wide range of applications. The exact nature of individual signal processing problems tend to be highly dependant on the particular area of interest. The aim here is to list and discuss very briefly a number of major application areas in which digital signal processing techniques have been successfully used.

SPEECH PROCESSING

One of the earliest fields of research to employ digital signal processing techniques was that of speech processing. Two major problems exist in this field. First is the analysis of human speech for such applications as speech recognition, encoding, and compression for efficient transmission. The second general problem area is that of speech synthesis. It is in this area that the greatest advances are currently being made. The applications are speech synthesizers for handicapped, voice response computer terminals, etc. For further information see [Oppenhiem, 43, 1978].

MUSIC PROCESSING

The application areas for music processing to which digital signal processing has been applied are, mixing multiple music signals into single performance, enhancement of music signal by the addition of special effects. Digital technique have also been used for the composition, synthesis, recording and transmission.

GEOPHYSICS

The major utilisation of digital techniques is concerned with the analysis of

seismic signals to aid the modelling of the structure and properties of the earth's interior, and with the study of earth quakes and volcanic activity.

RADAR

Radar systems are an example of how digital signal processing is used for high performance applications. The major signal processing functions of a modern radar system include signal generation, matched filtering, and estimation of target parameters. Another area of application is the adaptive digital beam forming radars.

SONAR

Sonar systems share many common signal processing concepts with radar. The application areas are associated with the detection and analysis of echos, navigation, mapping and spectral analysis.

IMAGE PROCESSING

The application of digital signal processing techniques to the processing of the images has been strongly influenced by the recent advances in integrated circuit technology. The major categories of image processing problems to which digital processing techniques have been applied include data compression, image restoration, enhancement as well as the creation of visual images from X-ray projections.

COMMUNICATIONS

Digital techniques have been applied to the problems of signal modulation, multiplexing, noise cancellation, echo cancellation, and tone detection. Many audio band communication signal processing functions have been implemented as a single integrated circuit.

BIOMEDICAL SIGNAL PROCESSING

The use of digital signal processing techniques is becoming wide spread in such medical applications as the analysis of EEG and ECG signals, and computer aided tomography (creation of 2 and 3-dimensional images) [Oppenhiem, 43].

Based on signal models (the signal modelling determines how the signal is interpreted to obtain information) and the specific goals of various applications, the required structure of the processing operations must be formulated. The specification of processing requirements is carried out in terms of mathematical formulas which have been described in earlier chapters and in the literature. In general, the signal manipulation tends to be based on a relatively small set of basic operations, such as convolution, correlation, discrete transforms and vector or matrix operations. The appropriate combination of these operations specifies the processing requirements.

The actual implementation of the various signal processing techniques and functions in a specific application area, may be implemented in three ways. For research purposes they are often simulated on the mainframe computers, the limitation of such computers generally restricts their use to low bandwidth applications. Alternatively, they may be implemented on the general purpose signal processor, which is a general purpose computer specially designed to carry out the signal processing algorithms. The third possibilities is to use a special purpose signal processor designed to execute one particular signal processing task. The aim of this chapter is to establish a dialogue between the implementation of special or general purpose processors and to outline the merits and limitations of both cases.

5.2. SPECIAL PURPOSE PROCESSOR

Special purpose processors fall into two categories:

1 Algorithm directed signal processors: are hardware implementations of particular signal processing algorithms. One example of this would be digital convolution.

2 Application directed signal processors: are hardware implementations of one or more signal processing algorithms, designed to be used in a specific application.

5.2.1. ALGORITHM DIRECTED SIGNAL PROCESSOR

Filtering, convolution, correlation, and discrete transforms are the usual algorithms implemented by algorithm directed signal processors. The design of these processors must take into account the following factors:

1 SPEED :- The maximum bandwidth the processor can handle is determined by its maximum sample rate, so a decision has to be made about the maximum clock rate that will be required.

2 INTERFACING :- The way in which the processor is communicating with other equipment e.g A/D or D/A etc. .

3 COST/COMPLEXITY :- The faster and more complex a processor is, the more expensive it will be to build.

The choice of which structure to use must then be made. Special purpose processors have three basic type of cells:-

1 MEMORY - This can consist of shift registers or RAM (Random Access Memory) for implementing the delays, and use for storing the coefficients.

2 ARITHMETIC UNIT :- It performs the necessary arithmetic operations, and often require multiply-add operations.

3 CONTROL - This involves control of the overall operation of the processor.

These basic blocks can be put together to realise processing algorithms.

5.2.2. SPECIAL PURPOSE PROCESSOR

One way of implementing the convolution of two sequences is as follows. Consider the convolution structure shown in figure.5.1 . The direct realisation of the convolution of figure 5.1 is to have;

1 An "N" shift register

2 A separate multiplier for each coefficient

3 An adder tree consisting of "N-1" adder

This is shown in figure5.2. This several advantages:

1 It is fast because it uses a maximum amount of parallelism.

2 Also it is simple to control, because all it needs is a clock signal.

This approach can be applied to FFTs and NTTs as well. The main disadvantage of the structure shown in figure 5.2 is the cost, which can be quite high. Nevertheless it is often used in applications where speed and simplicity of control matters most [Swartzlander,44]. Analogue implementation, such as integrated optic devices, use this technique because it is simple to implement [45-47]. This structure is also used for correlators, which use a one-bit representation of the signal, and have to run at high speed. Examples of these are radio astronomy receivers [48], and matched filters for hard decision spread spectrum [49].

However, most of the early special purpose implementations did not use this amount of parallelism due to the cost/performance. Instead they used one basic arithmetic unit which does all the operations required by the signal processing algorithm. This is shown in figure 5.3. This structure needs N clock pulses to produce an output and uses less arithmetic hardware than the direct realisation, and thus is cheaper to implement. Again, any convolution or filtering, or even discrete transforms, can be realised using this structure [White,50], [Groginsky,51]. The major disadvantages are that it is slow and needs more complicated control for sequencing the processor. However, this is not usually the case. For low frequency filters this technique with bit serial arithmetic is often used [Freeny,52]. Another system which uses the same structure is the microprogrammable arithmetic element(MAE) designed at the Plessey Research Center. A functional block diagram of the M.A.E is shown in figure 5.4 [Magar,53]. The multiplicand input register R1 can be loaded via two input ports X1 and X2 selected by the multiplexer, thus facilitating easy operation in

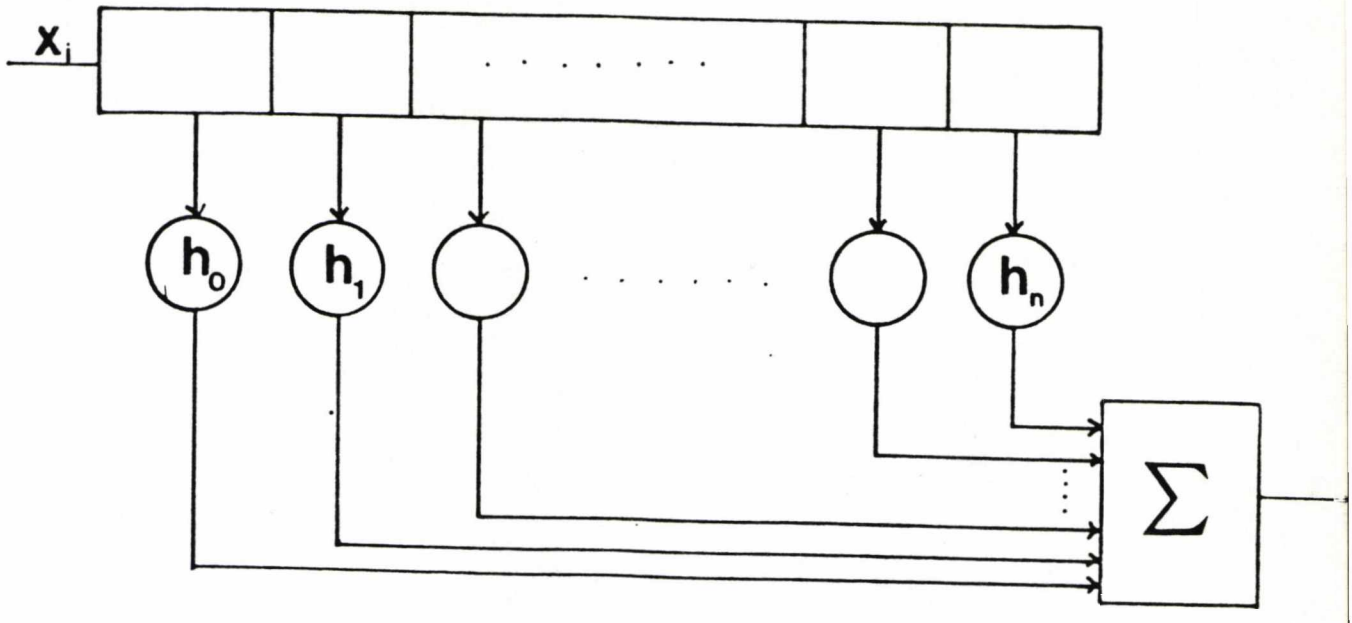


Figure 5.1 Convolution structure

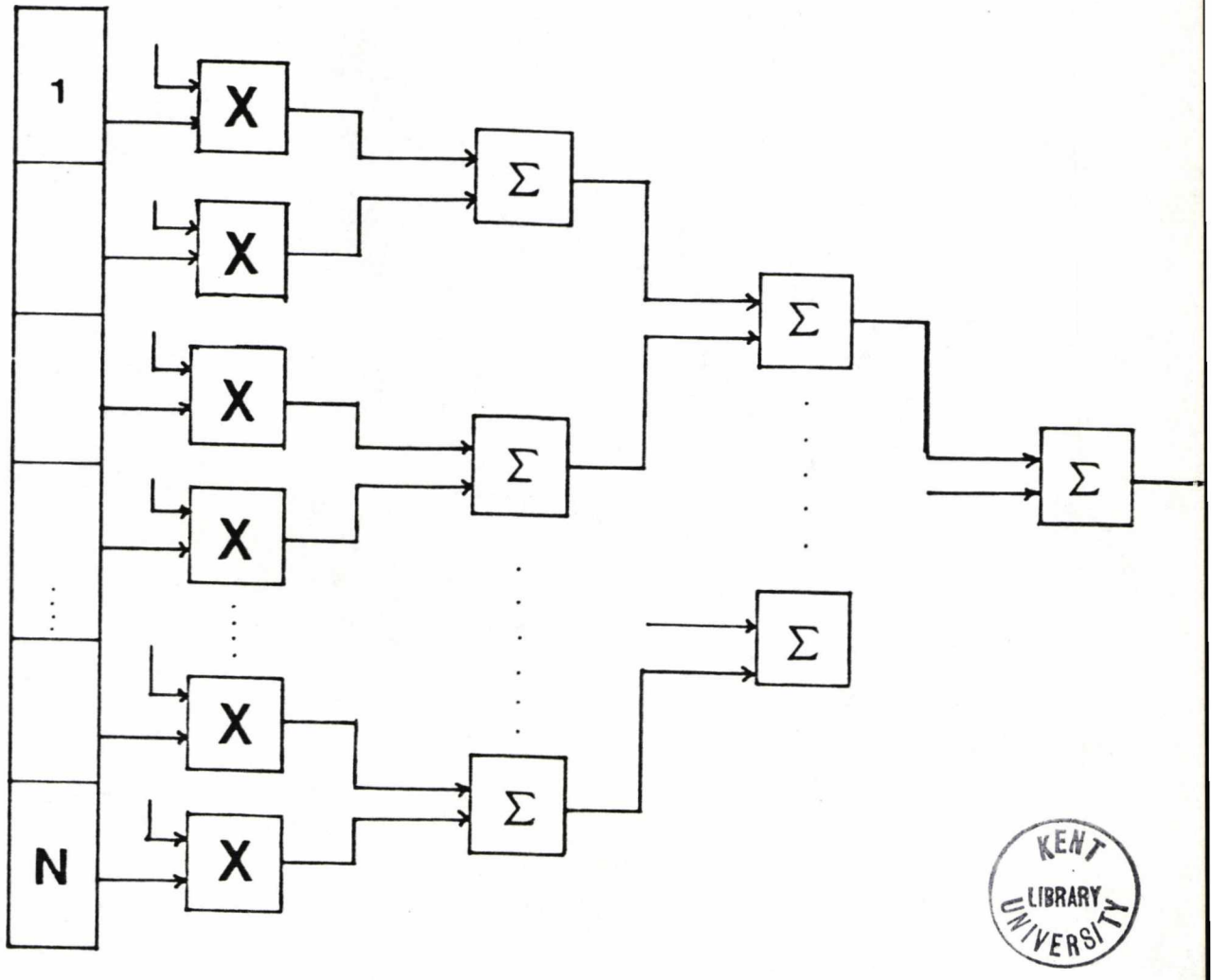


Figure 5.2 Direct realisation of the convolution

multibus systems. The adder can be externally accessed via register R2 and R3. The control of the chip is achieved via a two phase clocked instruction set. The data is represented with a 4-bit, however the dynamic range can be extended by directly cascading the number of chips. As an example a full butterfly for a radix-2 FFT has been realised by cascading the 4 M.A.E devices.

The complexity of the arithmetic unit depends on the algorithm being implemented for convolution. Also correlation a simple multiply/accumulation is a sufficient gesture for low frequency. However, for FFT and NTT the arithmetic unit is usually the basic butterfly block of the transform. In this case the arithmetic element for the butterfly requires a relatively large memory, to provide two complex data and two complex coefficients, and to store the result in case of FFT. Gold [54] discusses the various trade-off involved.

5.3. APPLICATION DIRECTED PROCESSOR

The algorithm directed structure is useful when the processing consists of similar operations. However, certain applications such as modem and speech processing often need several different algorithms. For example, a modem may need filtering, modulation, and channel equalisation. Although algorithm directed processors can be used, this is often expensive because of the need to provide different functions, and the necessary control to tie them together. Hence, these need some general purpose ability as well as dedicated hardware.

One solution is to use a general purpose LSI computer which is microprogrammable, and to add on to this the necessary special purpose processing element that the task may need. By doing this one has a more flexible microprogrammable signal processor. This technique is powerful because it has the advantage of adding extra facilities by adding or changing the microprogram. This technique has been used in many areas e.g. modem, speech processing [55-60].

A block digram of the Discrete Fourier Transform machine in [Chow,59] is shown

in figure 5.5. It uses two processors, a microprocessor and a bit-slice microprocessor. Because of the nature of the task the machine have to perform, the problem of data handling will become significant. In this case, the microprocessor is used to manage the data and send instruction to the bit slice processor. All the arithmetic computation is carried out on the bit-slice processor. It uses four AM 2903 to form a 16-bit wide processor. The machine is organised so that the microprocessor handles the I/O of the data as well as controlling the bit-slice processor. The machine was able to compute a 252 point Winograd transform with a sample rate of 6KHZ. However, the bit-slice can handle a sample rate of up to 18kHZ, but the microprocessor is not capable the data at this rate. The speed can be improved by loading and unloading the bit-slice processor using direct memory access. One major disadvantage of using this technique to perform the discrete transform of a sequence, is that by providing a fast processor to do the arithmetic it was found that the speed of computing a transform, using a microprocessor becomes limited by the number of memory access required another is the problem of calculating the address in order do the access.

A further approach is to have a special purpose hardware of an algorithm directed design and to control it using a general purpose microprocessor. In this case the special purpose hardware provides the speed, while the microprocessor will provide the flexibility. This approach has been used in image processing [Swartzlander,44], where a large amount of data has to be processed. A block diagram is shown in figure 5.6. It consists of two parts, a general microprocessor and a dedicated special purpose hardware. The general purpose microprocessor will handle the I/O interface as well as formating the data for the special purpose hardware.

The special purpose hardware is designed to be an efficient convolver. It requires the calculation of large numbers of the sum of the products in the form of the convolution. It is capable of 40 millions multiplications per second, which is needed in most image processing applications. Hence the dedicated hardware for this purpose is both cheap and efficient. The only limitation is the data bandwidth. Because the dedicated processor is connected to a general purpose processor, from which it receives

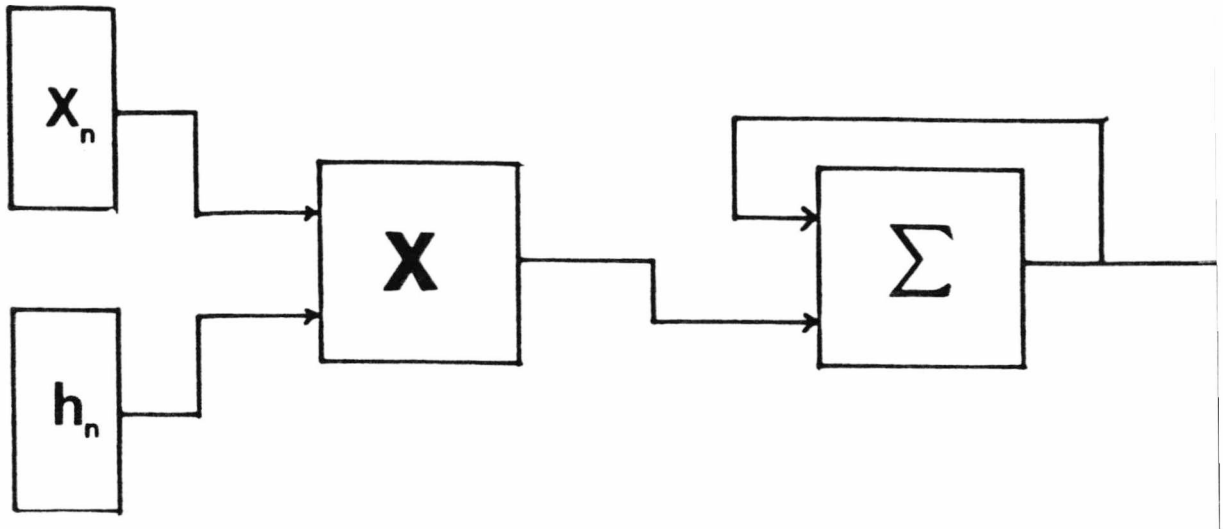


Figure 5.3 Sequantial realisation of the convolution

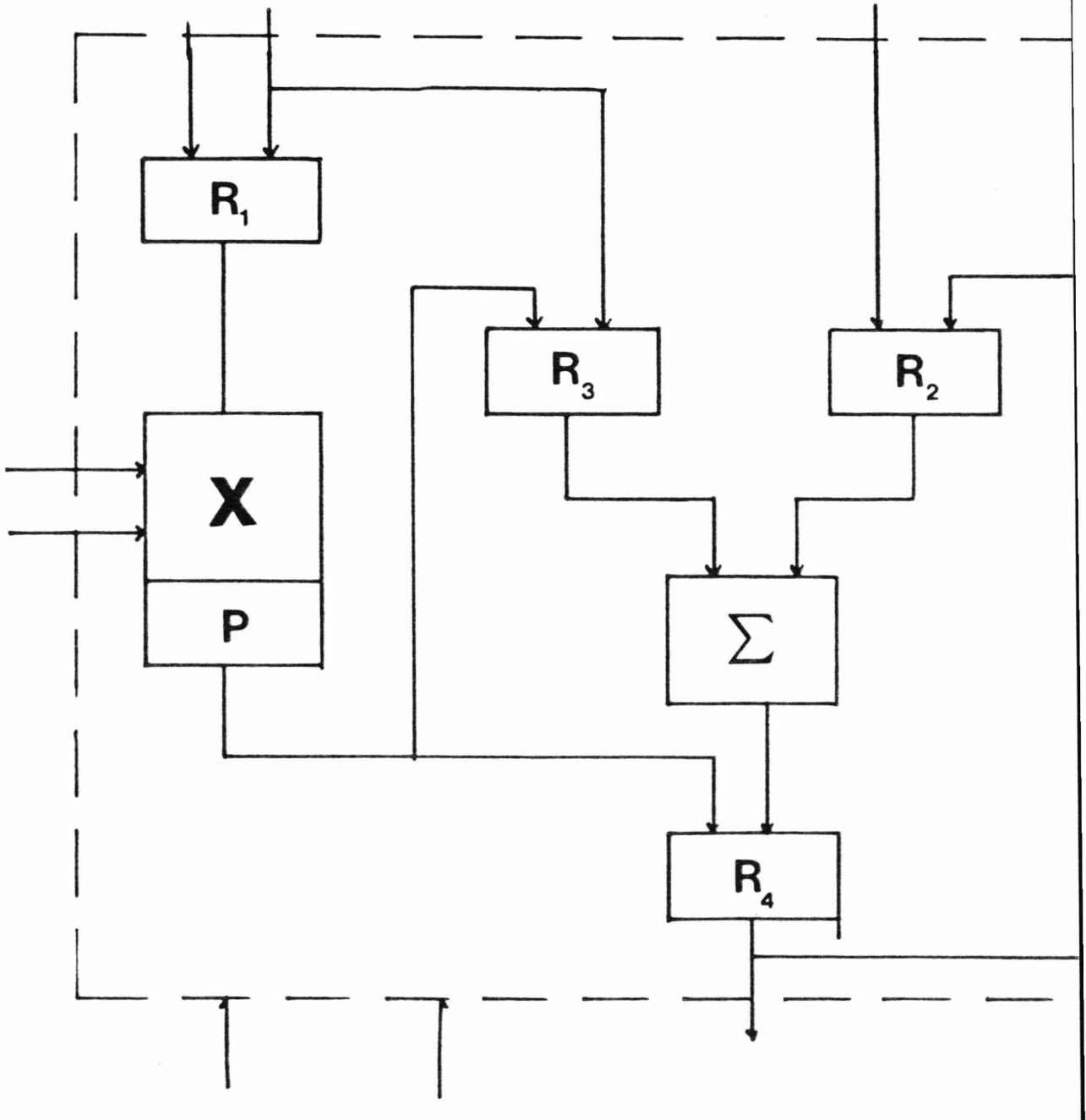


Figure 5.4 A functional block diagram of M.A.E [ref. 53]

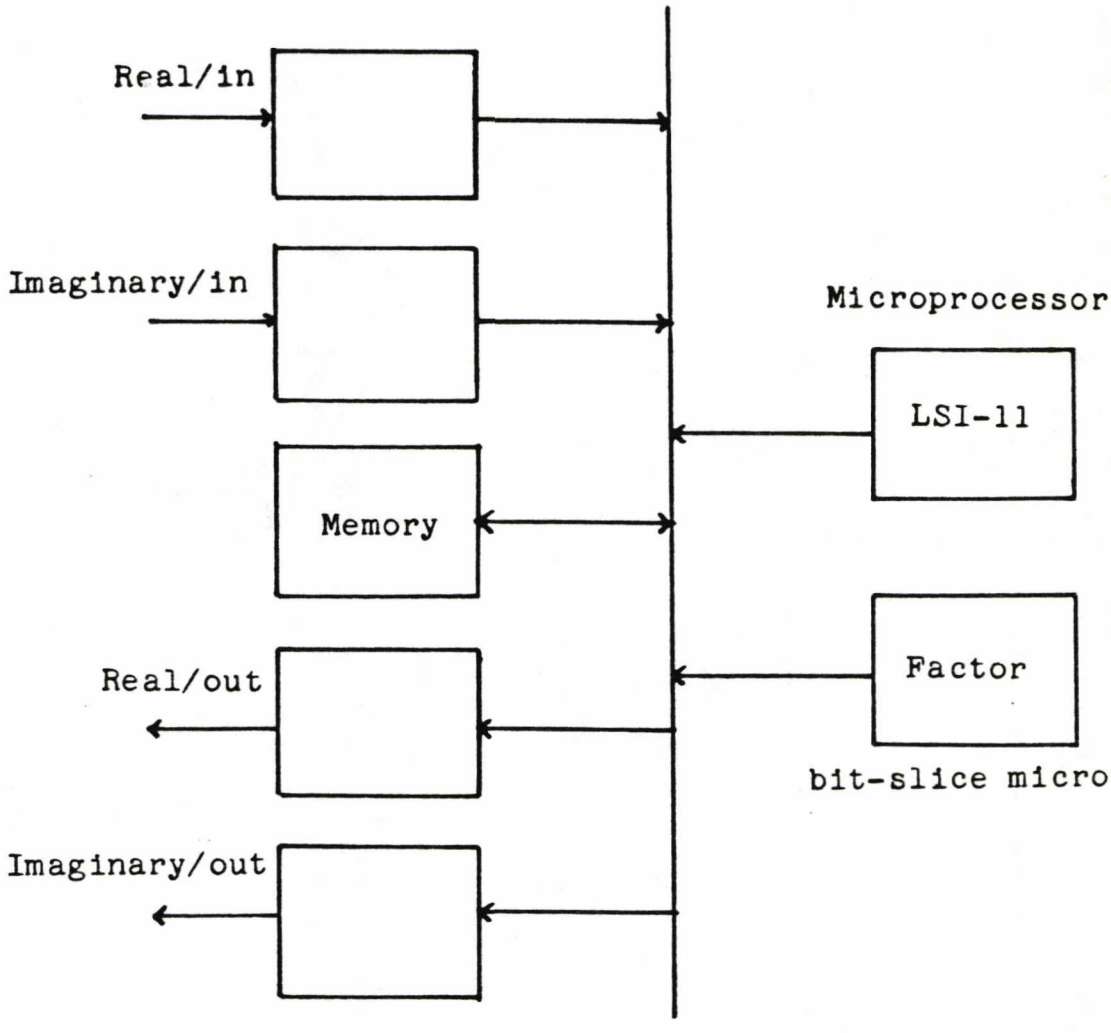


Figure 5.5 A block digram of DFT machine [ref. 59]

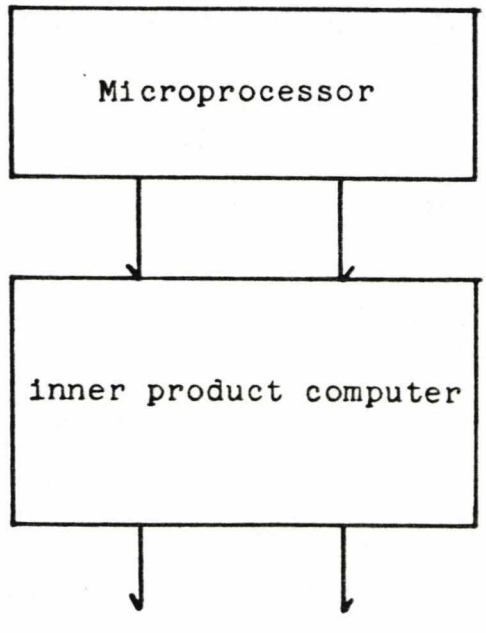


Figure 5.6 Inner product special purpose processor

data and to which it returns the results, Special efforts may be necessary to assure that the transmission bandwidth of the general purpose processor is adequately matched to the requirements of the dedicated processor.

5.3.1. SPECIAL-PURPOSE HARDWARE

One of the advantages of the special purpose processor is that the operations required are well defined. Therefore, it is easy to determine the design requirements which are not available in the general purpose processor. Here we shall examine some of the hardware technique used for the control and arithmetic operations.

CONTROL

Most special purpose processors execute a fixed sequence of steps e.g FIR filtering, convolution, and transforms. This means that the control can be very simple and based on a counter. The control signal may be obtained by decoding the output of the counter using a ROM (Read Only Memory). In the example shown earlier figure 5.1 shift registers are used for data storage. In this case all the control has to do is to control the movement of the data through the shift registers. The control for FIR filtering and convolution can be simple, relatively so in case of FFT's and NTT's.

ARITHMETIC

Special purpose processors have the advantage in that their arithmetic requirements e.g. word size, operation, etc are well-known before hand. This is especially attractive in cases where NTT's use modulo arithmetic. This is so, because , for example, the number of bits required to represent the data samples is known well in advance. The designers usually use the bit serial logic which offers a reasonable trade-off between cost and complexity [Lyon,]. However, for high speed parallel arithmetic, such as parallel multipliers, the multiple input adder [61-63] must be used. Hence, most effort in the design of the arithmetic unit has been carried out in order to reduce the cost of necessary multiplication

by coefficients.

Another advantage of the special purpose signal processor is that the coefficients are constant. An interesting technique, which was proposed by Peled and Liu [64-67], is known as distributed arithmetic. For example a digital filter perform the following operation;

$$y_k = \sum_{n=0}^{N-1} x_{k-n} \cdot h_n$$

where x_{k-n} is a set of input data samples and h_n is a set of pre-calculated filter coefficients. If both data samples are represented in two's complement then

$$y_k = \sum_{n=0}^{N-1} [\sum_{i=0}^{j} b_{k-n} \cdot 2^{-i}]$$

by reversing the order of the summations;

$$y_k = \sum_{i=0}^{J} 2^{-i} [\sum_{n=0}^{N-1} h_n \cdot b_{n-k}]$$

Hence, it is possible first sum all the coefficients multiplied by one bit of each data and then add and shift them. The advantage of this is that we can pre-calculate all possibile values of the result and store them in ROM (Read Only Memory). However, the amount of storage will increase exponentially with the filter coefficients. Hence, this feature of the distributed arithmetic technique them less attractive in cases of adaptive filtering because of the large amount of update needed for the pre-calculated coefficients.

Another aspect of the digital signal processing algorithms, such as filtering and discrete transforms, is the effect of finite-length register. In both cases sequence values and coefficients are stored in binary format with finite register lengths. This manifests itself in a variety of ways;

OVERFLOW

With data represented by finite word lengths, the processing results may need an additional bit for their representation. This is specially the case for the addition of two b-bit numbers. This situation is known as overflow. It's effect on the

two's complement numbers is shown in figure 5.7. In case of feedback loop digital filters, it can cause a full scale oscillations. A solution is to change the overflow characteristic to one of those shown in figure 5.8. Characteristic (a) is known as saturation operation and both (a) and (b) have been shown to provide stable operation [Freeny,52,1975].

In some cases, for example, intermediate stages in the discrete transforms, overflow is unacceptable. One approach is to divide the result of each butterfly stage by two. However, this aggravates another effect of finite word size, namely round off errors.

ROUND OFF ERRORS

Round off errors result from the arithmetic operations on two sets of data sequences. Since an M-bit data multiplied by an N-bit coefficient will produce M+N bit product, in all cases especially discrete transform the data word will grow unmanageably. For practical reasons one cannot continuously increase the word size, unless it is shortened. This can be achieved by rounding or truncation after each arithmetic operations. This inevitably produces an error at the output. This is combated by temporarily increasing the length of the data word for the duration of the computation. However, the hardware cost to provide these extra bits will manifest itself by an increase in the required storage, adder and multiplier hardware.

One way of avoiding these problems is to use floating point arithmetic [Freeny,52 Oppenhiem,68], but this is expensive to realise directly in hardware, or on a single-chip.

Overflow and rounding problems are often the result of intermediate calculation in digital signal processing algorithms. In this case the use of Finite Field arithmetic, for the calculation of signal processing algorithms, will overcome the problem of overflow and rounding. When Finite Field arithmetic is used overflow does not matter as long as the output is representable in the number of available bits. This is because the

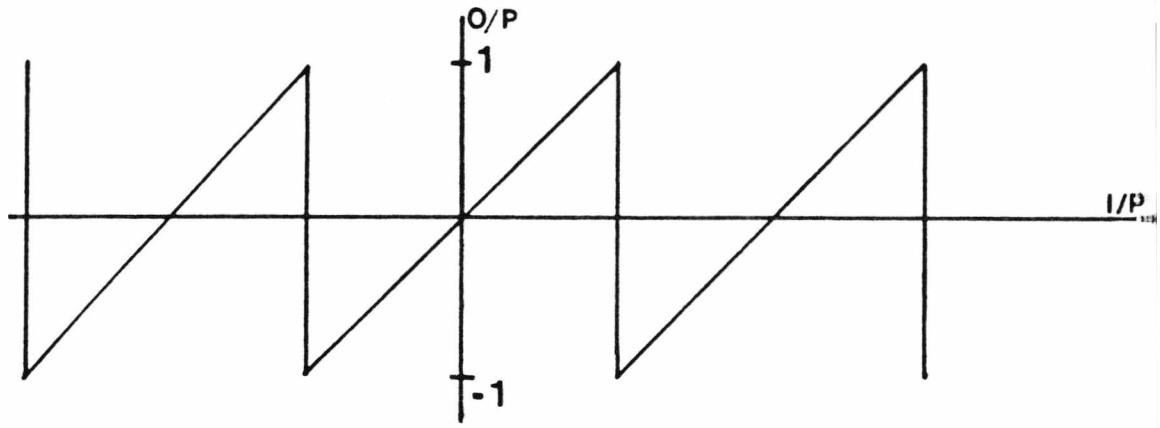
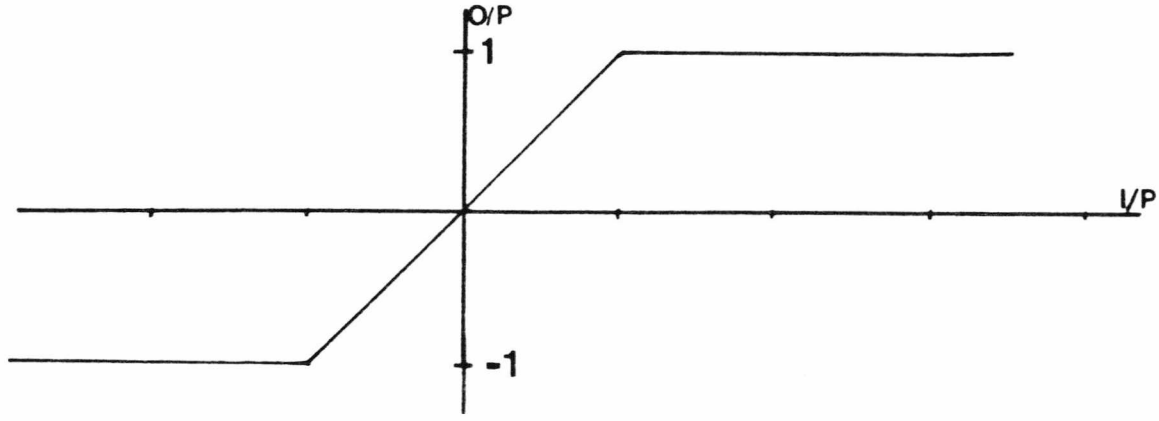


Figure 5.7 Overflow characteristics of 2's complement arithmatic



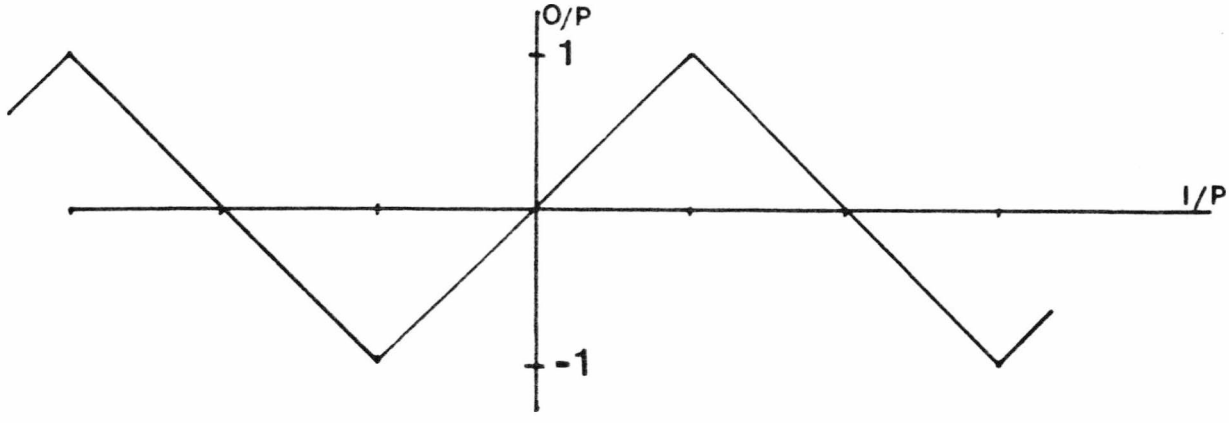


Figure 5.8 Modified overflow characteristics

output will be correct irrespective of the number of times the intermediate result have overflowed [Agarwal,22,23].

5.4. GENERAL PURPOSE SIGNAL PROCESSORS

A general purpose signal processor is a piece of hardware which executes a sequence of instructions that can be altered. In order to understand the requirements of a general purpose signal processor, it is best to study the architecture of a general purpose computer.

A block diagram of a general purpose computer is shown in figure 5.9. It consists of four basic units:

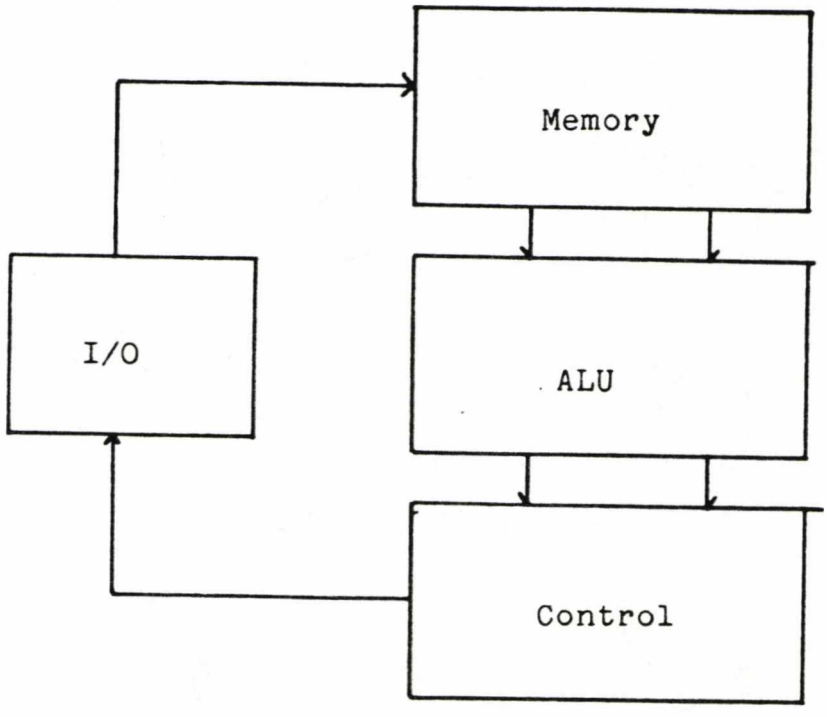


Figure 5.9 General purpose computer

THE CONTROL UNIT

The job of a control unit is to fetch an instruction from memory and, then to use the information contained in the instruction to control the other parts of the computer.

THE PROCESSOR UNIT

This manipulates data, which is obtained either from the memory or the input/output unit, to perform the operations required by the instructions received by the control unit.

THE MEMORY

This is a set of storage locations. A particular location can be selected by giving the memory unit its address.

THE INPUT/OUTPUT UNIT

The Input/Output unit is the point where data can be fed into the system and from which the result can be transmitted to the other systems or user.

The architecture of a computer is defined by the operations which it can perform and the precision and quantity of data it can handle. In this case the architecture of a computer will depend on the following features:

ARITHMETIC

The computer must have some form of calculating ability, and the provision of the necessary arithmetic operations, to perform a signal processing task.

DATA ACCESS

In a general purpose processor, the data accessing address must be programmed. Therefore, careful attention must be paid to make this efficient.

SPEED

The speed will be always determined by the technology. However, other techniques can be used, such as pipelining in order to increase the speed.

INPUT/OUTPUT

Signal processors must receive and transmit data. Sometimes this can be quite complicated, and hence, affect the performance of the processor. Therefore, it is important that the Input/Output structure be efficient, otherwise the processor will be restricted in its application range.

CONTROL

Unlike the special purpose processor, general purpose computers have to make decisions, and unless the control unit permits this to happen efficiently, the machine will be restricted. Another aspect of the control is the fetching and decoding of the instructions. If this procedure is not done efficiently it can cause a significant overhead on a general purpose machine.

A great many digital signal processors [70,82] are based on a general purpose computer structure slightly modified to improve its efficiency. The main disadvantage of the general purpose processor is the lack of parallelism. It is usually only one arithmetic unit which is used and there is only one path to the main memory. However, there are a number of techniques which can be used in order to adopt the architecture of the general purpose computers for digital signal processing tasks.

Firstly, the program and data memory are separate. This allows the processor to fetch instructions and data in parallel, unlike the more usual general purpose computers which must fetch them sequentially, because they are both in the same memory. The advantage of this technique is to remove the restrictions on the width of the instructions, which can be made as wide as necessary to control the processor.

Secondly, the arithmetic and memory access can be overlapped. This technique is known as PIPELINING. However, the designer aim is to keep the number of pipelining stages as low as possible, as the complexity of the control increases with the pipeline stages.

Let us now look at a specific example of a general purpose signal processor the using the above ideas.

5.4.1. THE LSP/2

A basic block diagram of the LSP/2 [Blankenship,72] is shown in figure 5.10. It is a programmable signal processor realisation, based on multiple functional units. The main architectural features are 64 dual-copy, 32-bit general registers, three 32-bit parallel data busses and a set of dedicated functional units. The dedicated functional units include index arithmetic unit, an Arithmetic Logic unit, a multiplier and division units, also 4kx32 bit data memory. The LPS/2 instruction set is divided into four classes: Arithmetic instruction, constant handling, control and memory addressing. Each class may encompass several function modules; e.g

1. Arithmetic A + B ----> D , A x B ----> D ,.....
2. Constants A + B ----> Y (Y is supplied in command control)
3. Control Y ----> P if A > 0 (P is program counter)
4. Memory A ----> M(B) , M(B) ----> A

Code is supplied by a separate program memory. It is this separation of data and program memory which allows the execution of instructions, done in parallel with the fetching and decoding of the next instruction while the present instruction is in progress.

Three streams of events work concurrently: see figure 5.11, accessing the program memory, the data register read/write and function module operations. The timing of the machine starts with the content of the program counter being altered, and new instructions being fetched from the program memory. When the program counter is altered, the instruction register is loaded with the instructions fetched during the previous epoch and decoding starts. In the epoch defined by T_0 to T_1 the operands A and B are read from data memory. During the intermin between T_1 and T_0 , selected function module operates on the buffered operands and reports the results to the data bus.

The timing cycles between the processor elements are not the same. In order to accommodate the variation in timing, linkage between the decoding control and the

timing generator is accomplished. In effect, spacing can be varied in discrete increments, according to the type of instruction being decoded. This is known as adaptive timing.

In summary, there are two main disadvantages;

1. Because the control mechanism is sequential, full utilisation of the potential parallelism of the functional modules cannot be achieved. Hence, only some of the hardware is active during each cycle.

2. The adaptive timing will cause a further complication in the system control.

5.5. LSI GENERAL PURPOSE SIGNAL PROCESSOR

The appearance of LSI technology in the mid seventies provided a special opportunity for an economical solution to signal processing problems. In this respect, a number of LSI single-chip general purpose signal processors [83-87] were created and designed in order to capitalise on the new technology, in order to provide a quick and cost effective way to implement a broad range of signal processing applications.

The approach taken for the development and design of the LSI single-chip processor is based on the more general purpose computer architectures. Namely, there are based on the single processing element, control, and separate data and program memories. As an example we shall look at a number of these LSI processor.

5.5.1. THE BELL DIGITAL SIGNAL PROCESSOR

A block diagram of the Bell LSI single-chip processor [Boddie,84] is shown in figure 5.12. The digital signal processor is a general purpose building block which can be programmed to perform a variety of digital signal processing functions in the telecommunication application areas such as filtering, modulation etc. It is fabricated in NMOS technology and packaged in 40-pin DIP.

It has the following features;

1. A separate data memory (RAM) used for variable data, and delay, and

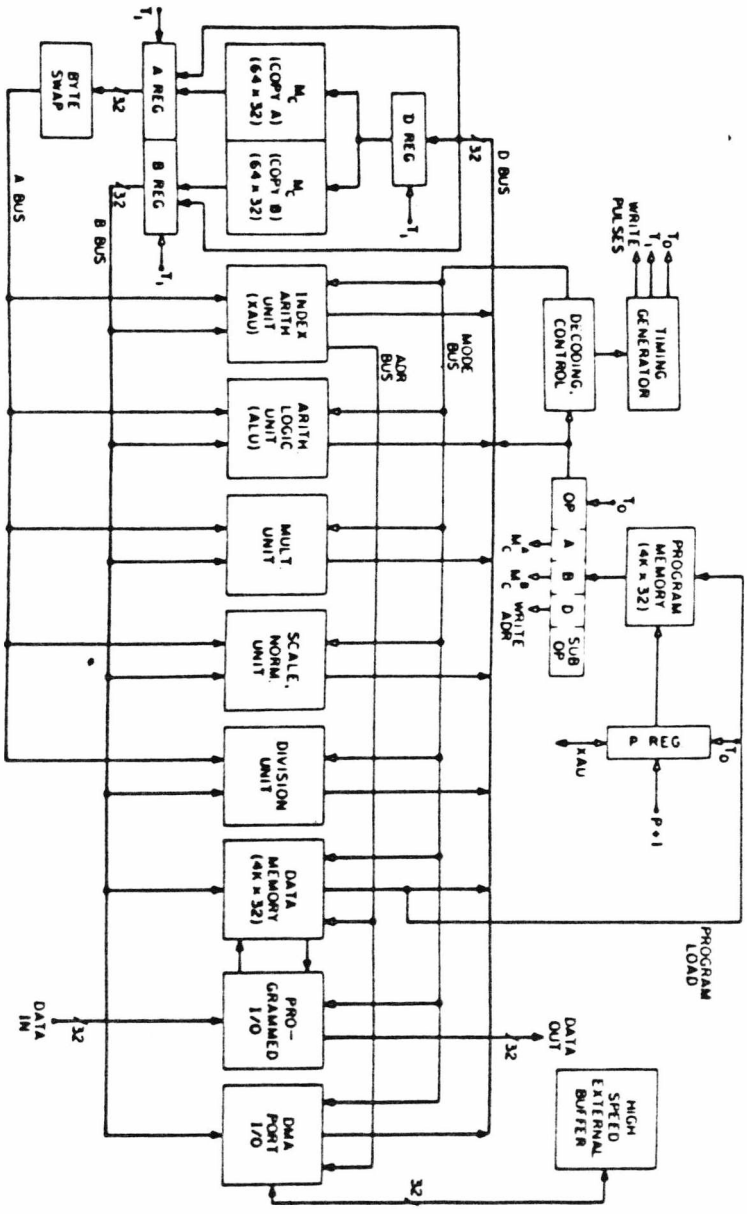


Figure 5.10 LSP/2 processor architecture

(ref. 72)

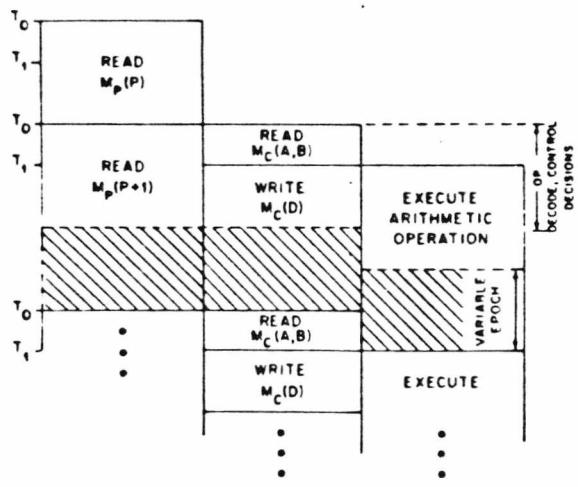


Figure 5.11 Overlapped timing (ref. 72)

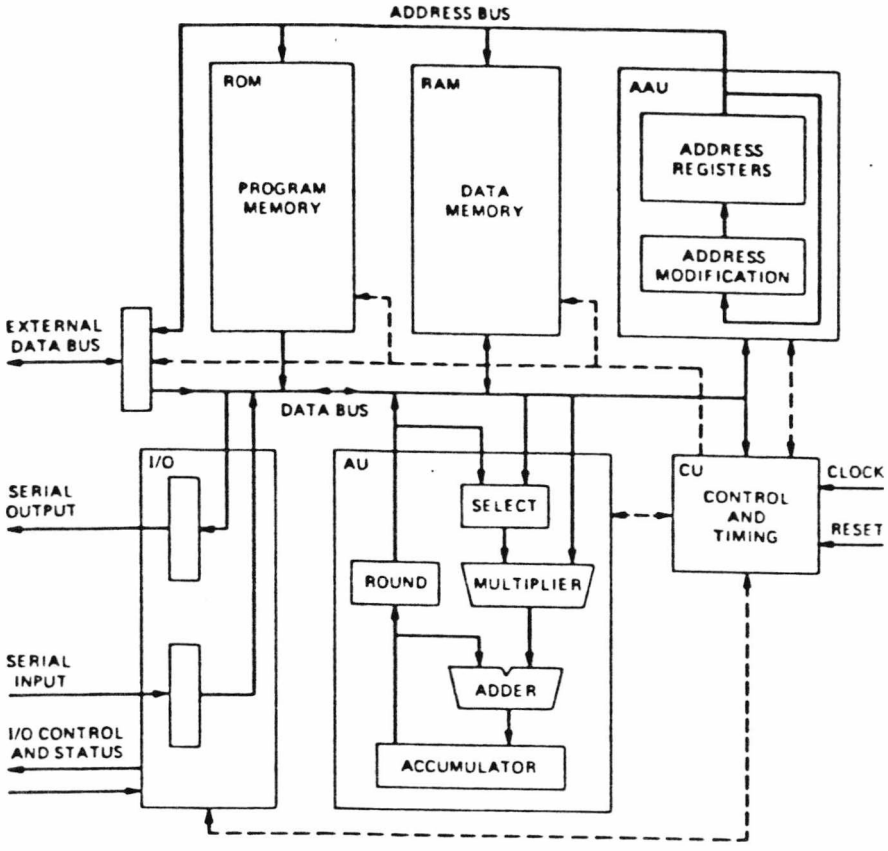
program and a fixed data memory (ROM) which they can be accessed in parallel.

2. An address arithmetic unit which generates addresses for the ROM and RAM.

3. An arithmetic unit which accept a 16-bit and 20-bit operand to form a 36-bit product, and which accumulates the product with a 40-bit accumulator, and rounds the accumulator to a 20-bit word for storage.

4. A serially input/output port.

The processor is also able to access an external memory, with no reduction in processing speed. The arithmetic unit is pipelined in three ways: i) the formation of the product of two set of data, ii) the addition of the product and previous result and



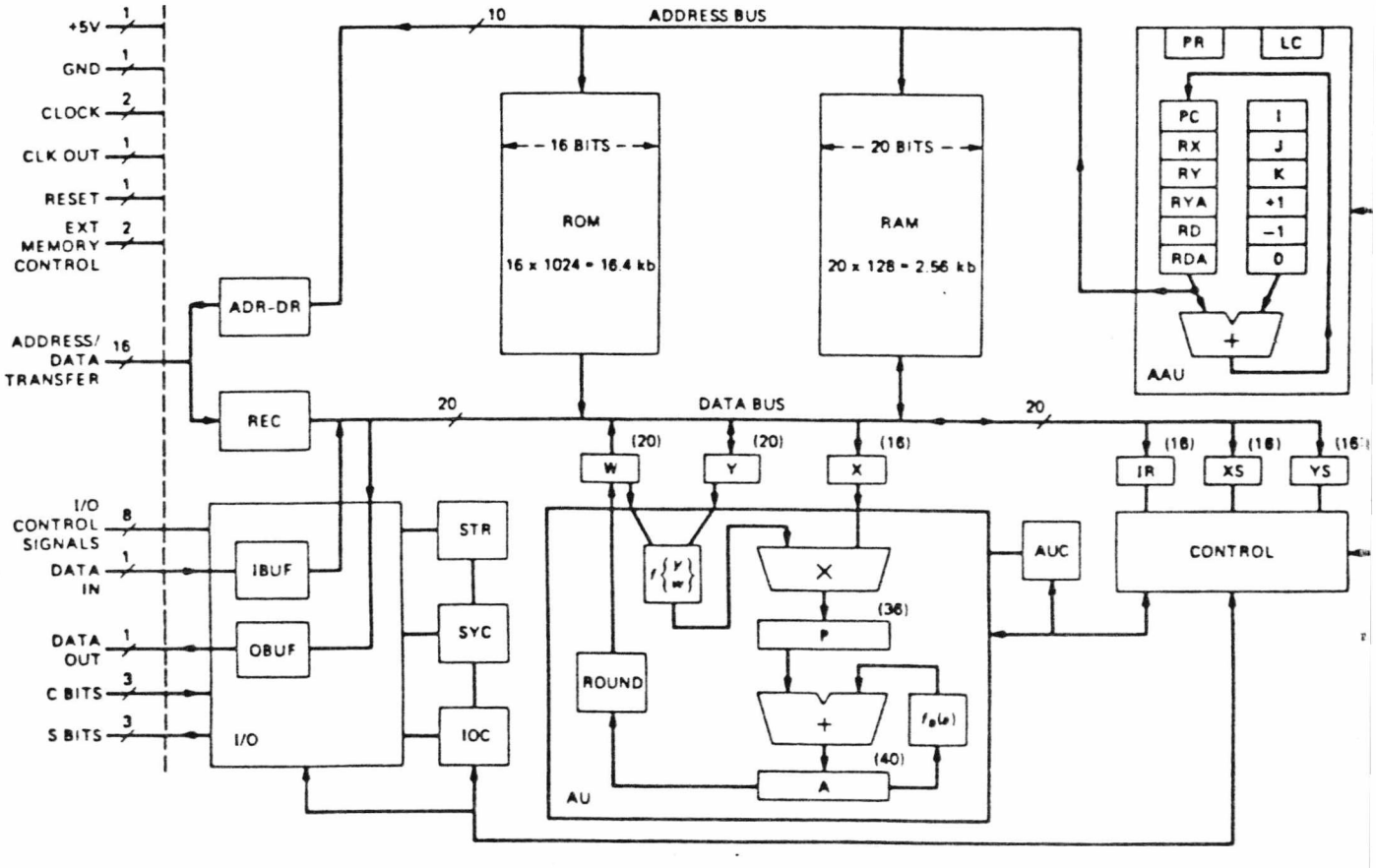


Figure 5.12 The block diagram of the BELL LSI signal processor [ref 84]

iii) the transfer of the accumulator to memory. This pipeline structure keeps all parts of the arithmetic unit busy at all times and allows the processor to maintain a high throughput. Input and out put are handled through an 8-bit buffers with an automatic serial-parallel conversion.

Control of the processor is done by the control unit. Instructions from ROM are latched into the instruction register and subsequently decoded in the control unit. Then the decoded signal is transferred from the control unit to the other blocks and registers as needed.

The programming is done in assembly language. It has two types of instruction: arithmetic and auxiliary. The arithmetic instructions control processor computation in the arithmetic unit in order to evaluate the required function. Auxiliary instructions are used to control noncomputational aspects of the processor such as initialisation of registers.

Because of the pipeline nature of the arithmetic unit, each instruction must contain up to four statements, one for each hardware components for example multiplier, accumulator, register, and store. In this case, in order to speed up the operation of the processor, each instruction is fetched from the ROM two cycles before its execution begins. This allows time to decode the instruction before execution begins. However, the prime difficulty will occur in the jump instructions. In this case, that two instructions the follow the jump are already in the operating hardware when the jump takes effect, and their data field will affect instructions which follow the jump. They may even differ from the data filed which would be fetched if the jump destination were reached by normal program the counter. In this case, the assembly cannot determine any differences, and hence produce some strange result.

The performance is measured in terms of amount of signal processing and can be performed by the processor. This obviously depends upon the cycle time, which is the time for basic machine operations, such a multiply or register setting.

In summary, the digital signal processor provides a solution to many voice band

application areas. Its particular advantage is the separation of data and program memory plus arithmetic operation operations done in the pipeline.

However, the main disadvantage of single-chip processor is the small amount of on-chip memory.

5.5.2. THE REAL-TIME SIGNAL PROCESSOR

The Real-Time Signal Processor has a fully programmable signal processing architecture with external data and program memory. A block diagram of the Real-Time processor [Mintzer,85] is shown in figure 5.13. The machine is designed for implementing the signal processing algorithm, and especially in the area of telecommunications. This machine is designed with consideration for easy programming, since the application software dominates the cost of a processor development time. The broken lines represent the chip boundary. The Real-Time processor interfaces with the external world through three parallel ports, one control and two data lines. The data port will allow for a faster flow of data to and from the processor, while the control port is used for loading programs into the machine, via a fully parallel Input/Output interface. The data and instruction memory are external, hence they provides flexibility but need more control instruction.

The Real-Time processor architecture consists of four functional subunits. They are:

1) The instruction fetch and sequencing

2) The data store address generator

3) The data store access

4) The arithmetic unit

Unlike the Bell processor[Boddie,84], the arithmetic unit of the Real-Time processor does not have a fast parallel multiplier. However, multiplications are mechanised by shifting the multiplicand in one of the arithmetic unit registers and accumulating the partial sums in the second arithmetic unit register. The reason for not having a fast multiplier hardware is traded off against its flexibility, and provides the other hardware

with easy programming and with addressing of the data.

An instruction pipeline is used to increase the performance. Most Real-Time processor instructions use each of the first four units once, but only one of them is used during any cycle. In the first cycle of the instruction execution that instruction is fetched, followed by computing the data store address in the second cycle. In the third cycle the data store is accessed and in the forth cycle the arithmetic or logical operations are carried out as shown in figure 5.14.

The Real-Time processor is a one address architecture and has four addressing modes. These are, direct addressing, offset addressing, index addressing, and mask addressing. The index and mask addressing use two index registers which are extremely useful for testing, and the completion of loops. This address generation technique will provide the Real-Time processor with a robust of addressing capability, and it will also spend a sizable area of the chip.

One major disadvantage of the processor is the presence of the instruction pipeline. This instruction pipeline in conjunction with one of the addressing mode will create a pipeline hazard. That is, the processor, in an instance of time will want to transfer the result of an operation which is not yet computed. In this case the user needs to provide a no operation instruction in his application program. But this is a formidable task from the user's point of view. Hence, the processor needs a software support which accepts the naively written code and converts it to conform with the instruction pipeline. Hence, one needs a number of hardware supports for this procedure.

In summary, the Real-Time processor architecture was strongly influenced by the desire to make programming the processor as simple as possible. Hence, a number of desirable hardware supports such as a fast parallel multiplier for coefficient multiplication, have been traded-off for the hardware support of easy programming.

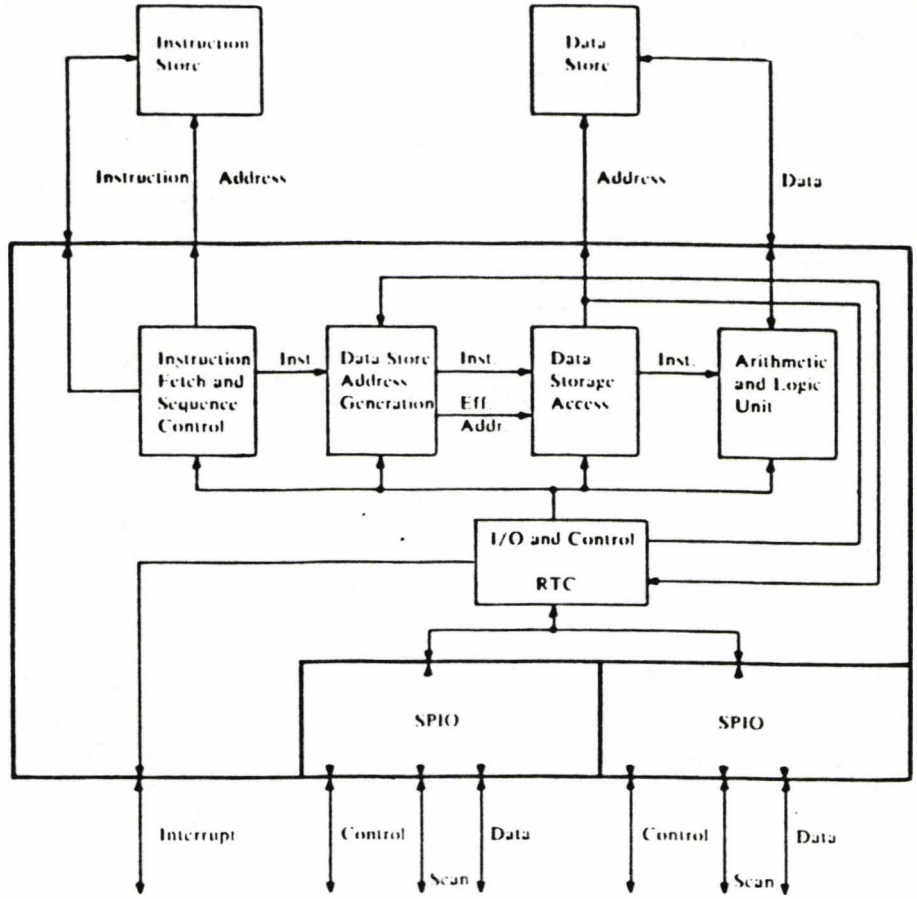


Figure 5.13 The block diagram of the Real-Time signal processor

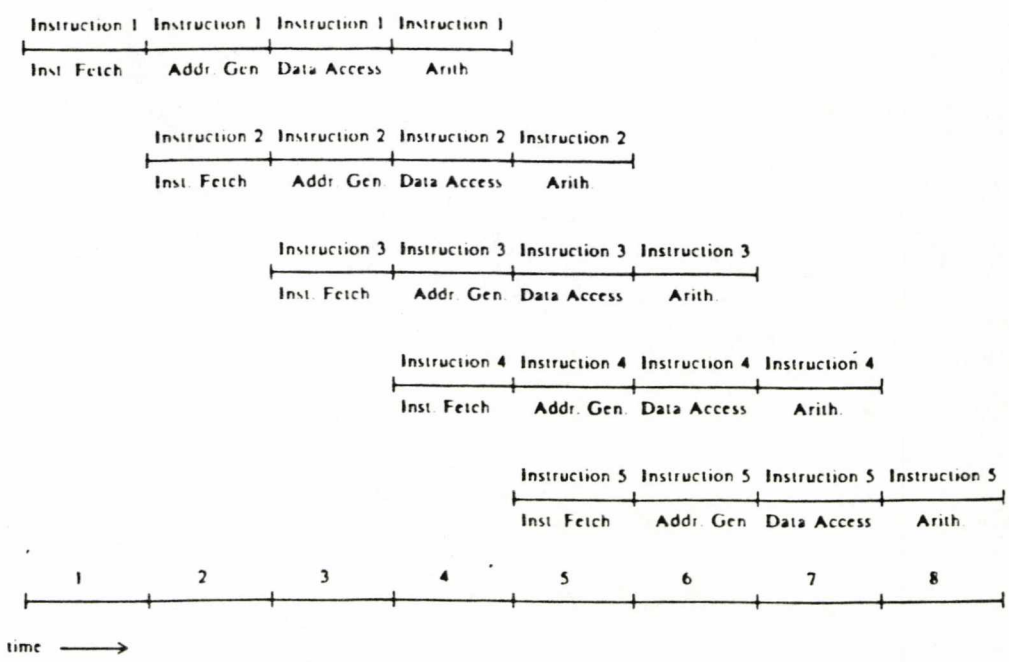


Figure 5.14 The timming diagram of the instruction set [ref 85]

5.6. CONCURRENT AND PARALLEL PROCESSING

Many signal processing algorithms, such as Number Theoretic Transforms, filtering etc. often deal with the array of data. It is possible to design computer systems which will speed up the computation of such data arrays. Computers which are designed to process large amount of data quickly are known as array or vector processors.

Parallel computers can be classified in terms of parallelism within the instructions and parallelism within the data. Flynn[89] observed that the method for achieving parallel operation depend on replicating the instruction stream and data stream. This gives rise to four classes of computers;

I) The Single-Instruction Single-Data stream (SISD) computer is a serial computer which has already been mentioned in the previous section.

II) The Single-Instruction Multiple-Data stream (SIMD) computer is an array processor.

III) The Multiple-Instruction Single-Data stream (MISD) computer, which each operand operates upon simultaneously and by using several instructions.

IV) The Multiple-Instruction Multiple-Data stream (MIMD) computers.

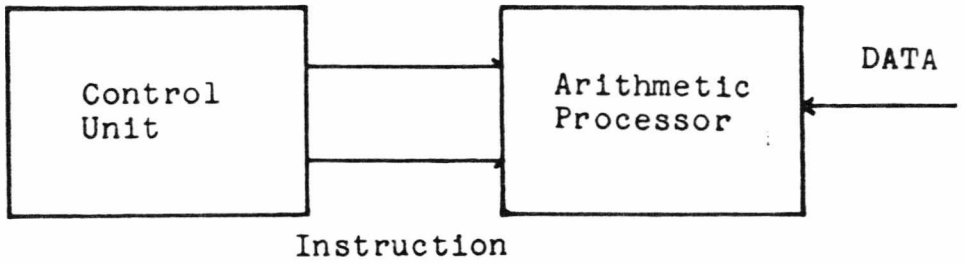
see figure 5.15

The array processor consists of a number, N, of identical arithmetic and memory units, all controlled by one control unit (which itself a computer). These processors are of the SIMD type class of parallel computers: figure 5.15b. Clearly if all the data is present at a right place the array processor can exhibit an N fold increase in the processing time.

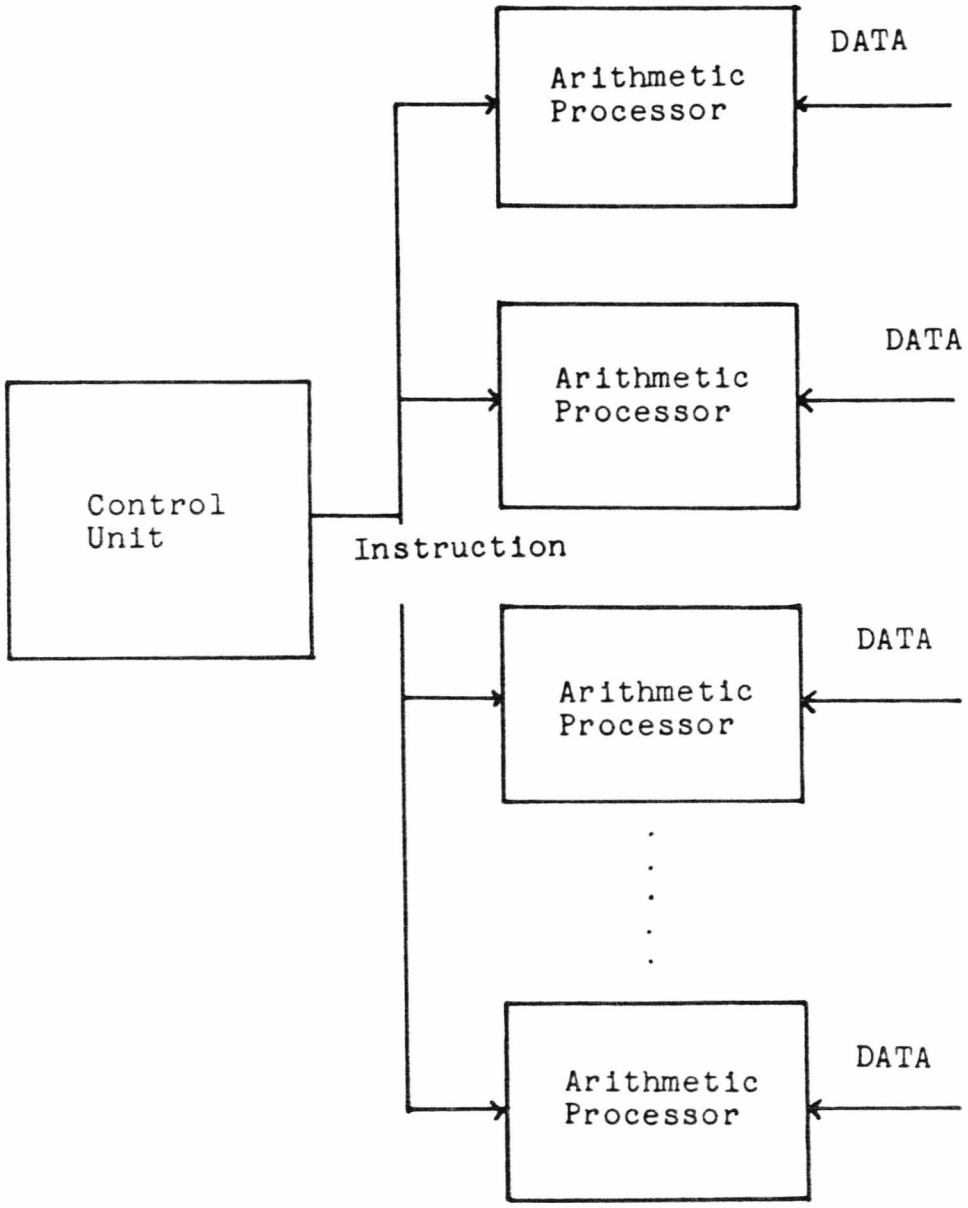
This sort of architecture at first sight seems quite ideal for some signal processing algorithms. However, there are two difficulties with this type of architecture

1) Control flow

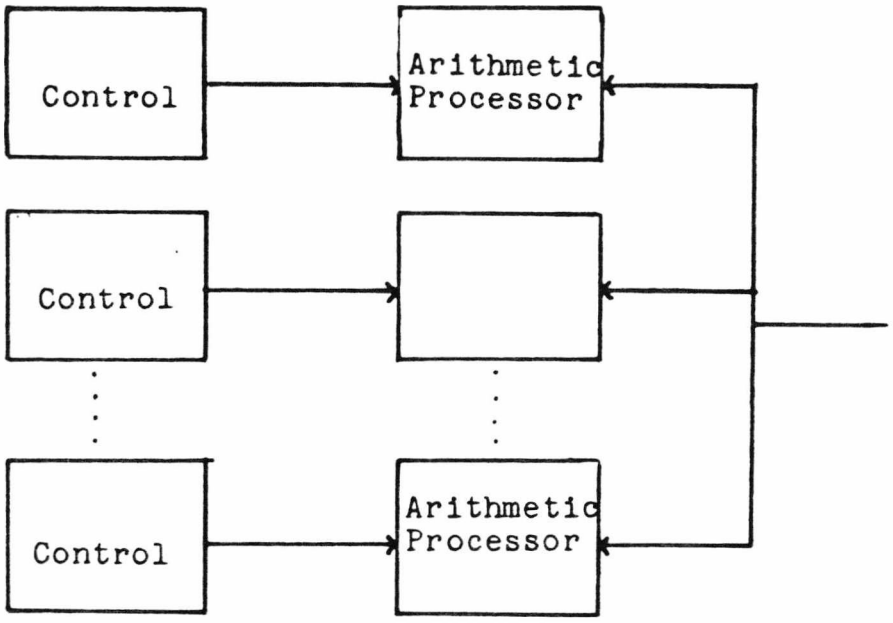
Conditional branches are particularly vexing in a SIMD computer. Suppose, for


Control
Unit
Arithmetic
Processor
DATA
Instruction

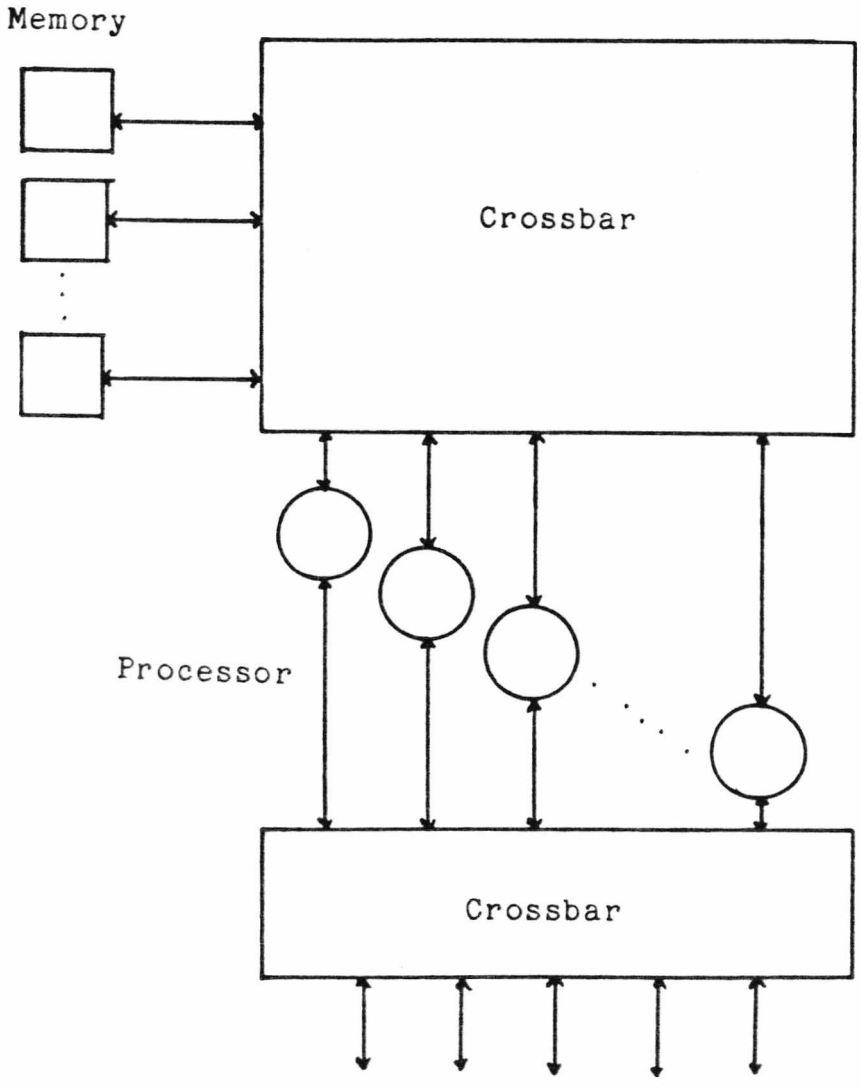
a) SISD


Arithmetic
Processor
DATA
Arithmetic
Processor
DATA
Control
Unit
Instruction
Arithmetic
Processor
DATA
Arithmetic
Processor
DATA

b) SIMD


Control
Arithmetic Processor
Control
Control
Arithmetic Processor

c) MISD


Memory
Crossbar
Processor
Crossbar

d) MIMD

example an action must be done if the result of a calculation is zero and different one must be done otherwise. What is to be done if some of the processors have results that are zero, and some do not? Because all the processors obey the same instructions, it is not possible to split the execution streams as one would desire. In this case, one has to test for zero conditions for those processors with zero results, and test for those processors with a non-zero result. In order to implement this facility a control bit is required in each processor. Which can be set or cleared, and will cause a processor to ignore the instructions. The processor which fails the test can be masked out.

Using this technique conditional instructions can be executed on an array processor. However, some of the processors are idle during the execution of conditional instructions, thus reducing the effective parallelism.

2) Data flow

A SIMD structure assumes that each memory unit can provide the necessary data to its related processor. However, the ability to do this is dependent on the way the data is stored in the memory. The basic constraint which hampers the utilisation of memory is, that it can only access a data per memory cycle. In the most favourable case the N data of a vector instruction lies in distinct memory, and thus can be fetched simultaneously. In the least favourable case, the N data lies in a single memory and must be fetched sequentially. In one dimensional cases the vectors involved are normally stored so that one element of the vector is in each memory module. However, the problem arises in the two dimensional cases. Stone[90] presents two way of storing the data in the memory which can be accessed in parallel.

The second problem which arises in the SIMD machines is, that which occurs when a processor is required data from its own local memory and from elsewhere. This is particularly the case for transform algorithms such as NTT. This problem arises the question of interprocessor communications. The ideal interconnection network between the processors and memories would be a cross-bar exchange. This would allow any processor to connect to any memory in one machine cycle.

Unfortunately, the cost and complexity of such networks is too high. The simplest possible interconnection is the nearest neighbour one. This has the advantage of cheapness and modularity. However, there are other interconnections which can be found in the literature.

Another alternative way to processing vectors, is to use a highly pipeline structure. Such structures are known as MISD machines. A block diagram of such a machine is shown in figure 5.15c. It contains its speed by having a highly pipelined arithmetic unit perform such basic operations. The stages are isolated from each other by registers. By cascading several of these stages an arithmetic operation can be performed at higher clock rates. The problems of MISD machines are again data and control flow, but they are different to those of SIMD machines.

In the case of control flow, the speed of pipeline architecture will be reduce by a large amount when conditional instructions are performed. This is because a new instruction must be fetched, which means that all the data must be discarded, and thus the time taken to access that data and operate on them is wasted. Therefore, in the worse case M cycles of processor time are wasted by one conditional branch. Also, the necessary controls and interlocks which ensure that the correct sequence of instructions are obeyed can be quite complex. A related data flow problem is which makes sure that the memory locations which may be modified by the output from the pipeline are not used until this has occurred. Another problem is the memory bandwidth. Hence the accessing of operands form memory will have a major effect upon the overall performance of the machine.

The problem caused by the conditional branch in the array(MISD) processors can be overcome by allowing each processor to be autonomous and execute its own instruction stream. Then, when a conditional branch occurs each processor will continue to follow its own instructions, and hence parallelism is restored. A further advantage is that completely different programs can be executed by different processors, allowing greater parallelism in tasks which require many different operations. This structure is known as an MIMD machine. A block diagram of such a machine is shown in figure 5.15d.

The interconnections amongst these modules are extensive as shown in figure 5.15d. A switch connects every processor to every memory. The switch is an NxN cross bar, where every N^2 cross point is a potential connection. As N increases, the complexity of cross bar network will increase enormously. To overcome this problem a technique known as bus interconnection is used [Angus,]. Each processor can access its own local memory without using the main bus, unless it requires data contained in a memory other than its local one. This works well only for a small number of processors per bus.

Another approach is to have several bus connected in hierarchy. Several low level busses with a small number of processors per bus are connected together by a large bus which can be connected to another one. This structure reduces bus contention by having several busses running in parallel. However, accessing via a higher bus take more time.

Further problems of MIMD machines are the related ones of synchronisation of separate processors and controlled use of shared data. As MIMD machines can execute different programs there is no guarantee that they will be in step. If one processor need some data that is being calculated by the other one the data may not be available when the program get to the part that requires the data. Hence, there is a need for some form of control where by each processor should inform each other. This will greatly increase the overhead control. The final problem which relates to the synchronisation is the possibility of the deadlock, which occurs when a processor is waiting for some results from other processors.

In general the major problem with these classes of parallel computers is the intercommunication between the processors and memory units. This is due to the fact that most of these processors are of the Von-Nueman. That is every processing element has to communicate with its local memory in order to retrieve the data. It is this bottleneck and hence the memory bandwidth which will reduce the speed of operation.

As we explained earlier most designers have used microprogrammed computer-like architecture to retain a high throughput by providing hardwired multipliers and addressing units, separating the program, data and coefficient memories, and adding a complicated multiple bus structures in order to avoid bottlenecks. In this case programming is used to control the flow of data through each processing elements.

5.7. SYSTOLIC ARRAYS

One could look at the digital signal processing algorithms in a different way. Because digital signal processing algorithms will process the same computation all over, it is possible to describe most digital signal processing algorithms strictly in terms of data-flow graphs. For example, the NTT of figure 5.16. Note that the flow of data is regular and there is no need for conditionals of any kind, and hence it is preferred to use special purpose processing elements. In fact this is in contrast to the general path computers where the flow of data is control either with a high level or low level programming language. However many designers have used the general data path computers rather than special purpose processing elements. This is because the overhead cost of using the special purpose machines was too high to be economical.

With the great advances made in the integrated circuit technology in recent years and in particular VLSI (Very Large Scale Integration), it is now possible to design and implement special purpose processing elements cheaper than before. Therefore the cost-effectiveness can be reduced by the use of appropriate architectures. If a structure can truely be decomposed into a few types of simple substructures which are used repetitively, great saving in terms of design, implementation, and fabrication time can be achieved. This is specially true in the case of VLSI designs.

In order to utilise the potential advantages of VLSI technology new design philosophy and methodological concepts have to be defined. Mead and Conway[91,1980] have set up such concepts:

1) A few different types of simple cells.

2) Simple and regular flow of data, so local and regular interconnection can be achieved.

3) Use of pipelining and concurrency.

One solution to the above challenges is the concept of systolic arrays Kung[92,93,94]. A systolic array (system) is a collection of relatively simple processing units, either all of the same type or a mixture of a few different types, which are connected by a simple communications network and which operate in parallel. The basic principle architecture of a systolic array is illustrated in figure 5.17.

By replacing a single processing element with an array of processing elements, a higher computation rate would be achieved without increasing the storage bandwidth. In this case, the array uses a set of data which has been retrieved from the memory many times over without having to store or retrieve the intermediate result, thus allowing speedup relative to memory bandwidth. Kung[94-1979] has shown that a linearly connected array, figure 5.18, can be used to multiply an NxN matrix by a vector of N elements using a N/2 processor, each of which performs the inner product operation of $Y=Y+X.W$. As the matrix and vector shift into the array, they always move in the same way and hence require no control. In this case each processing element performs one computation at each step. Kung further used the same technique of linear systolic array in applications where two sequence of data have to be convolved [93,1982]. He illustrates several systolic convolution array structures:-

a) A systolic convolution array with global data communication

b) A systolic convolution array without global data communication

In design a) a broadcasting technique has been used, where one sequence is preloaded to the cells, one in each cell, and stays in the cell throughout the computation, while the partial results move systolically from one cell to another, that is each of them moves over the cell during each cycle. At the beginning of a cycle one data is broadcasted to all the cells, this is shown in figure 5.19. During each cycle the partial result moves from the left-most cell to its neighbouring right cell, and the final

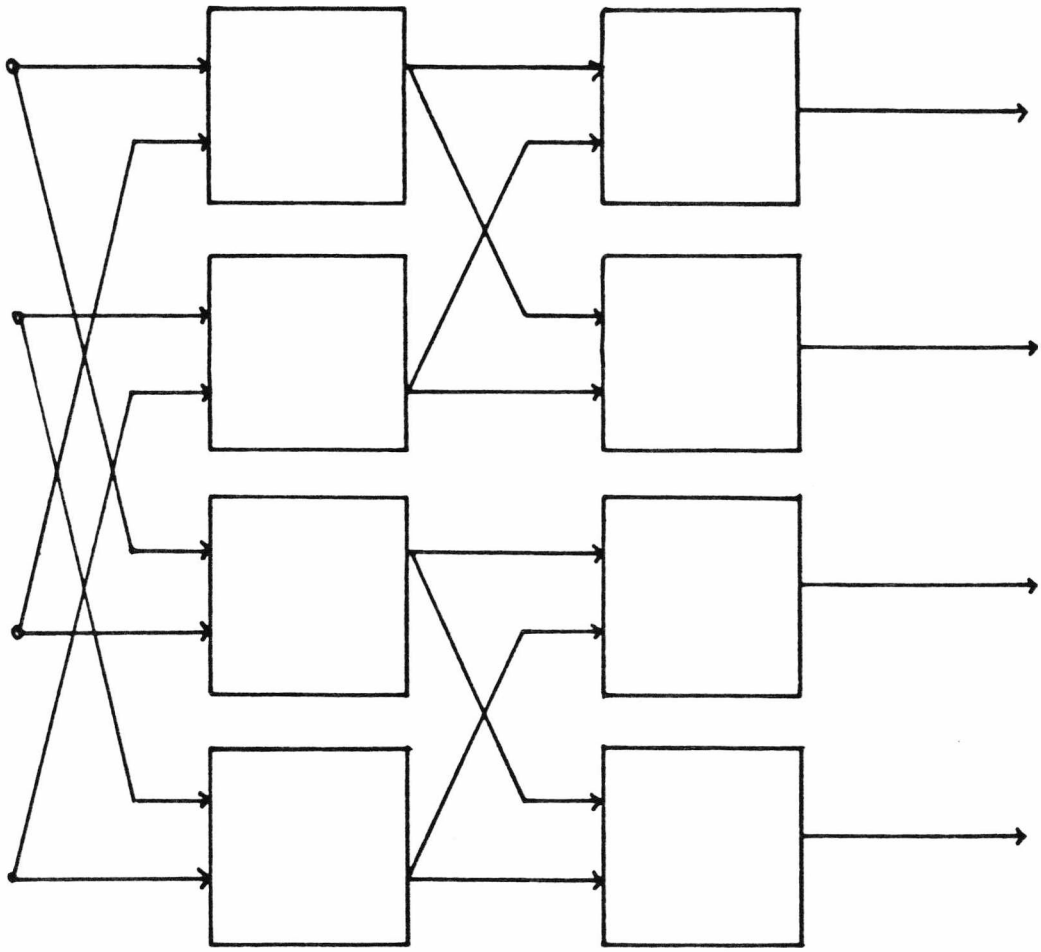
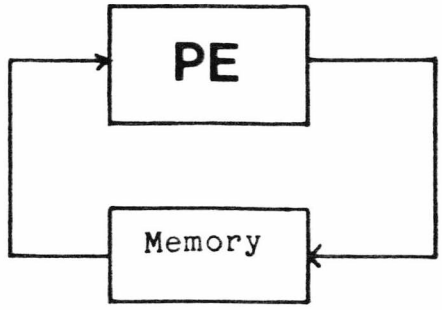


Figure 5.16 NTT structure



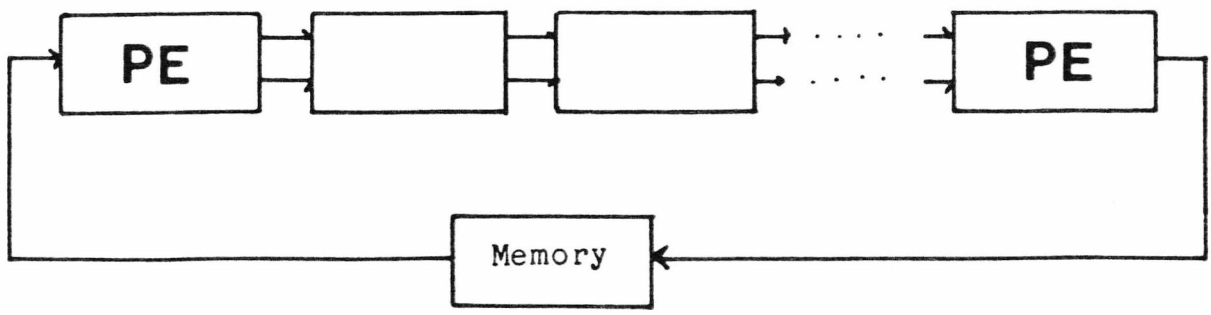


Figure 5.17 Basic principle of systolic array structure

result is obtained from the right-most cell at a rate of one result per cycle.

The opposite of broadcasting is fan-in through which data items from a number of cells can be collected. This is shown in figure 5.20. Designs of this type, using the fan-in, have been known for quite a long time.

Although global broadcasting or fan-in techniques have partially solved the speed problem, implementing it in a modular and expandable way presents another problem. Providing a data item to all the cells in the systolic array requires the use of a bus. As the number of cells increases, wires become long for a bus, and hence expanding these non-local communication paths to meet the increasing load is difficult without slowing down the system throughput, which is very significant at chip and board levels. However, Kung argueed that it is possible to overcome this engineering difficulty by using the array structure without global communication. In this case both sets of data will move either in the same direction or in opposite direction as shown in figure 5.21. The problem of the case is that only half of the cells are active at one time so it will lose some computational power.

A number of author's have investigated the implementation of the digital signal processing algorithms by applying the similar ideas explained so far at bit level, rather than the word level, as suggested by Kung. Foster [95,1980] designed a special purpose pattern matching chip. It uses a linear array of cells as depict in figure 5.22. The chip achieved in two modules, each consisting of a linear array of identical cells. Cell A is a one bit comparator and, has one bit of the pattern (data1) flowing from left to right, while a one bit of the string (data2) is flowing from right to left and the comparison result for the pair of the characters flowing from top to bottom to cell B. Cell B will accumulate the result receives from the comparator above. It maintains a temporary result, and at the end of the pattern it uses the temporary result to replace it instead of the final result which flows from right to left. This structure is again an example of non-global communication which is most important in the chip design.

Another example of such structures is presented in figure 5.23 [Evans,96,1983].

This is a multi-bit convolver based on the bit level systolic array architecture. The convolver array consists of three distinct regions. Each region contains a different type of cell. However, the function of each cell type is similar, in that each contains some simple logic, a number of latches and either a half or full adder. The data flow through the array is as follow;

On one phase of the system clock, signals enter these cells from the direction indicated. The required results will be generated, and together with the necessary data are made available at the cell outputs on the opposite phase of the clock

The main convolution operation takes place in region 1. Within this region each coefficient row is associated with a particular row of the array. The interconnection of each cell is such that on each row the coefficient bits are moved from right to left, while the data bits move serially from left to right. The interaction which occurs on each row of region one, between bits of a given coefficient word and the incoming data, may be regarded as an individual multiplication operation, in that all the partial products required within the multiplication process are generated by the interaction between the two bit streams. However, rather than summing up these partial product to form a final result, each partial result is latched to the cell below. This movement will result in a diamond shaped region. The reason for this action is that each diamond shape is associated with a single convolution result. Hence, the corresponding partial products generated within a particular row of the array are accumulated as this diamond region moves down.

All partial products generated within a particular column of the array are accumulated in order to form the final result. This is done in region 2.In summary a single chip CMOS convolver has been implemented which is based on the systolic array. The design and layout are simple with regular interconnection between the cells.

Other authors have used the systolic array structure for the implementation of digital signal processing algorithms and can be found in relevant literature [97-98].

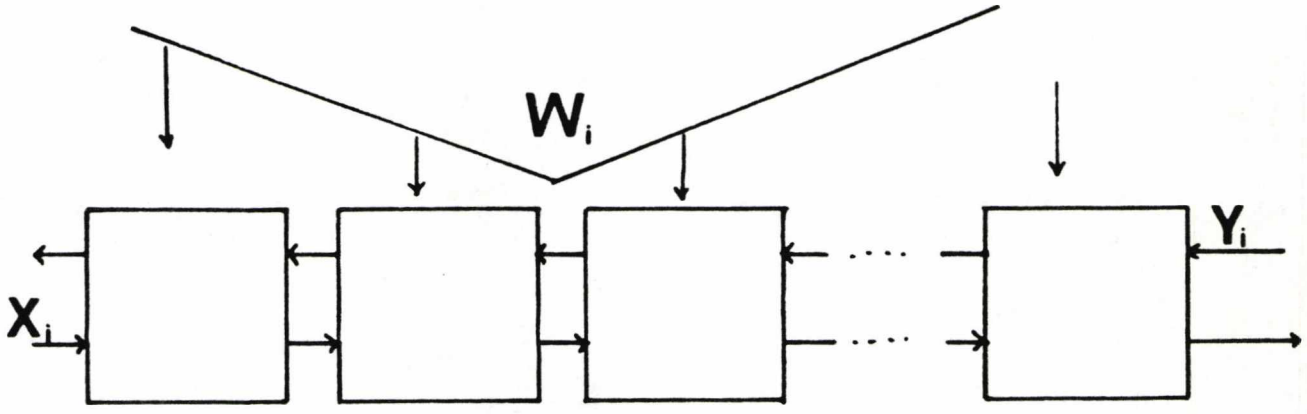


Figure 5.18 Systolic convolution array structure

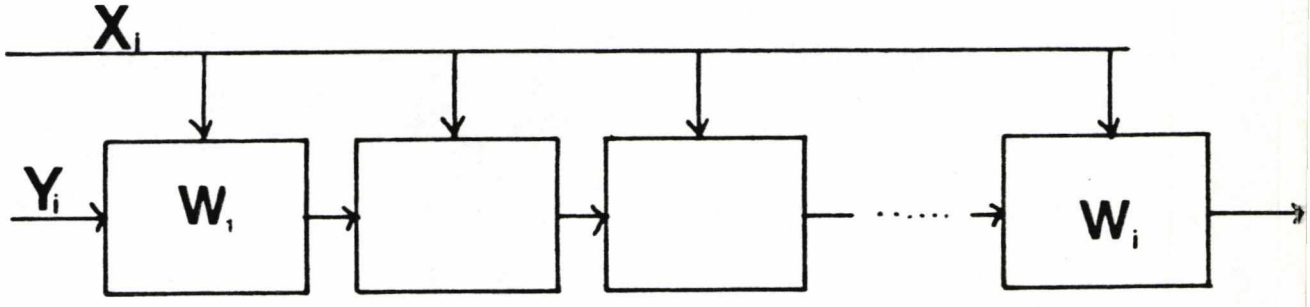


Figure 5.19 A systolic convolution array with global data communicatio

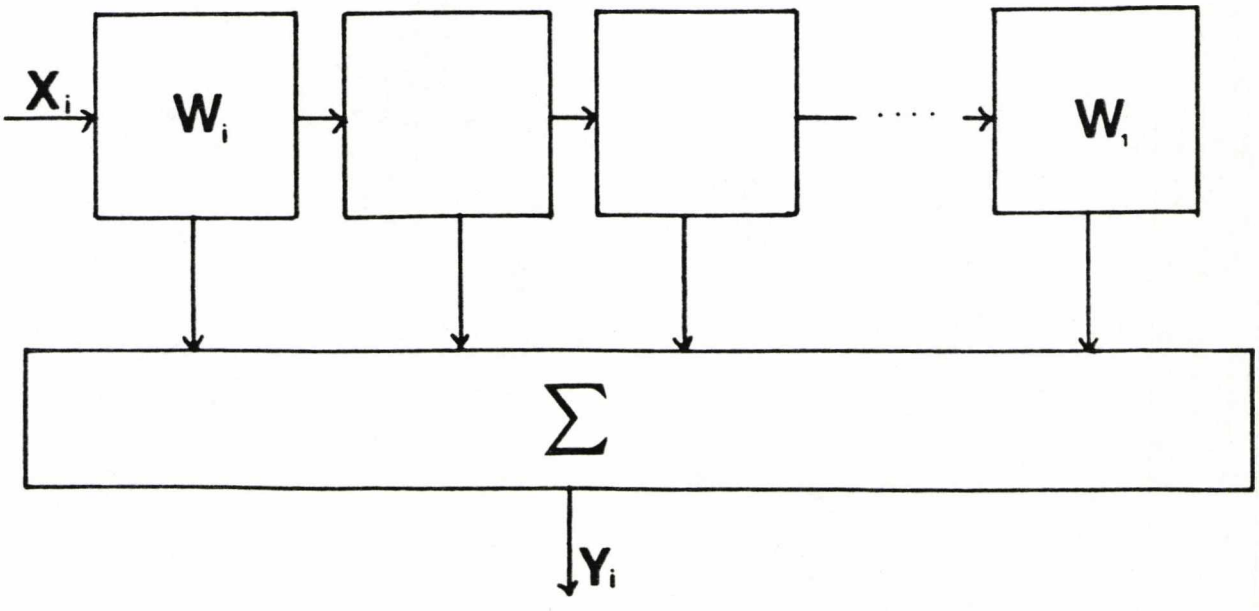


Figure 5.20 A systolic convolution array without global data communicati

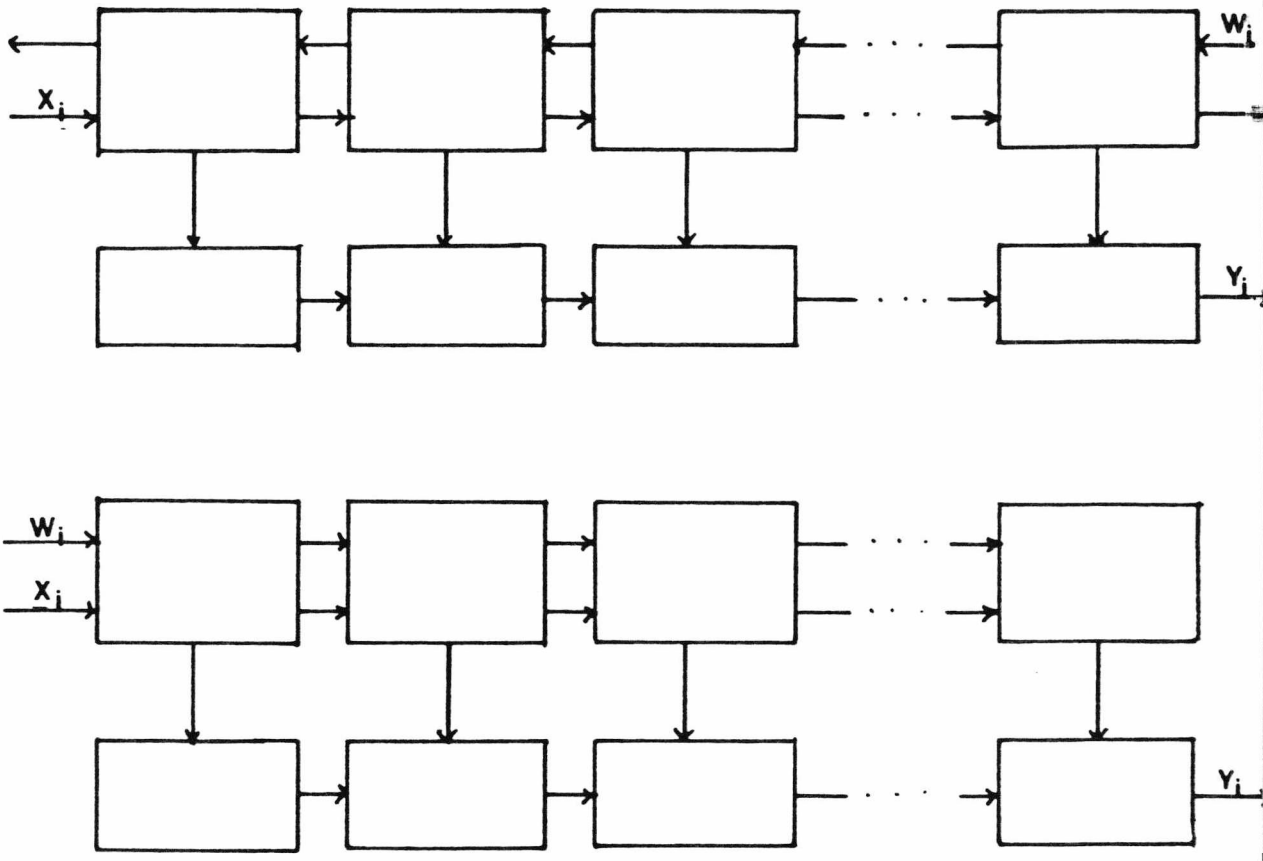


Figure 5.21 Systolic convolution array

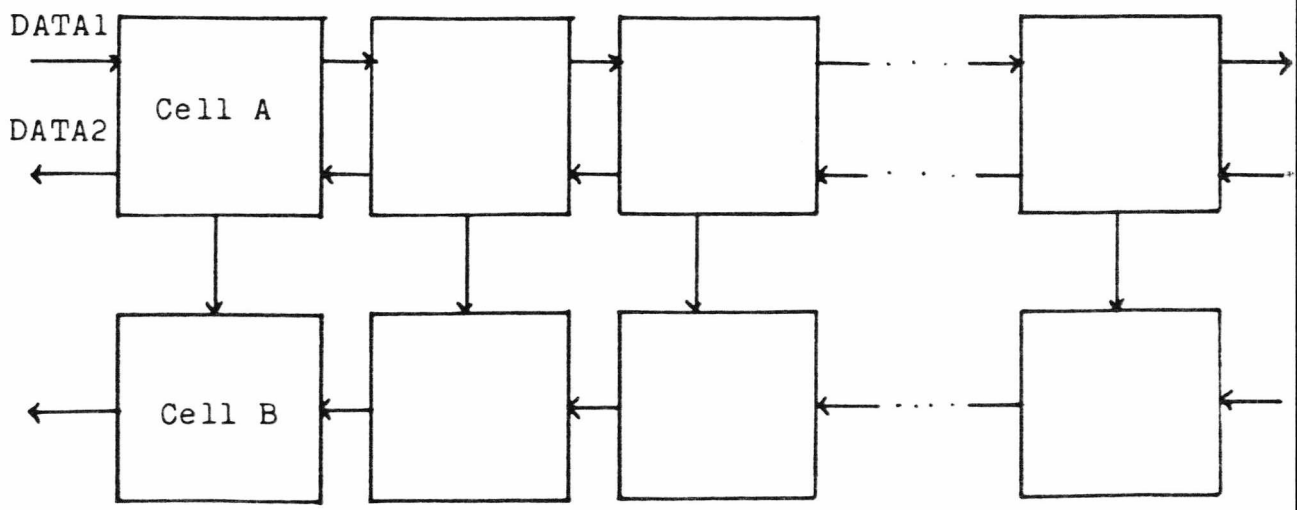


Figure 5.22 Systolic convolution array

In Summary, systolic array architecture of digital signal processing algorithms have a number of advantages over conventional structures, namely;

1) The design makes multiple use of each input data without the need to restore and retrieve the intermediate result. Because of this property high throughput can be achieved. The data flow communication between each element of the array can be achieved by either use of global communication, such as broadcasting or fan-in, or non-global communication. For modular expandibility the later approach is preferable.

2) The processing power of systolic arrays comes from concurrent use of many simple cells rather than sequential use of a few powerful processors, as in many conventional architecture. This can be achieved by pipelining the stages involved in the computation of each single result, or by multiprocessing many results in parallel, or by both.

3) Systolic arrays will use only a few types of simple cells, and hence reduce the overhead cost of design and implementation will be reduced.

4) The data and/or control flow in the systolic array are simple and regular, hence it is easy to expand the array structure.

Systolic arrays will avoid long-distance or irregular wires, or bus-structure for data communication, because long wires or bus-structure at chip level will cause long delays and therefore reductions in the throughput.

The only long distance communication (apart from the supply lines) is the system clock. Of course, self-timed schemes can be used instead of synchronising(clocking) the neighbouring cells. However, for the following two reasons the synchronisation is preferred to self-timing [99];

1) Each cell in the systolic array performs the same kind of operation as every other cell in the array, thus there is little variation in the speed at which the array will operate.

2) In cases where the variations do exist, the throughput of computation along the path in the array is limited by the slowest cell on the array.

In general, the systolic array architecture is simple, regular and expandable. The data flow is easy and simple, and once the data has entered the array can be processed without the need to communicate with the storage unit, which in terms is advantageous over the conventional parallel architecture. The control flow is very simple, and in most cases the data movement is controlled by a simple clocking system as opposed to the complex and complicated control for the conventional array processors. However, the only disadvantages of these architectures is the clocking skew between the two end cells(PES) in the very large linear systolic arrays. This limitation can be overcome by either lowering the system clock or introducing delays between each member of the array or simply folding the array.

SUMMARY

In this chapter we presented a survey of different digital signal processor architectures. The main advantage of general purpose processors is that they are flexible. However, their limitations are namely, speed, limited memory and data storage, lack of parallelism, and the need for efficient control mechanism. The advantage of special purpose processors is that they are built to perform a specific function hence, operate at high clock rate. This makes them economical to built especially in VLSI, as one needs to produce the hardware necessary for that function. The main disadvantage of these is inflexibility. As for the concurrent and parallel processors architectures, it can be concluded that systolic arrays are better suited for VLSI implementation in terms of data communication between the elements of the array, simple control, and regular interconnection between the cells.

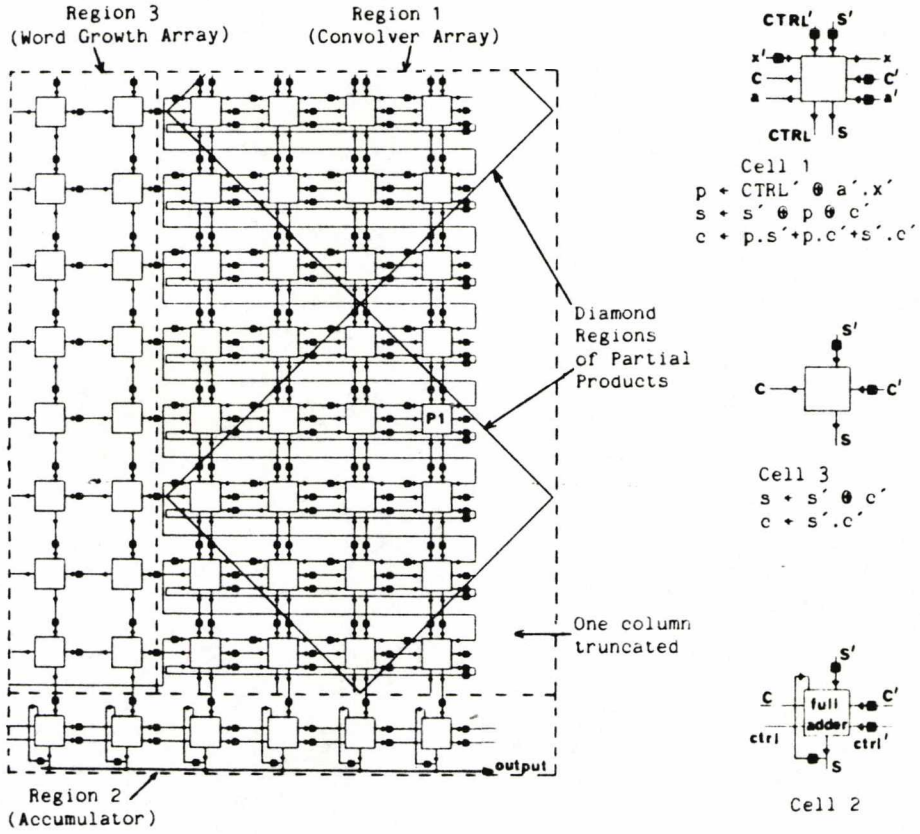


Figure 5.23 Cell structure of convolver circuit (ref. 96)

CHAPTER 6

THE VLSI ARCHITECTURE FOR ADAPTIVE DIGITAL FILTER

6.1. DESCRIPTION OF THE VLSI ARCHITECTURE FOR ADF

In this chapter the design of a special purpose VLSI processor architecture for an Adaptive Digital Filter is presented. It was designed to have the following characteristics;

1) Highest possible speed

2) Ability to perform arithmetic operations required by NTT

3) A few simple and regular processing elements

4) Use of pipelining and concurrency

In this chapter we first outline the design considerations taking into account the new concepts in the system design utilising the VLSI technology, and then give an overall view of the machine. The constituent parts are then be examined in more detail down to the circuit logic and layout of each cell. Particular emphasis will be placed on the following;

1) Simple and regular processing elements which support the NTT arithmetic operations

2) Local and regular communication between the processing elements which minimises the time and silicon area

DESIGN CONSIDERATION

As explained in previous chapter, most digital signal processors use many off-shelf LSI and MSI integrated circuits to design and implement a general purpose system, and with an aid of programming it was possible to do as many as signal processing algorithms in a cost-effective manner. However, with great advances in the integrated circuit industry it is possible to design and implement a cost-effective special purpose processor for a number of signal processing algorithms. In this respect it was decided to design a special purpose processor for an adaptive digital filter utilising the VLSI technology.

In order to fully materialise the advantages of VLSI technology for designing a special purpose adaptive digital filter processor, new design concepts have to be defined. The cycle of designing such a system can be roughly broken down into a number of step as follows;

1) Task definition

2) Design

   algorithm

   system level

   logic

   circuit level

3) Fabrication

If all these three cycles are used hand in hand then the potential advantages of the VLSI can be realised. However, in this thesis we shall only concentrate on the issues concerning the design phase.

Our system design strategy is based upon the "top-down" design methodology, that is, the problem(algorithm) which has to be solved is decomposed into smaller subproblems. Once we achieve the decomposition it is then possible to define the functional block for each subproblem into more detailed block diagram until we get down to the low-level operators such as adders, multipliers, etc. Then by assembling low-level operators into higher ones it is possible to say that the system is designed.

Based on the design methodology expalined, we first examine the decomposition of the algorithm which has to be done mainly ADF, follows by the functional block diagram for which the algorithm can be implemented. Once this stage has been achieved, it is possible to define how each functional block diagram can be designed.

6.2. DECOMPOSITION OF THE ADF ALGORITHM

As explained in previous chapters the adaptive filtering can be done either in the

time or frequency domain. It was decided that, for greater accuracy and higher filter length, design a frequency adaptive digital filter. A block diagram of the ADF is shown in figure 6.1

It consists of three major parts, transformation from one domain into another, the adaptation, and inverse transformation.

Following the study outlined in chapter 3, it was decided to use the NTT for the transformation part. Recall equation 3.1

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot \alpha^{(nk)}$$

for k = 0,1,2,.....,N-1

It is fairly easy to see that a simple multiply-add technique can be recursively used in order to obtain the corresponding coefficients. However, in order to utilise the VLSI design model set out earlier, the NTT algorithm can best be implemented by a decomposition technique. A common approach for the decomposition of the NTT algorithm is to partition the algorithm into smaller parts, find a solution for the parts and then combine the solution for the smaller parts into a solution for the whole. This approach is known as Divide-and-Conquer.

The analysis of the NTT algorithm is as follows;

Evaluating X(k) given in the above equation is equivalent to evaluating the polynomial;

$$P(x) = \sum_{i=0}^{N-1} a_i \cdot x^i \qquad (6.1)$$

for x= $\alpha^0,\alpha^1,.....,\alpha^{N-1}$ where a_i is a length N input data vector.

The principle of divide and conquer will be applied to the polynomial evaluation of (6.1). If N is even and a power of two, then the polynomial P(x) can be written as;

$$P(x) = P_0(x) + x^{(N/2)} P_1(x)$$

Where

$$P_0(x) = \sum_{i=0}^{N/2-1} a_i \cdot x^i$$

$$P_1(x) = \sum_{i=0}^{N/2} -1 a_{(i+N/2)} \cdot x^i$$

Recursively applying this divide-and-conquer approach until we are left with a N-1 1st degree polynomials. Then substitute the value of x and add the results. This direct sequential approach to the evaluation of P(x) is not efficient and it takes a considerable time for real time signal processing. However, a more efficient way is as follows;

Evaluating the polynomial P(x) at a point $x = \alpha$ is equivalent to finding the remainder where the P(x) is divided by $x - \alpha$. Therefore the evaluation of the NTT is reduced to finding the remainder when N-1st-degree polynomial is divided by each $x - \alpha^i$'s.

Simply dividing P(x) by each $x - \alpha^i$'s is a long procedure. To obtain a faster algorithm, we multiply each $x - \alpha^i$'s in pairs, then multiply the resulting N/2 polynomial together until we are left with a polynomial Q_1 and Q_2 . Next we divide P(X) by each Q_1 and Q_2 , obtaining the remainders R_1 and R_2 each of degree N/2. Since $x - \alpha^i$ is a factor of Q1 and Q2, finding the remainder of P(x) divided by $x - \alpha^i$ is equivalent to finding the remainder when each R1 and R2 divided by each N/2 appropriate $x - \alpha^i$'s. We define the remainder R_j when P(x) is divided by Q_j as follow;

$$R_j = \sum_{i=0}^{N/2} -1 \; (\; a_i + \alpha^s \cdot a_{i+N/2} \;)$$

However if one were to compute the NTT of a sequence, one would work only with the coefficients, and that simplifies the procedure even further.

The best way to show how the above decomposition for the NTT algorithm takes place is by using a tree type graph. We use a binary tree as a model for the decomposition of the NTT. The decomposition procedure for N=4 (transform length) is shown in figure 6.2. At the top of the tree lies the NTT algorithm while on the consecutive level lies the remainder of the previous result divided by the corresponding $x - \alpha^i$'s. Since one only works with the coefficients, it is possible to present the

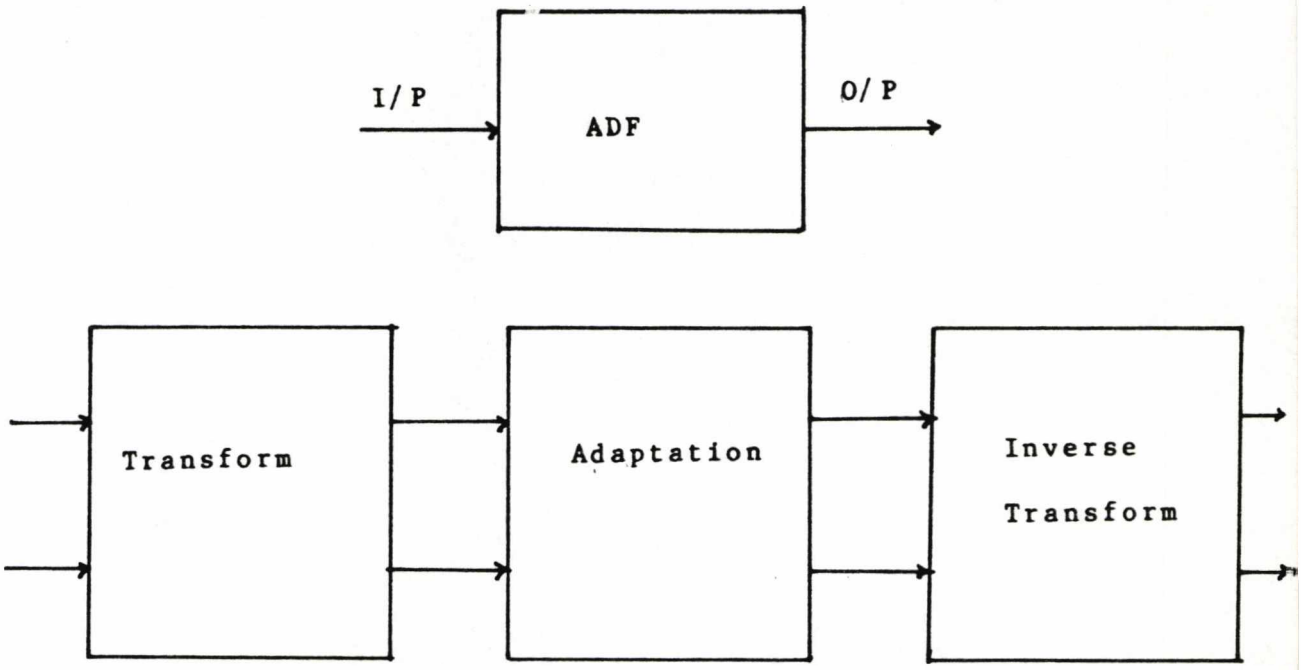


Figure 6.1

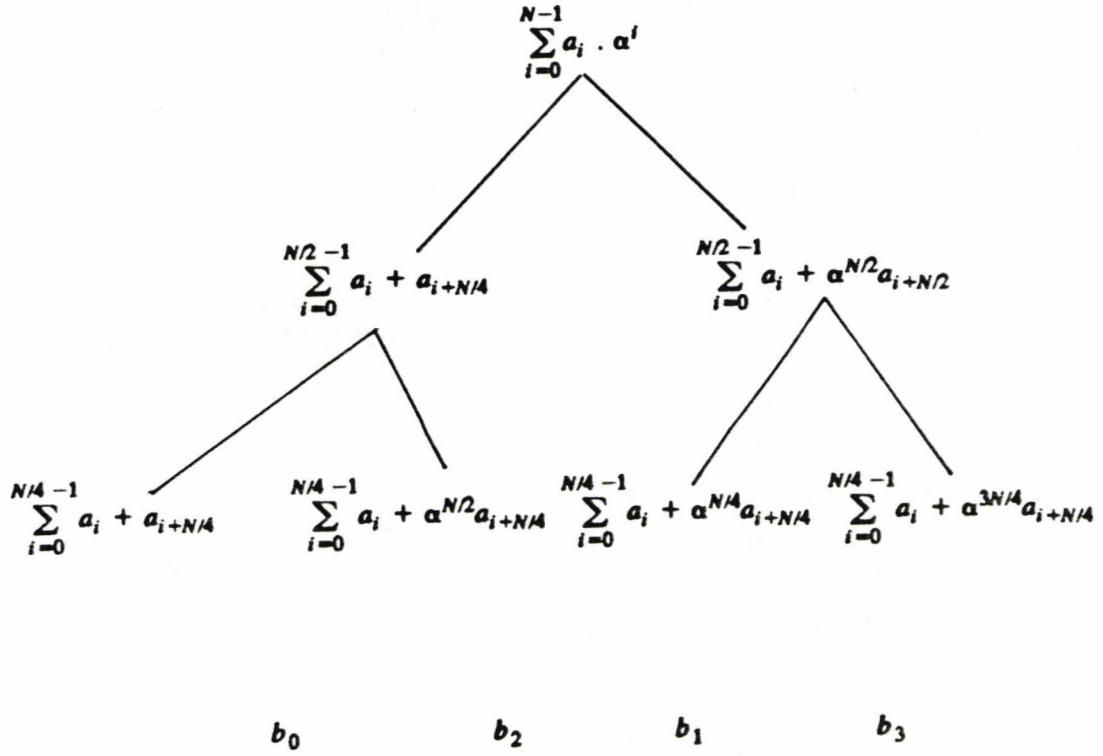


Figure 6.2

decomposition procedure strictly in terms of the input data. This is shown in figure 6.3. It can be seen that each node in any level of the tree will receive two sets of data from its previous level and after some simple arithmetic operation on the data, it passes the result to its descendant level for further processing until we end up with the base of the tree which is the final result.

This top-down decomposition of the NTT algorithm shows clearly that, the flow of data is simple and well-defined, as well as parallel and/or pipelining. Hence it is attractive for the VLSI implementation. The same procedure can be taken for the inverse transform algorithm. In the next section we shall look at some of the NTT structures which are suitable for the VLSI implementation.

6.3. VLSI ARCHITECTURE

Following the studies outlined in previous chapters, it was decided to design a special purpose processor. To this end attention was focused to the speed of the arithmetic operation rather than facilities to carry out the broad range of signal processing algorithms.

A broad overview of the frequency domain adaptive digital filter block diagram is shown in figure 6.4. It consist of three major parts;

1) The NTT processor

This provide the necessary arithmetic operation in order to transform a set of N-point data sequence to another domain for further calculation and processing.

2) Filter output and adaptation

This will provide the required filter output, by multiplying the coefficients of NTT(1) with the preloaded filter coefficients, as well as, performing the necessary operation needed for the adaptation procedure.

3) The inverse transform

This will process the filter output and present the results in the domain needed.

Since all the operations are done in the finite field, and for the reason given in the previous chapters, it was decided to use the FNT technique for the transform stages.

When designing filters, it is important to notice that the dynamic range of the filter should be sufficient, however, this is dependent upon the choice of the modulo M. For higher dynamic range M must be large, but the problem of large value of the modulo could cause great complexity and in most cases it is not realisable in hardware. In this case one can use a composite modulo $M = m_1, m_2, \ldots\ldots, m_L$ where each submodulo is mutually prime.

In this instance the adaptive digital filter is divided into L identical sub-block filters, each of which processes on the same set of data but on different modulo, and the final result can be achieved by accumulating the outputs of each sub-block. And since the result of one sub-block is independent of the other sub-blocks, greater parallelism can be achieved. This new adaptive digital filter block diagram is shown in figure 6.5.

Let us now in great detail look at the structures for each part of adaptive digital filter block diagram.

6.3.1. THE NTT DATA PATH

The algorithmic structure for an FNT of length 4 is shown in figure 6.6. Each step consists of the parallel propagation of N data points along the data path, followed by the parallel execution of N butterfly operations. Each node in the butterfly operation can be depicted as a multiply and add operation which is shown in figure 6.7. Since the root of unity (α) is a power of two, then the scaling stage in the butterfly operation would be reduced to only a word shift, hence reducing the hardware complexity. Let us now present a number of optimal or nearly optimal designs for an FNT network.

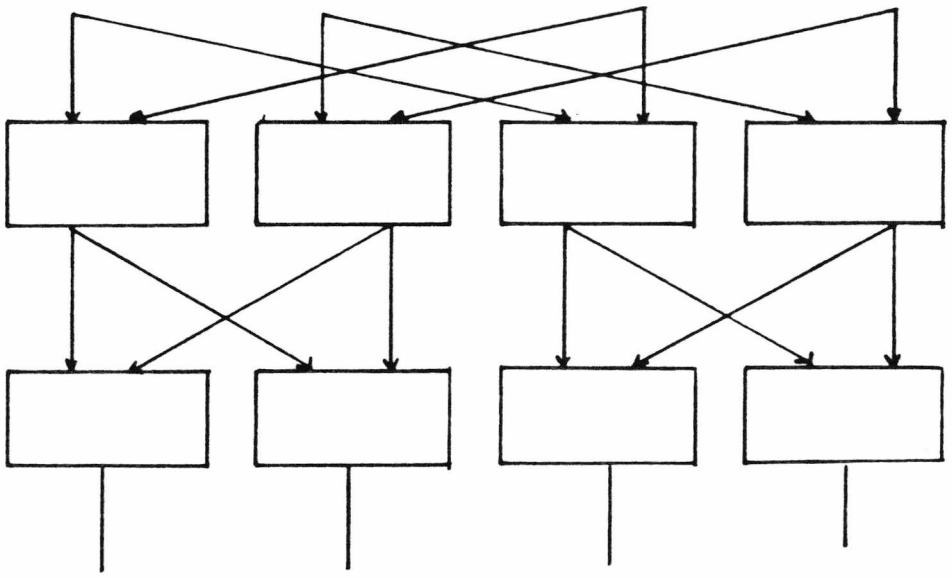


Figure 6.3 Decomposition of the FNT coefficients

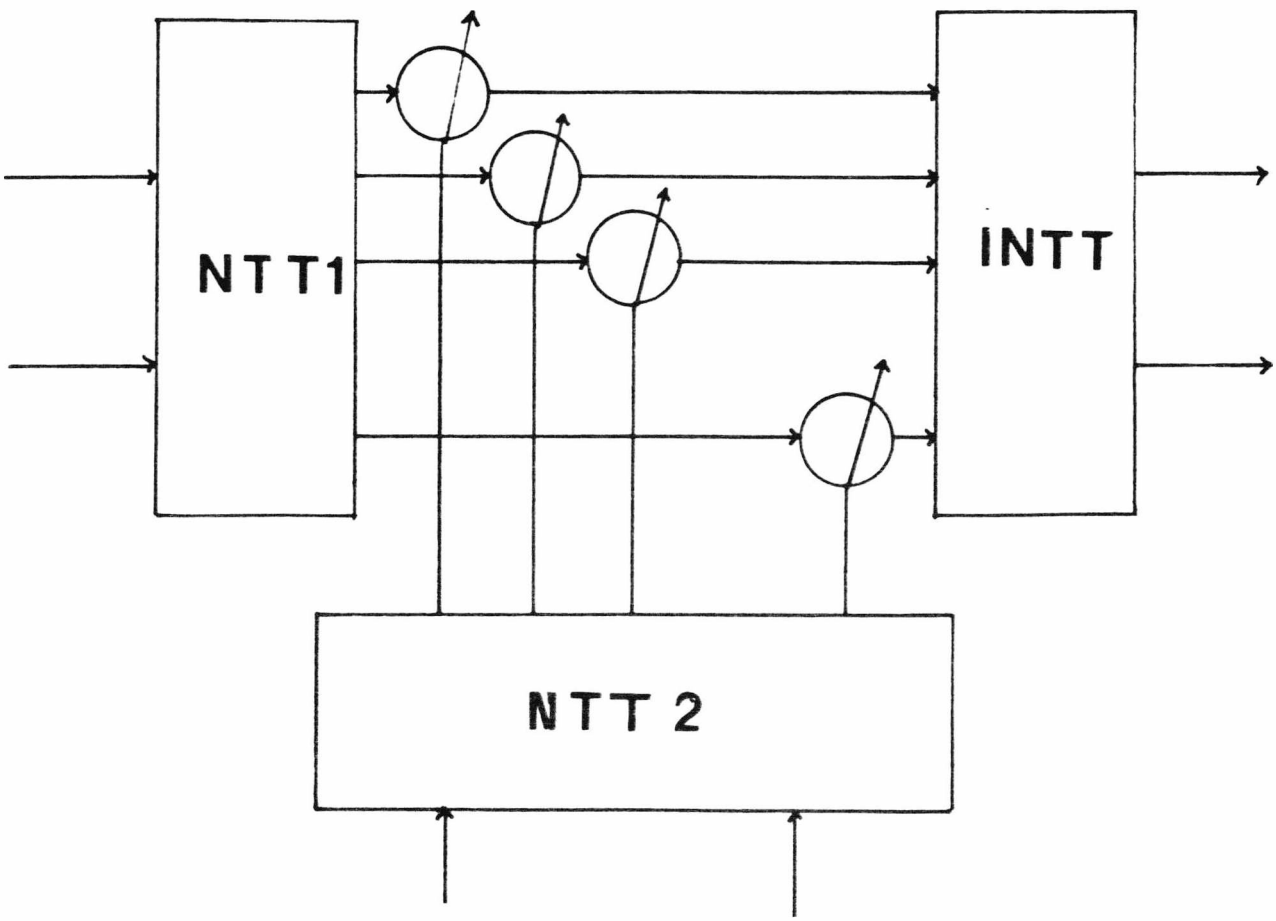


Figure 6.4 Frequency Adaptive Digital filter structure

6.3.1.1. THE DIRECT FNT ON A SERIAL PROCESSOR

A block diagram of a highly serial implementation of the FNT algorithm is shown in figure 6.8. It has three basic parts;

1) Memory unit

this is used to store the FNT coefficients namely α the root of unity, as well as the intermediate results obtained from the processing element.

2) Processing element

This does the finite field arithmetic calculations required by the butterfly stage. It has two units, a programmable shifter and an adder. The design of these two units will be delt with in next following sections.

3) The microprogrammed control unit

This controls both the processing element and the memory unit simultaneously, thus avoiding any synchronisation problems. This is a finite state machine and we can use a programmable logic array for providing the required control signal needed by the other units.

The operation of this circuit is as follows;

The appropriate data would be fetched from the memory and fed into the shifter for scaling it with the correct value of α , which can be fetched from the memory unit. In this case, it is the job of the controller to make sure that the correct address for retrieving the correct data from the memory is achieved. The second data would the be retrieved from the memory and would be fed into the adder. The adder would the compute the sum of the latest data and the output from the shifter and store it back in the memory. These operations would be carried out recursively until the required FNT coefficients have been achieved.

This highly serial implementation of the FNT would take a long time and hence it is slow. This is because it has far too little parallelism, and therefore it does not fully fulfill the VLSI design consideration which was set up earlier.

The designs in the next three sections employ progressively more parallelism to achieve better performance.

6.3.1.2. SYSTOLIC ARRAY FOR FNT

If one looks at the FNT decomposition algorithm of figure 6.3, any single butterfly stage can be viewed as a transfer from one register to another through a processing element. This is shown in figure 6.9. Here, we have a clocked input register, a processing element with no timing attached to it, and an output register clocked on the opposite phase. In this case the inputs are stored in the input register during the ϕ_1 , they are then propagated through the processing element with the result stored in the output register during the ϕ_2.

A sequence of FNT butterfly operations can be performed on a data stream by a series of such blocks separated by registers as shown in figure 6.10.

A 4-point FNT problem can be viewed as;

$$a_3x^3 + a_2x^2 + a_1x + a_0$$

where $a_3,a_2,\ldots$ is the input data and $x = 1,\alpha,\alpha^2,\ldots$. Evaluating the FNT coefficients at x distinct point gives;

$$y_0 = ((a_3 + a_2) + a_1) + a_0$$

$$y_1 = ((a_3 . \alpha + a_2) . \alpha + a_1) . \alpha + a_0$$

$$y_2 = ((a_3 . \alpha^2 + a_2) . \alpha^2 + a_1) . \alpha^2 + a_0$$

$$y_3 = ((a_3 . \alpha^3 + a_2) . \alpha^3 + a_1) . \alpha^3 + a_0$$

We see that all the y_i 's can be computed in a pipeline fashion by a single systolic array. It uses an array consisting of N-1 linearly connected basic cells as depicted in figure 6.11. The array is initialised by loading the input data to each cell. The output coefficients which is initialised to a_3 at left-most cell, together with the root of unity α are moving from left to right and gather their values and achieves their final result as

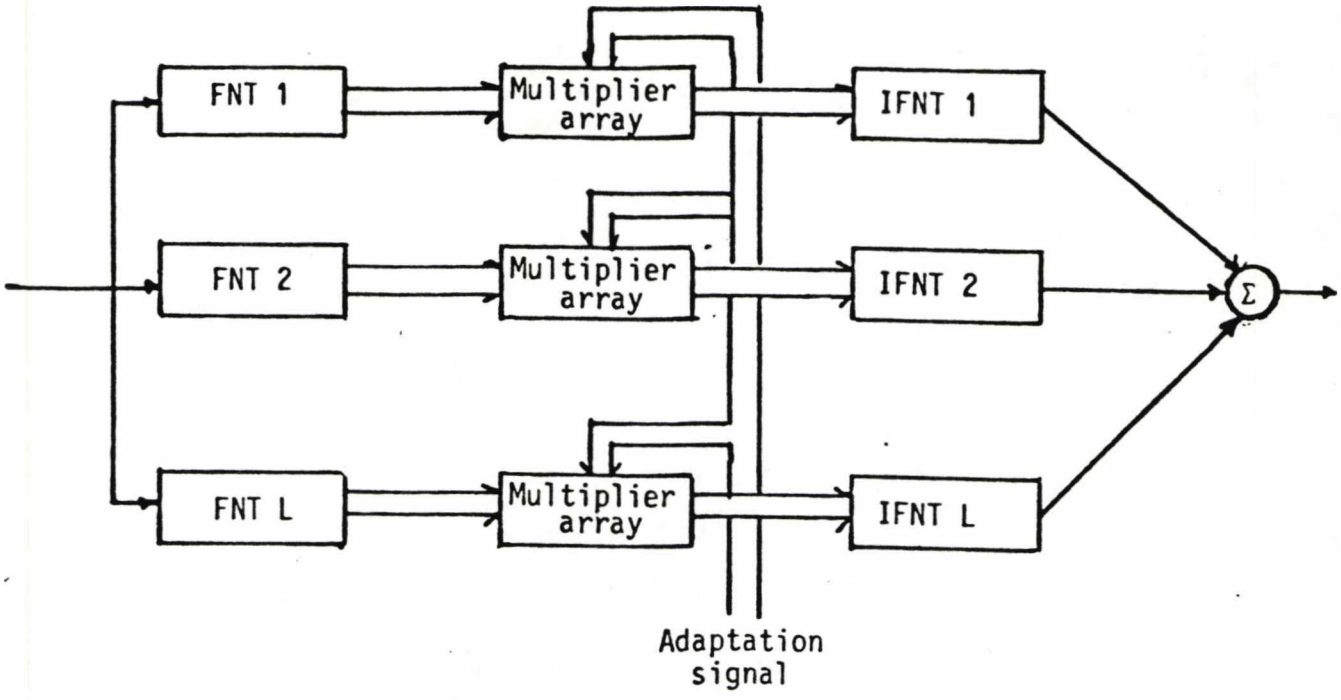


Figure 6.5
An Array of Adaptive Digital Filter Sub-Blocks

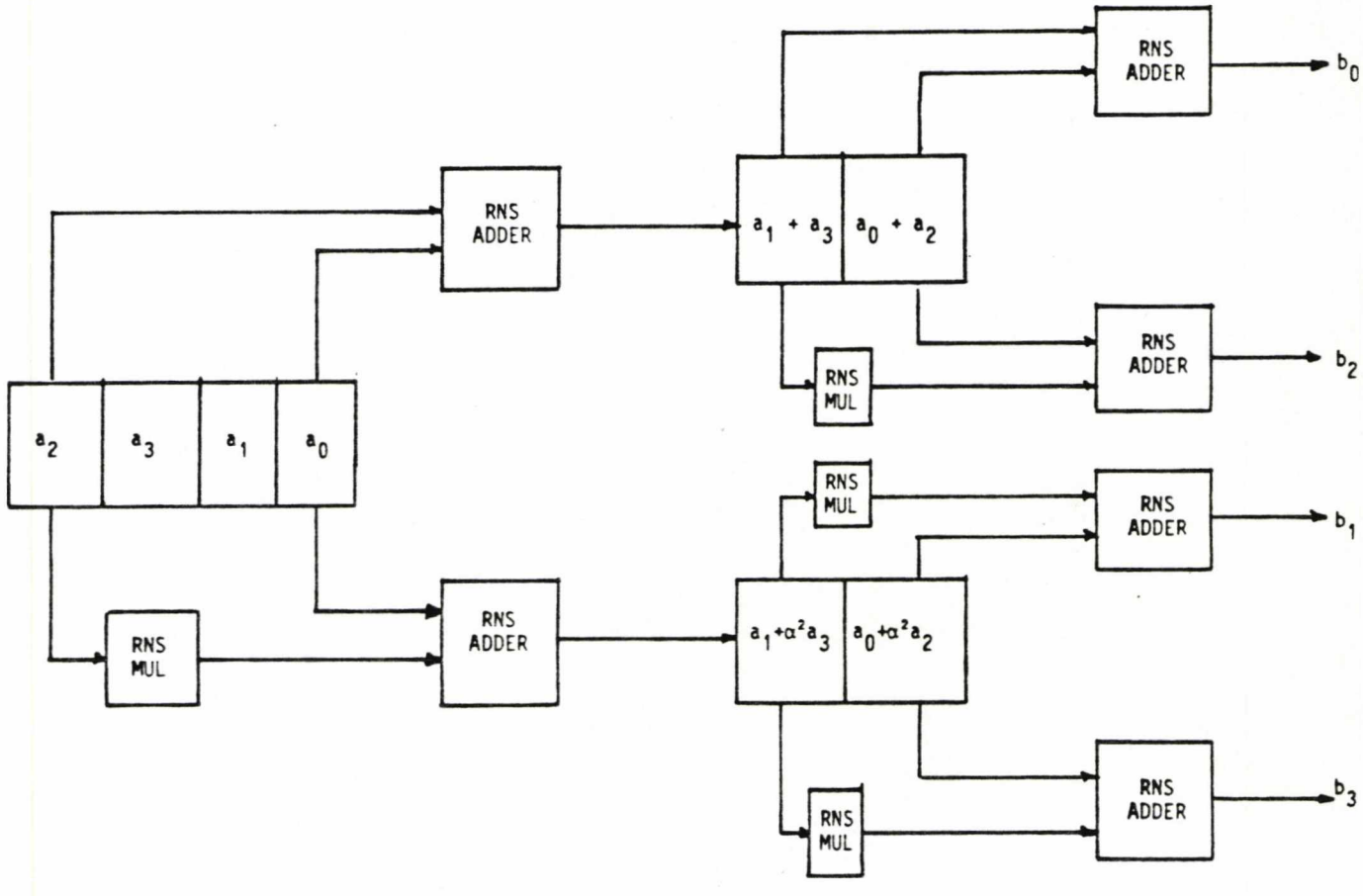


Figure 6.6 Algorithmic structure of a FNT of length 4

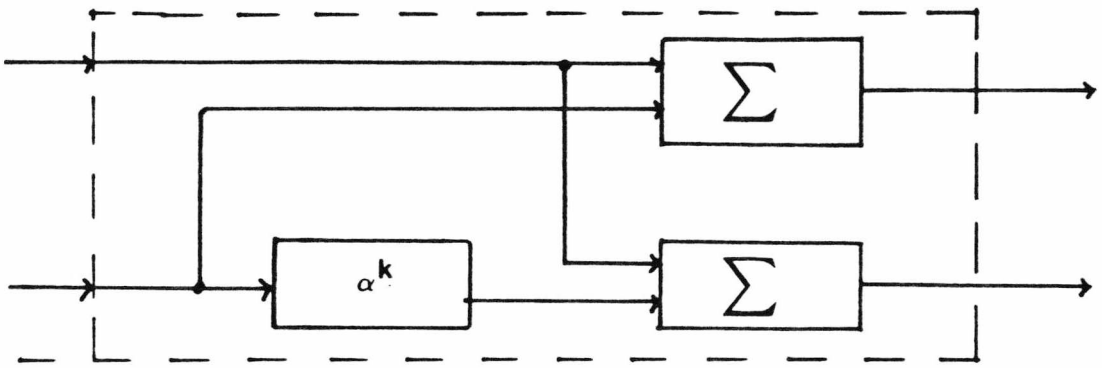


Figure 6.7 A node performing the butterfly operation

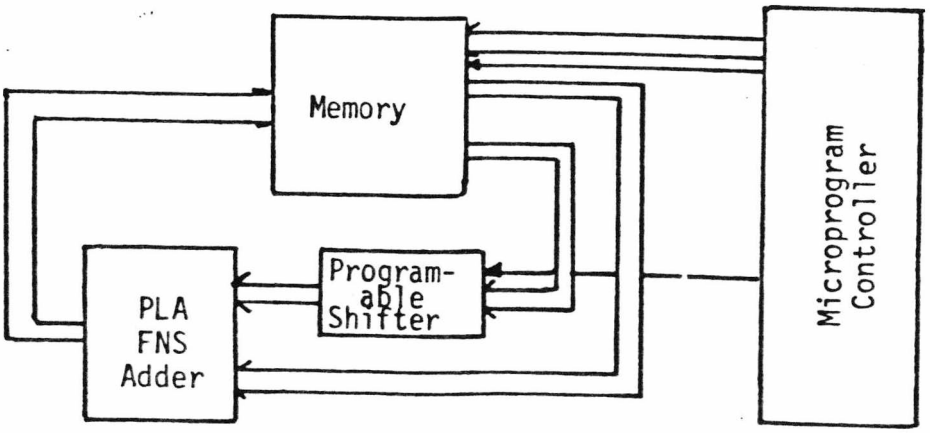


Figure 6.8
Serial Realisation for a Fermat Number Transform (and entire Adaptive Digital Filter)

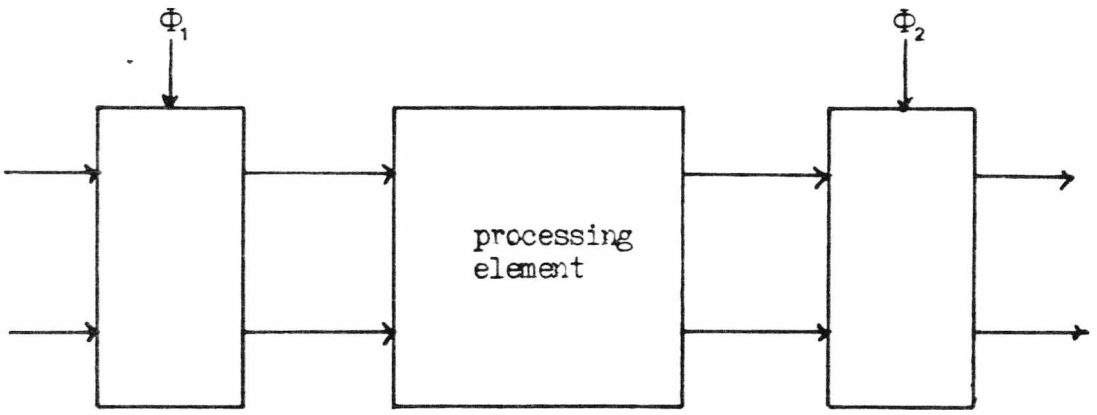


Figure 6.9 Butterfly operation block

leaving from the right-most cell. The number of cycles (clocks) which takes place in order to achieve all the FNT coefficients is shown in figure 6.12.

It can be seen that with this structure there is no need for complex control signal for the movement of data through each cell except a simple two-phase clock signal ϕ_1 and ϕ_2 . The communication between each cell is simple and local therefore minimises the time complexity since long wires is formidable in the VLSI design. The only disadvantage of this structure is that, as the number of cells increases the feedback wire for the recycle of the root of unity will increase. In order to overcome this limitation, one possibility would be folding the array.

6.3.1.3. PARALLEL AND SYSTOLIC STRUCTURE FOR AN FNT

In the previous section we presented a systolic FNT array that uses N-1 processing elements (cells) in order to compute the FNT coefficient of a given sequence. It takes 2N-2 cycles to obtain all the results. However, even though it is better than the sequential structure, the FNT coefficients are still obtained in series. It is therefore possible to boost the speed of the computation of the FNT coefficients by using a combination of parallel and systolic structure. This is shown in figure 6.13.

Unlike the systolic array structure where the constant FNT coefficients namely the root of unity which have to travel from one cell to another, the parallel-systolic array does not use this method. Instead the constant coefficients are stored in the processing cell and they will stay there throughout the processing procedure and the data input will move from left to right accumulating the results as it moves through the array. In this case the array will uses 2N-2 cells and it takes N-1 cycle to complete and give an output of final results.

Each processing element in the array is comprised of a single multiply-add cell plus a register which hold a particular value of α. Note that communication between each cell in the array is local where, each cell is only connected to the cell in the row following it.

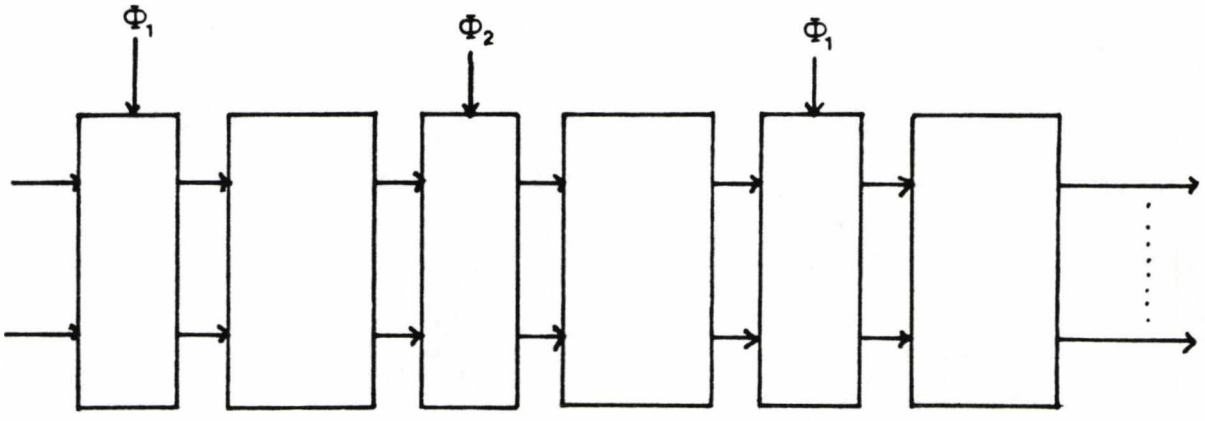
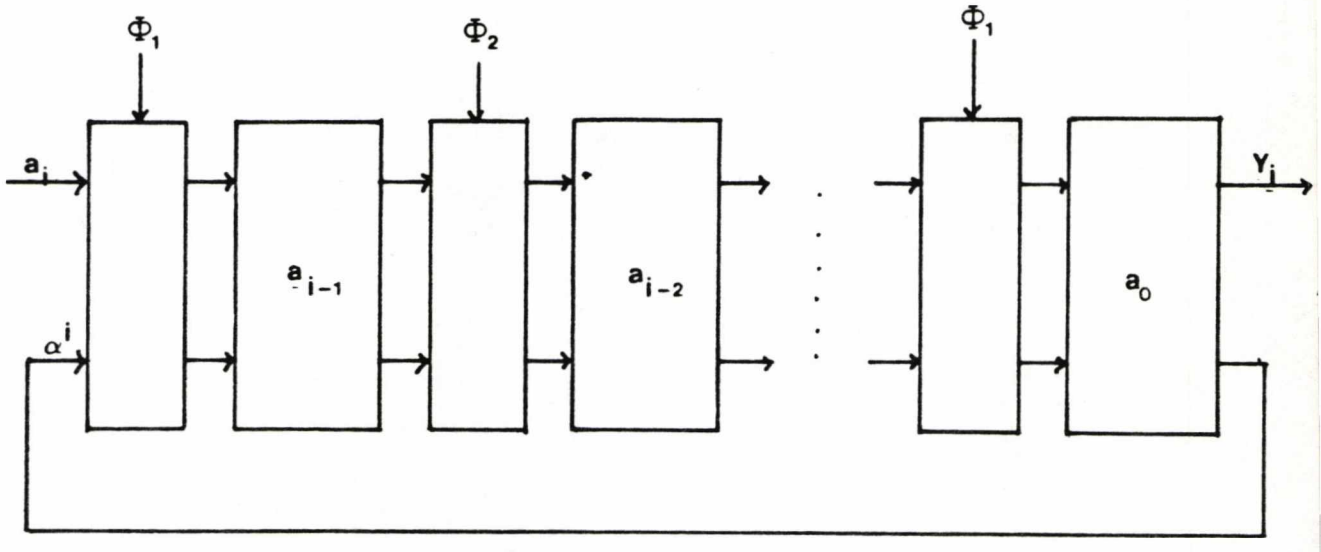
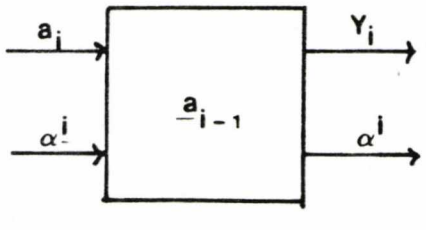


Figure 6.10 Pipeline structure for FNT





$\alpha^i \longrightarrow \alpha^i$

$Y_i \longrightarrow a_i \alpha^i + a_{i-1}$

Figure 6.11

a) state of FNT systolic array after loading

b) basic FNT systolic array cell

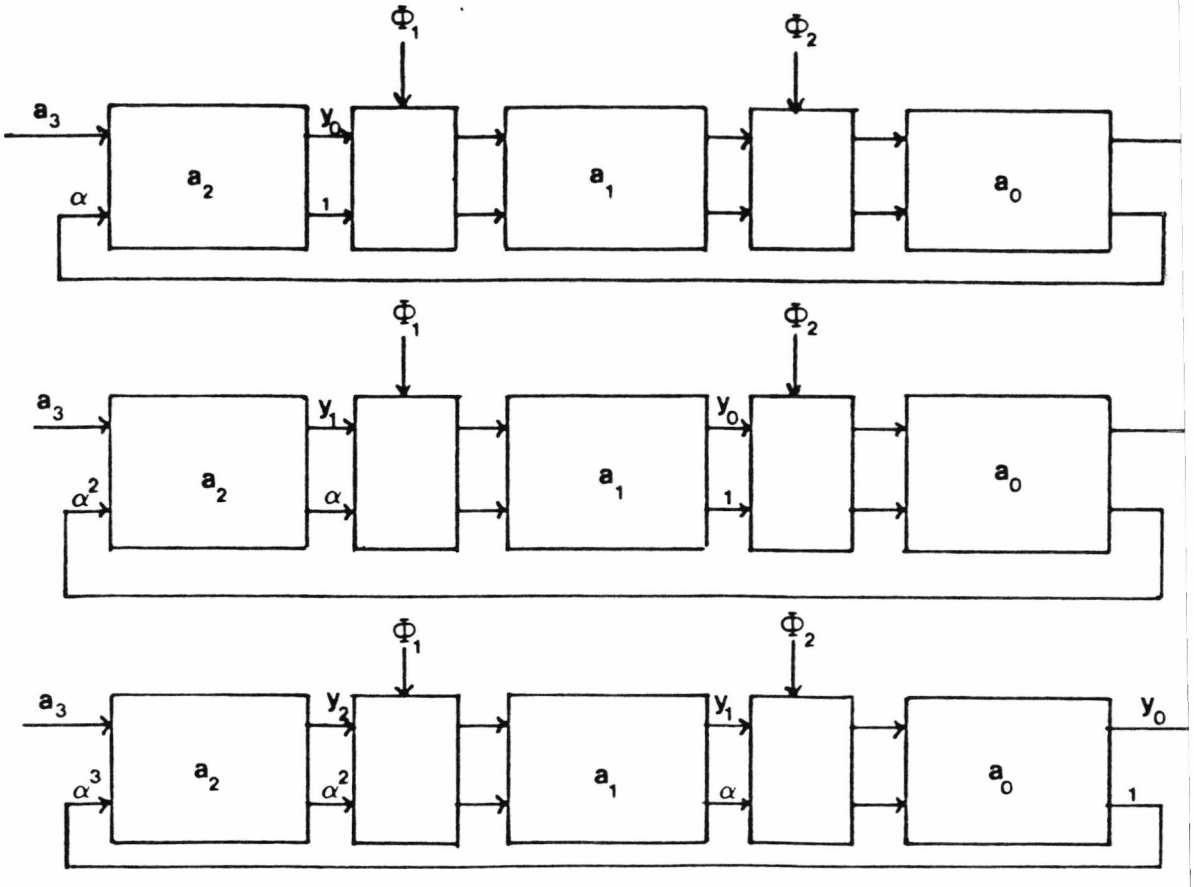


Figure 6.12 Number of cycles of the FNT systolic array

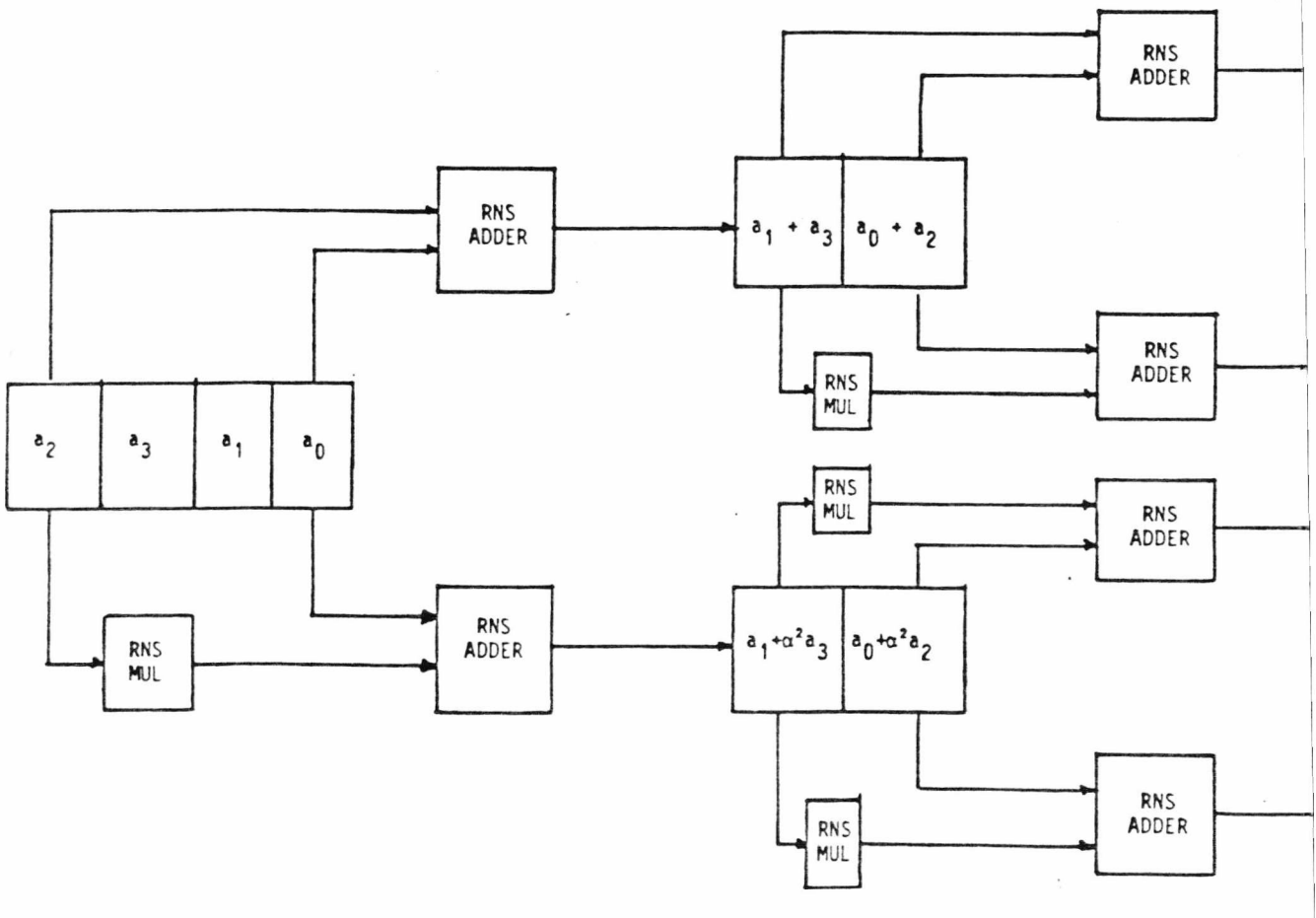


Figure 6.13 Parallel-systolic structure for FNT

This structure has advantages over the systolic structure namely, the FNT coefficients are available at the same time so it improves the speed of the filtering and adaptation procedure and more importantly the long feedback wire which is used for the recirculation of the α has been omitted at the expense of some extra hardware.

6.3.1.4. PARALLEL STRUCTURE FOR AN FNT

One of the most obvious ways of designing the FNT is to provide one processing element for each butterfly execution. This will lead to a highly parallel structure which is depicted in figure 6.14. The data movement between the processing elements are the same as the parallel-systolic architecture, where the α will stay and the data moves from one level (column) to the next and so on. The interconnection between the cell in one column to its neighbouring cell in the other column can be obtained from the following:

The cell i in the first column is connected to two cells in the second column, cell i and cell i+N/4. Cell i in the second column is connected to two cells in the third column and so on. It can be seen from the diagram that N vertical tracks are necessary and sufficient for the interconnection between the first two column, while the connection between the second and the third columns need only N/2 vertical tracks. In fact the interconnection between the first two columns can be cumbersome when it has to be implemented on a chip.

In order to complete the FNT algorithm, the structure shown in figure 6.14 will uses NlogN processing elements in logN cycle of time.

SUMMARY

In this section we presented four FNT structures. The later case of the four designs is optimal in terms of speed and processing power. The problem with the non-optimal design namely the serial processor is that, it is processor poor, that is the number of multiply-add cells does not grow with the problem size.

The problem with the pipeline structure is that even though the processing power

is increased, the results are still obtainable in the serial manner. But the advantage of this structure is that the interconnection between each cell is simple, that is a single track is used for the movement of data.

The problem of the parallel-systolic structure is getting the data into its correct place in the registers. That requires an extra hardware for the distribution of the data into its appropriate register cells. However, the advantage of this structure is that it is faster than the systolic array and the coefficients are available at the same time.

The optimum of all these structures is the parallel design. The data flow is easy and unlike the parallel-systolic structure, it does not need any distributor for loading the input data to the registers. But one limitation is the interconnections between the processing elements in the first row. The width of the tracks which will transfer data to each processing element depends upon the number of bits that each data is represents. On the other hand it is depends on the modulo that been chosen. The larger the modulo would be, the wider the tracks becomes, and since each track crosses over each other it could be quite costly in terms of the area on the chip. However, the assumption which we make here is if one track needs to cross over another one, that can be done by moving up to another level in the layout process. This in fact would not cause too much inconvenience in the fabrication process.

The architecture for the inverse Fermat transform is identical to that of the FNT, differing only in the pre-scaling of each input data.

6.3.2. FILTER OUTPUT AND ADAPTATION STRUCTURE

In this section we present the design of filter output and adaptation procedure From figure 6.4, it can be seen that each filter output procedure can be thought of as a first order filter, that is each FNT coefficient is first scaled with the appropriate filter coefficient and once the output is obtained the adaptation procedure which was explained in earlier chapter, can be proceeded. Therefore each stage of filtering and adaptation can be obtained independent of the other stages. Since all the stages are similar we shall present only a structure for one stage. A block diagram of the

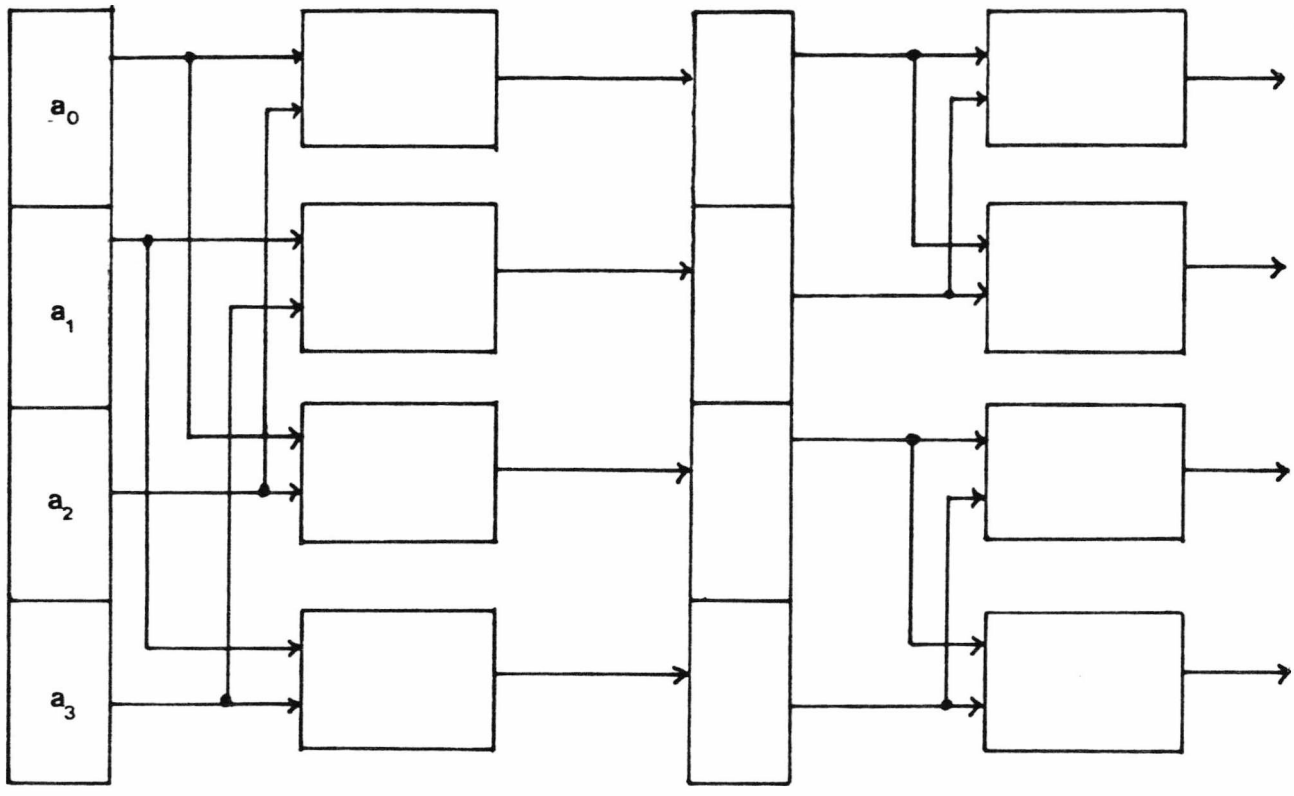


Figure 6.14 Highly parallel structure for FNT

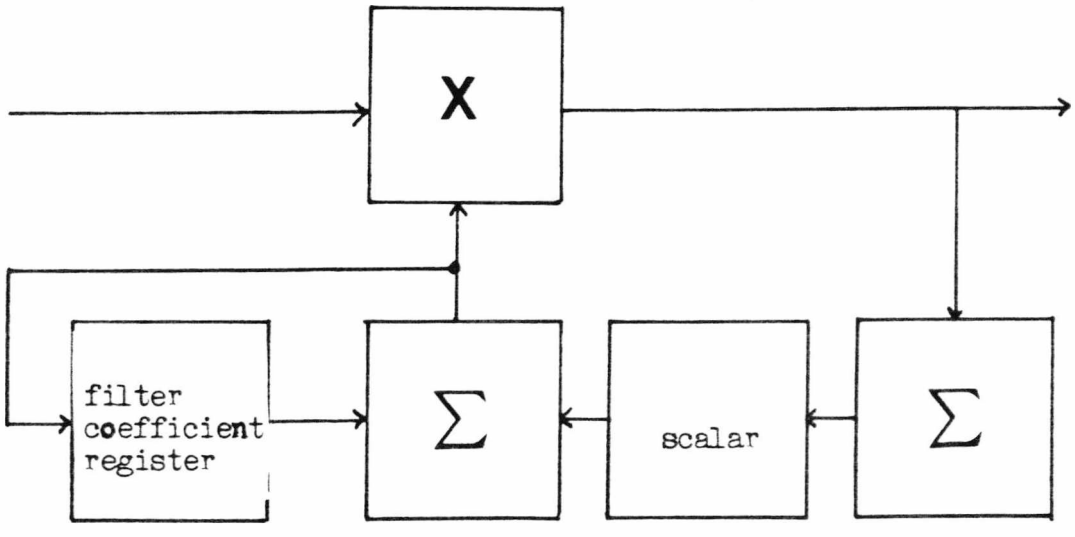


Figure 6.15 Adaptation procedure structure

filtering and adaptation structure is shown in figure 6.15.

It consists of two scalar (multiplier), two adder and a filter coefficient register unit. The register is first loaded with the initial value of the pre-calculated filter coefficient. The adaptation will take place in two stages;

Step1) Once the FNT coefficient is obtained from the transform stage the filter output can be obtained by applying both the FNT and filter coefficients to the scaler.

Step2) As soon as the filter output is obtained, section "A" can be activated which is the adaptation procedure for updating the filter coefficient. In this case the filter output will be compared with the desired response data in order to obtain the error. After the scaling the error output appropriately, it can then be added to the previous value of the filter coefficient which can be discovered from the register. The output of the adder which is the updated value of filter coefficient can be fed into the scaler unit for the calculation of the next filter output as well as to the register for the next adaptation process.

However, there is an important feature about this structure and that of the FNT, that is the problem of synchronisation. It is appropriate to point out here that with structures explained so far for the FNT processor, its output coefficients are available at every clock cycle. This is due to the fact that every stage in the procedure is pipelined, that is a new set of data can enter the network as soon as the previous set of data has left the first column of the processing elements. Hence, it is important from the synchronisation point of view for the filtering stage and adaptation to be done within one clock cycle. However, one has to be careful that the adaptation process should not take place until the filter output is available.

The system synchronisation can be achieved by placing a number of latches in appropriate places. For example, let us assume that the delay across the multiplier in the figure 6.15 for the calculation of filter output is of the order of T, then one possibilities would be to start the transformation of the desired response (NTT2 figure 6.4) T units of time after the transformation of the input data (NTT1, figure 6.4) has

started.

6.4. THE PROCESSING ELEMENT ARCHITECTURE

In the previous section we presented a number of structures for the FNT stage of an adaptive filter for the implementation using VLSI technology. The structures consist of two blocks, a) shifter register and b) processing element.

In this section we shall look at the functional blocks of the processing element in greater detail. The main functional blocks of a processing element are a multiplier and an adder as shown in figure 6.16. In computing the FNT, arithmetic is done modulo $M = 2^b + 1$. In this case the only allowed integers are 0,1,2,.....,M-1. Using a b-bit register all integers from 0 to M-1 can be represented, hence the arithmetic operations such as multiplication and addition are done on two sets of data. However, a problem arises when we want to represent M. To overcome this limitation it is possible to use an extra bit to represent M which means extra complex hardware. However, we will show later the approach we have taken to design the requirements for the FNT calculation which will almost eliminate this complexity. Or, in case M does occur we can round it to -2 or zero. Let us now discuss how various hardware requirements such as a multiplier and an adder can be implemented.

The implementation of hardware requirements for the residue numbering system can be accomplished by either designing a special logic network (mostly random logic) or by a table lookup method.

6.4.1. SPECIAL RANDOM LOGIC IMPLEMENTATION

When computing a residue arithmetic operation, attention has to be given to the output result of any stage. That is, it is possible that the result from an operation might lie outside the field of operation, which is the case in many operations such as multiplications. Therefore one has to be capable of converting the result from outside of the field back within the field we are operating. With this in mind let us look at the design of an adder and a multiplier.

THE ADDER

When adding two b-bit integers, we obtain a b-bit sum and possibly a carry bit. If however the carry bit is set, it can be concluded that the result is outside the field of operation and it has to be converted in order to represent it in a b-bit register for further calculation. One possibility would be to subtract the carry from the sum. Therefore, the hardware should be of the carry-subtract type. This is shown in figure 6.17.

Another approach is to have a programmable adder that is capable of selecting whether a subtract operation is needed or not. This is shown in figure 6.18.

THE MULTIPLIER

When multipling two b-bit integers, we need a 2b-bits register to show the product result. However, since one of the data inputs is a constant α and a power of two, then the multiplication is reduced to only word shift (except in the filtering and adaptation stage). Suppose we need to multiply the content of a register by $\alpha = 2^k$ where $0<k<b$, all we need to do is left-shift the contents of the register by k bits and subtract the k overflow bits. A block diagram of a multiplier is shown in figure 6.19. It consist of a shifter which is capable of shifting the data to the left by the amount of required places, then the output of the shifter is divided into a low and high order part and they will be subtracted using the adder technique explained above.

Computation of the inverse transform requires a multiplication by negative power of two. In this case we need to shift the data to the right by the required number of places and the rest of the operation is the same as in the transform stage.

6.4.2. THE TABLE LOOKUP IMPLEMENTATION

The random-logic approach for the implementation of the adder and multiplier of previous section has two main disadvantages if one has to think about VLSI implementation of these cells;

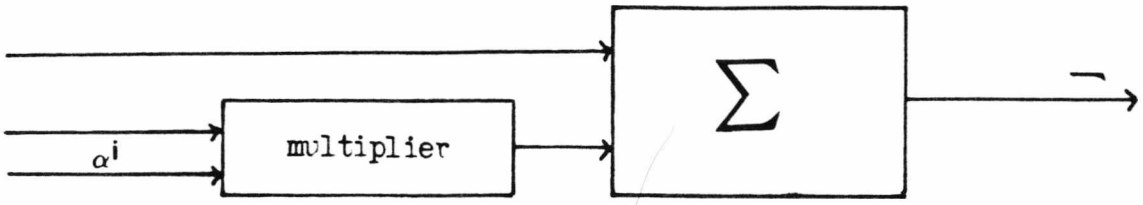


Figure 6.16 Functional description of processing element

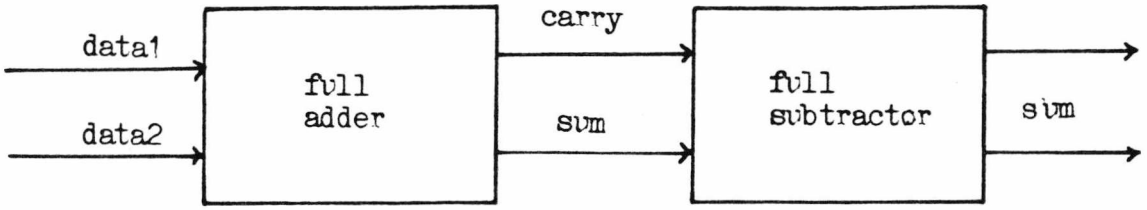


Figure 6.17 A carray-subtract type adder

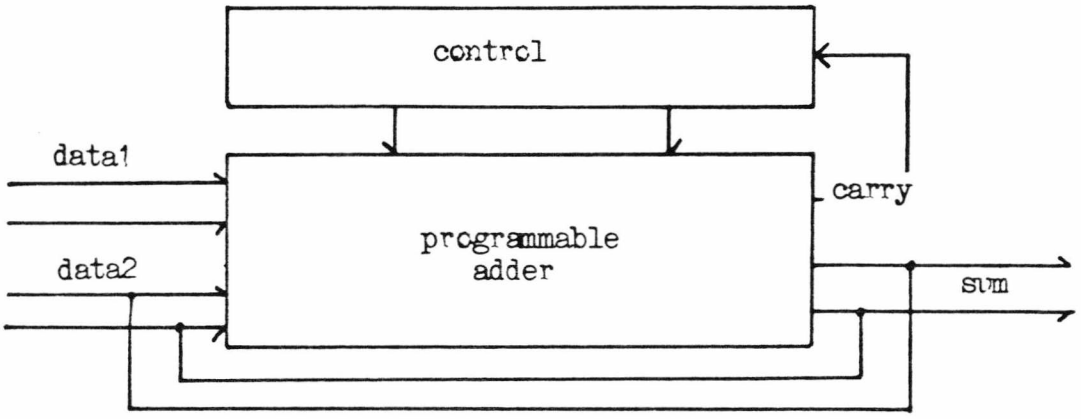


Figure 6.18 Block diagram of the programmable adder

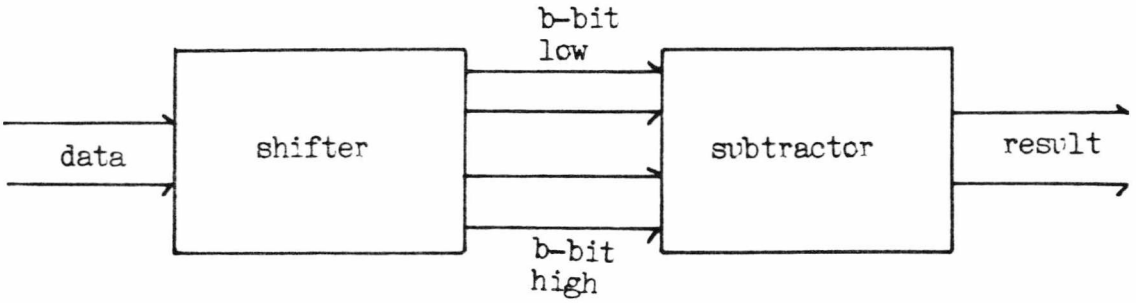


Figure 6.19 Block diagram of the multiplier

1) Since the layout process of the random logic for each cell on a silicon could be cumbersome and more importantly the design and fabrication cost is too high and since they are not regular it is not suitable for the implementation utilising VLSI technology.

2) There is a need for extra cell for the situation that needs rounding. For example, in the adder design an extra cell (subtractor) is required when the carry bit is set. Since the speed is an important factor, and more importantly the data path is pre-defined, the result of the adder is fed into the subtractor regardless of whether the carry bit is set.

In order to over come these limitations and more importantly for the high speed realisation of such blocks, the lookup table approach offers better a solution.

To see how the arithmetic operation modulo M can be implemented using the table lookup approach, it is important to point out some of the properties of the modulo M. Since M is a cyclic group it can have only M distinct value and it will periodically repeat itself after M-1 steps, hence one can recognise this as a finite state machine, with the number of states depending on the value of M. In this case any arithmetic operation modulo M can be thought of as a finite state machine too, hence it is possible to store these states and using the data to address specific state. In general, every discrete function of D operands (inputs) can be represented by a mapping table, and thus after encoding, by a binary-encoded truth-table it provide outputs for all possible 2^D input combinations.

In general, the RNS arithmetic operations namely multiplication and addition truth-tables may be implemented in storage unit by two distinct addressing techniques. The first is by using Location-Addressable Memories (LAM) such as Read-Only or Random-Access Memories (ROM,RAM). The second technique uses Content-Addressable Memories (CAM) such as associative memories or various Programmable Logic Arraies (PLA, PAL, etc). The basic advantage of the Content-Addressable Memories over Location-Addressable memories is that the later require storage of the

entire function truth table where as the Content-Addressable Memories require only the minterm portion of the table. Thus, by using the latter approach, a substantial saving in silicon area may be achieved.

The basic idea of the table lookup implementation by Content-Addressable Memory is illustrated in figure 6.20. In this structure, all input bits are simultaneously applied to each CAM module, while each output bits correspond to a distinct module. The operation is based on the matching property of Content-Addressable Memory, that is, the processing consists of comparing input patterns to prestored information in the CAM module, with the appropriate output bit signifying detected matchings.

Among many array logic configurations we use the Programmable Logic array of figure 6.21. The overall arrangement of a PLA is shown in figure 6.21. It consists of two switching matrices in cascade performing the AND and OR functions, two sets of buffers, an interarray drivers, and precharge section. The operation of the PLA is as follows;

The inputs are run vertically through a matrix of circuit elements called AND-plane. The AND-plane generates specific logic combinations of the inputs and their complements. The outputs of AND-plane run horizontally through another matrix called OR-plane. The outputs of the OR-plane will run vertically and can be obtained from the output buffers. In order to clarify the operation of a PLA let us look at an example;

The circuit diagram of a specific PLA is shown in figure 6.22. The input is applied to inverting and noninverting buffers. The buffers drive two lines run vertically through the AND-plane, one for input term and one for its complement. The outputs of the AND-plane are formed by horizontal lines with pull-up transistors at their left most end. The function of the PLA's AND-plane is then determined (from a truth table) by locations and gate connections of pull-down transistors connecting the horizontal lines to the ground. Each output running horizontally from the AND-plane carries the NOR combination of all inputs that lead to the gates of transistors attached

to it. The OR-plane circuit elements is identical in form to the AND-plane.

We shall now detail the truth table implementation scheme of residue based functions in PLA logic. We shall assume that a multioperand residue-based function is given in tabular form. This is illustrated in figure 6.23(a) for two operand addition modulo 5. Then, by encoding the function residues, we generate a binary-encoded truth table as shown in figure 6.23(b). Note that the truth table consists of an input and output parts. The input and output table bit entries correspond to the AND-plane and OR-plane fusable link of a PLA respectively. Clearly each column of the output table (figure 6.23b) corresponds to a PLA output. The rows or minterms of the input table are simply the PLA word patterns to be embedded as product terms (P-term) and to be matched by appropriate input combinations.

Similar approachs can be taken for the implementation of the shifter and multiplier. However, since the number of shifts at each stage of the FNT structures is known, it is possible to combine the shifter and the adder so that in all cases the processing element is simply a PLA.

As we shall see in the next section, the VLSI system performance is directly proportional to the time-space complexity of each processing element. Since processing elements are identical, that is they are only a single PLA, we shall concentrate only on the area-time complexity of the PLA. To formulate our frame work, we define the time-space complexity in terms of total number of truth table minterms required in order to implement a specific function. A direct implementation of the truth-table minterms using a PLA could be very costly in terms of the area and time. However, it is possible to reduce the number of P-terms in the AND and OR planes with the aid of a minimisation procedure. A PLA layout for a modulo 5 and 17 adder is shown in figure 6.24 after using a minimisation program [119]. In case of modulo 5 adder the number of P-terms is reduced from 25 to 18, where as for the modulo 17 adder the P-terms reduced from 268 to 71 which is a great saving both in time and area.

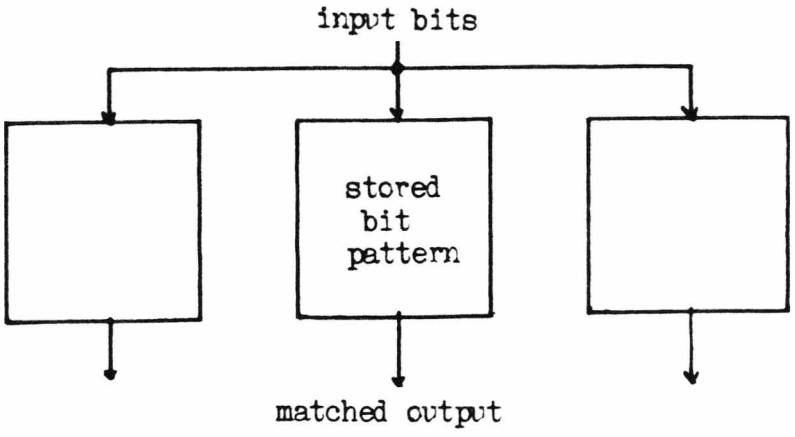


Figure 6.20 Content addressable memory module orgnisation

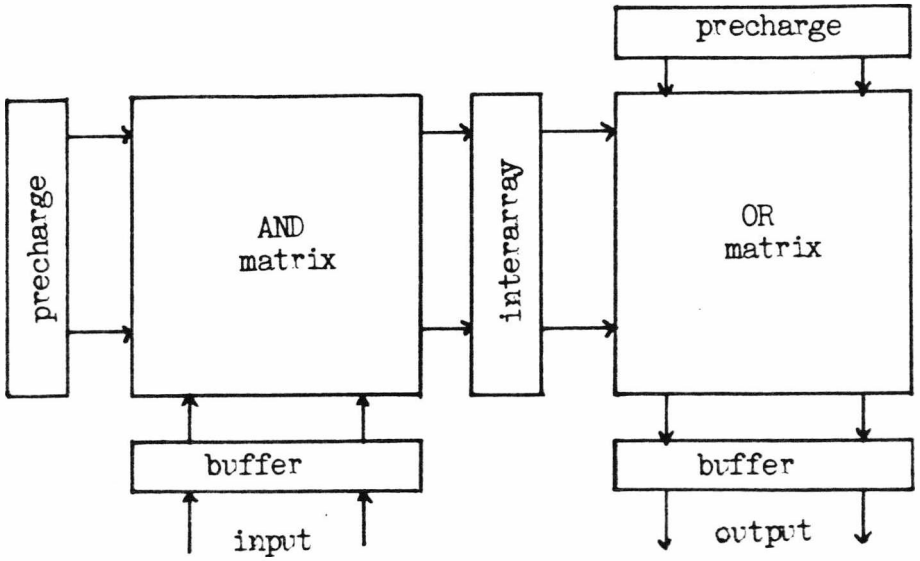


Figure 6.21 General PLA configration

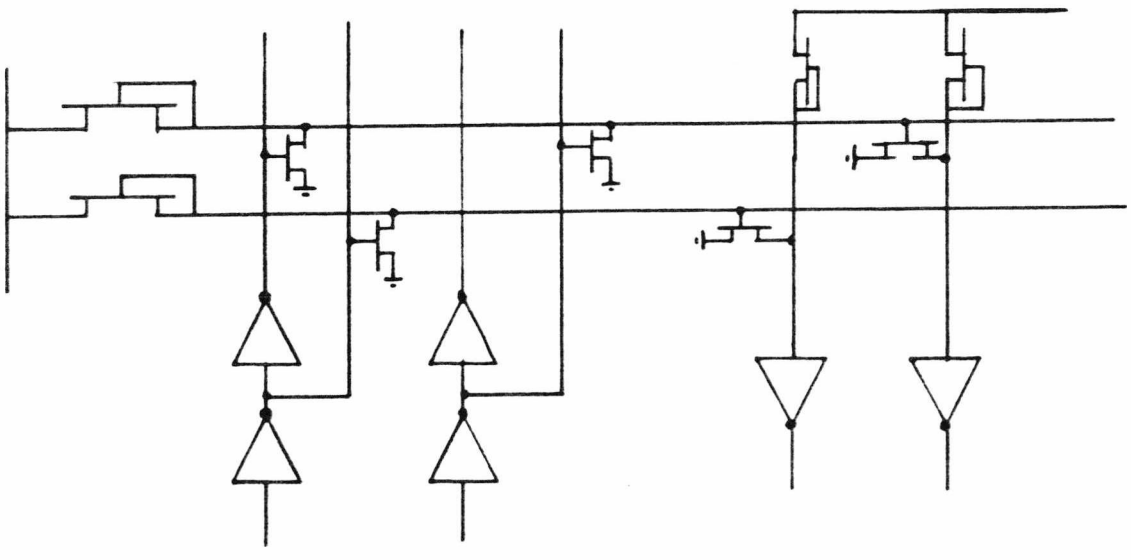


Figure 6.22 Circuit diagram of a PLA

INPUTS		OUTPUT
1	2	
0	0	0
0	1	1
0	2	2
0	3	3
0	4	4
1	0	1
1	1	2
1	2	3
1	3	4
1	4	0
2	0	2
2	1	3
2	2	4
2	3	0
2	4	1
3	0	3
3	1	4
3	2	0
3	3	1
3	4	2
4	0	4
4	1	0
4	2	1
4	3	2
4	4	3

a

INPUTS		OUTPUT
1	2	
000	000	000
000	001	001
000	010	010
000	011	011
000	100	100
001	000	001
001	001	010
001	010	011
001	011	100
001	100	000
010	000	010
010	001	011
010	010	100
010	011	000
010	100	001
011	000	011
011	001	100
011	010	000
011	011	001
011	100	010
100	000	100
100	001	000
100	010	001
100	011	010
100	100	011

b

Figure 6.23 Two-operand addition tables modulo 5

6.5. SYSTEM PERFORMANCE

The measure of VLSI device performance often involves both speed and device area. Given that the architecture of figure 6.14 can be realised on a chip, or partially on a chip, we are also interested in its overall data throughput.

In most structures explained so far, we assume that the register array are driven by a biphase clock. The minimum clock period is determined by the sum of the register and PLA delay. Taking account of the layout of a PLA of figure 6.24, an estimate can be derived for the worst case PLA delay which is dependent upon the number of product terms on each of the AND and OR plane and upon the possible outputs:

PLA delay = < input buffer > + < AND > + < OR > + < output buffer >

Where < X > mean the delay across X.

Using the design rules of Mead and Conway, the following formula for the PLA delay can be derived;

worst case

$$PLA\ delay = \begin{bmatrix} (1+K)t \\ or \\ Kt \end{bmatrix} + [(q_i \cdot Kt) + Kt] + \delta \cdot Kt + Kt$$

where;

K is the aspect ratio

t is transit time

q_i is the highest number of ones or zeros on a specific line

δ is the number of outputs being true simultaneously

Using typical parameter values for K and t of 9 and 0.3ns respectively, a total worst case delay for a PLA adder using two 4-bits input and output data and arithmetic done modulo 17 is approximately 90ns. Taking into account also the delay through the registers, and the fact that the clock is biphase, a reasonable estimate for

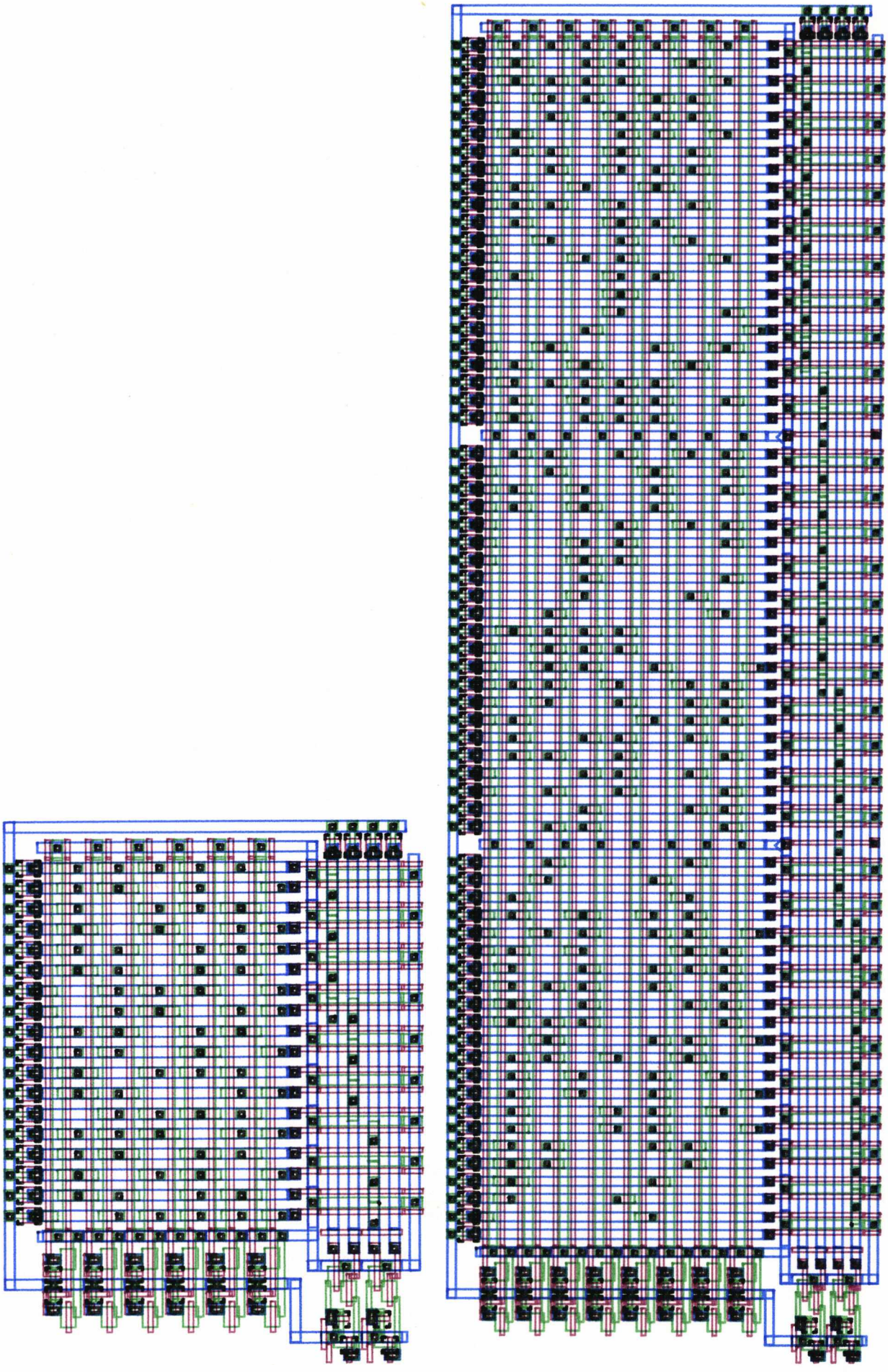


Figure 6.24

the cycle time for the device is 200ns.

Let us now determine the minimum sample period for each of the structures explained so far. Since the architecture of figure 6.14 is designed to be used in a pipeline mode, eight data input samples can be processed every cycle, resulting in a minimum data sample of 25ns, and the data delay through the entire data path is between 700-800ns. As in the systolic architecture of figure 6.12, the eight data input samples can be processed every N-1 cycles, resulting in a minimum data sample of 90ns, and data path delay of 1.7 μ s. A similar analysis can be performed for the serial architecture of figure 6.5. A summary of performance estimates is given in the table below:

architecture	transform	minimum sample	data path
	lenght	period in ns	delay in μ s
parallel	8	25	0.7-0.8
systolic	8	90	1.4
serial	8	300	2.5
serial	1024	4	4

Another factor which will influence the complexity of VLSI and hence the system performance is the area, that is the area which each processing elements will take on a silicon. Hence, it is possible to estimate the required area and thus work out the number of processing elements which can be intergrated on a chip.

The area for which we are trying to estimate is that of a PLA adder shown in figure 6.24b, and depend upon the number of inputs, outputs, and product terms:

PLA area = PLA high x PLA width

yielding to;

$$PLAarea = [\,(2W_p + W_d) + 8\lambda\,]\,P + L_{OR} + P\,(W_m + \pi) + L_b$$

$$x\,[\,(2W_p + W_d) + 8\lambda\,]\,n + L_{AND} + M\,(W_m + \pi)$$

where

W_p = width of poly-silicon

W_d = width of diffusion

W_m = width of metal

N = number of inputs

M = number of outputs

P = number of product terms

L_{OR} L_{AND} = the length of pull-up transistors in the AND & OR plane

L_b = length of the buffers

Using typical parameter values for W_p , W_d , W_m , L_{OR} , L_{AND} , L_b of 2,4,4,15,53 lambda respectively, a total area for the PLA adder is 749x193 lambda square. Let us estimate the required area for a transform length of 4, shown in figure 6.14. One possible layout including the interconnection between each processing cell (figure 6.14) is shown in figure 6.25. The total area is given by:-

TOTAL AREA = H x W

where H is the total height and W is the total width.

The total width W is given by:-

W = width of PLA + width between each PLA

= 4 x 204 Lambda + 3 x 3 Lambda

= 915 Lambda

The total height H including the interconnection between each PLA is given by:-

H = height of PLA + width of interconnection lines

Assuming using polysilicon for the interconnection lines (1-8), the area for a line (4 bits wide) is equal to 14 Lambda. Hence, the total area required for the interconnection lines is 112 Lambda. Therefore, the total height is:-

H = 2 x 740 Lambda + 112 Lambda

= 1592 Lambda

If Lambda = 3 micron then it is possible to integrate between 8 to 16 PLA (not including the input/output pads) on a single chip of a size of 6000 micron by 6000 micron.

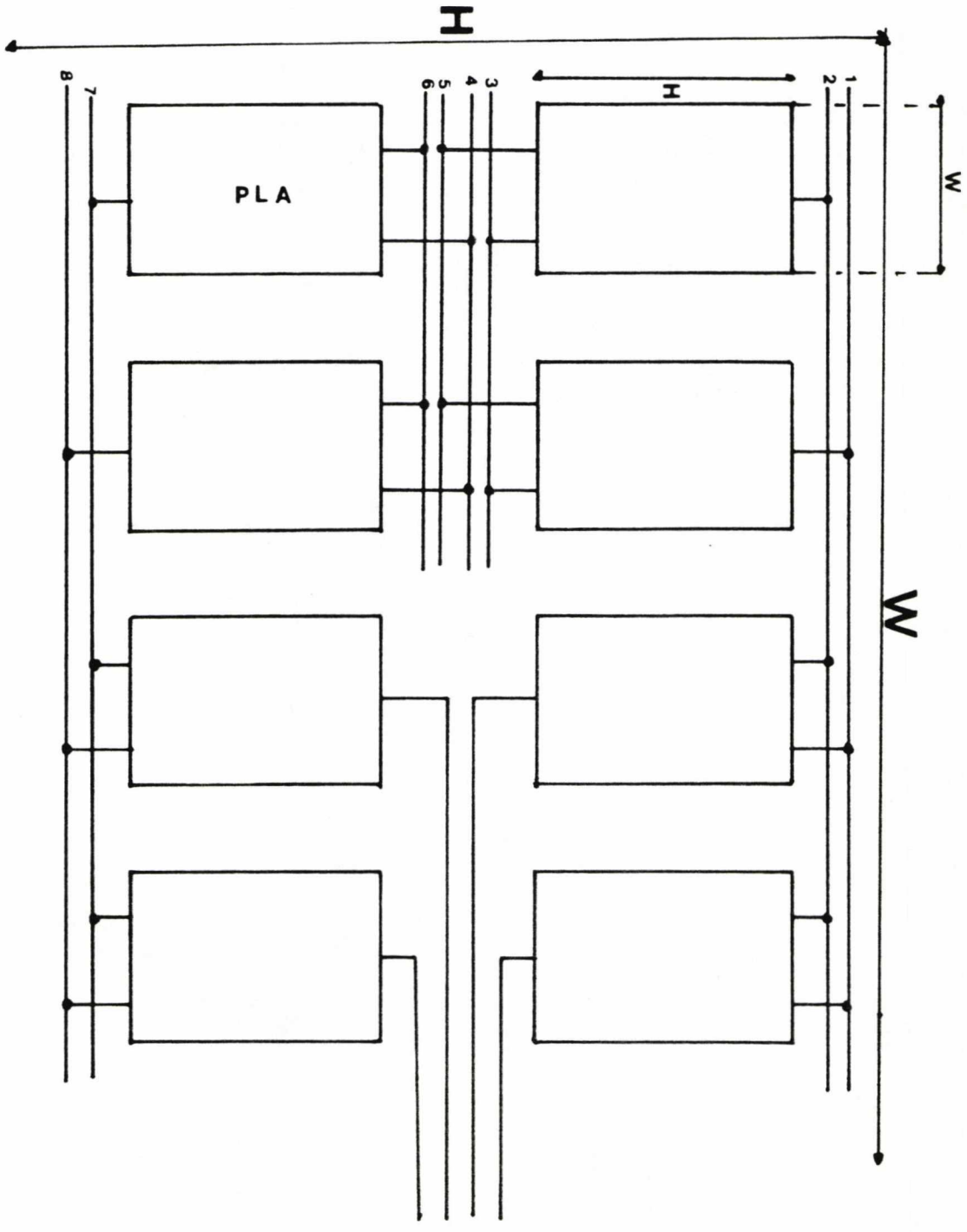


Figure 6.25 A possible layout for an FNT of length 4

CHAPTER 7

CONCLUSION

7. CONCLUSION

This project began with the study of how signal processing algorithms and in particular digital adaptive filters can be implemented in VLSI utilising the NTT. It then covered the design of a special-purpose processor for an adaptive digital filter, based on the concept that the major operations performed are of the form AxB+c i.e. sum of product.

7.1. ADAPTIVE FILTER AND THE NTT

Adaptive filters perform two major operations: a) convolution of two sequences to compute the filter output and b) adaptation of filter coefficients according to the measured error. Implementing adaptive filters in the time domain has two major drawbacks;

1) There is a need for a significant amount of computational power for convolving large sequences.

2) The entire adaptation process is performed once per sample of data, hence introducing a further large increase in required computational power.

These drawbacks are somewhat improved by using transform techniques such as the FFT. In this case the number of arithmetic operations needed will be reduced and more importantly the adaptation process will be done once per block of data. However, the drawback in using the FFT is mainly the need for complex arithmetic operations, which are computationaly demanding. Another limitation of using FFT techniques is the problem of finite word length, that is, it is not always possible to represent the result of arithmetic operations, especially multiplication, with precision. In this case the result has to be either truncated or rounded to the nearest value hence introducing errors.

These limitations can be somewhat improved by implementing these transform techniques in a ring of integers such as in the NTT. A study of the published literature on the use of various NTTs such as FNT,MNT,..etc. was carried out and presented to ascertain how these transform techniques can carry out the required

algorithms much more efficiently. With the advantages of NTTs, it is expected to make frequency adaptive digital filters more attractive. This thesis develop and present the design of frequency adaptive filters utilising the NTT. The choice of frequency mean-square error as performance index led to a frequency adaptive filtering formulae which adjusts the filter weights once per data block, by using a frequency mean-square error gradient estimate approach. Convergence properties of FMSE and LMS algorithms are analysed and compared. They are shown to be analogous and under the proper circumstances equivalent. A number of results have been presented, which shows that under certain assumptions the FMSE converges to Weiner filtering.

These assumptions can be summerised as follows;

1) The input data and its corresponding residue representation must be scaled so as to overcome any ambiguity (overflow) in the output result.

2) The convergence constant must take on values in a range that will insure the convergence.

Frequency adaptive digital filters utilising the NTT algorithms are shown to involve less computational complexity than the FFT and LMS adaptive filters when implemented on serial processor. This is specially true for large filters. For example, a 1024-tap adaptive filter would be well over ten time more efficient. Further speed gains are expected in implementations using parallel processing and VLSI.

7.2. CONCLUSION ON THE VLSI DESIGN FOR ADF

The most important conclusion from this study was that concurrency for compute-bound computation is an important aspect of any design in order to utilise the advantages of VLSI technology. This is because the speed of discrete digital components are increasing very rapidly and therefore, in order to meet the required computational power for digital filters, it is necessary to use parallel and multiprocessing techniques, and in particular pipelining and systolic arrays, which are

suitable for VLSI implementation. It is ascertained by providing a survey of published literature on the design of various digital signal processors.

It was concluded that, fully effective use of VLSI technology for high performance digital signal processors to carry out a specific task, requires special purpose VLSI chips. In common with other researchers, it is found that architecture for high throughput circuits using VLSI technology when there is;

1) Simple flow of data and control.

2) locally connected, so as to minimise the long distance communication.

3) Simple and regular cells, so as to minimise the design and fabrication time.

With the features outlined above, and the fact that multiprocessing and pipelining is desirable a design of an adaptive digital filter utilising the FNT and residue numbering system has been presented. A full description of the design is given in chapter 6. The main characteristics are;

1) It is capable of performing the desired arithmetic operations required by NTT with the aid of a table-lookup approach.

2) Massive pipelining and parallelism have been achieved.

3) Flow of data is regular and the control flow is simple and need only a simple clock for the movement of data between the processing elements.

There are several conclusions which can be drawn from the design presented in chapter 6. First, the realisation of high or partially parallelism on a single chip is not possible for long sequences. Therefore it has to be fabricated on several chips. Second, the interconnection between the processing elements in the 1st stage of the transform can be cumbersome when it has to be implemented on chip. Third, the area of the table-look ups processing elements (namely the PLAs) is of great importance as these occupy the greatest proportion of chip area. Because each PLA consists of a few simple cells, the design time is minimised. However, the area which they occupy is large, this is because the size of each plane matrices of the PLA depends upon the number of product terms and the number of product terms may be high. It is therefore

possible and highly desirable to reduce the number of product terms and hence the area by using a minimisation program and a proper coding of the input data.

Finally, in presenting the adaptive digital filter architecture a number of aspects of design have been considered. The largest processing block is clearly the transform and inverse transform. In fact, the multiplier array of the adaptation process can also be realised by a set of PLAs. The adaptation rate of the filter will be fast since only the minimum number of coefficients need be updated. The size of the PLAs in the design may appear somewhat daunting. However, at the expense of some regularity, these can be reduced very considerably by the introduction of decoders at the inputs. It was shown in chapter 6 that, for a filter length of four and dynamic range of 4-bits using the Fermat number transform, it is possible to integrate between 8-16 PLAs on a silicon area of 6mm by 6mm, However, for more realistic filter size, it may be possible to place between 1-6 PLAs on a silicon depending on the modulo, and the dynamic range.

In conclusion, a very promising VLSI architecture has been presented for adaptive digital filtering. The major characteristics of the design are its very high sampling rate capability, regular internal structure, capability to parallel devices for increased word length, sampling rate and adaptation time.

7.3. FUTURE WORK

As a result of the study presented in this thesis, the author would suggest three main areas for future research:-

a) Finite number system and NTT

The main drawbacks of such transform techniques are:-

1) The rigid relationship between the transform length, word length, and root of unity.

This is because NTT has to support the residue arithmetic operations. However, a number of algorithms such as FNT, MNT, CFNT etc. have been adopted which allow for compromise between these relationships.

In fact it is difficult to specify which of these techniques has advantages over the others. However, these transform techniques can be compared, by considering computational complexity (number of additions and multiplications). Several results were examined, and from them it can be concluded that the most promising of all these transform techniques are the FNT and CFNT, due to the fact that it needs fewer additions and multiplications for a given transform length.

i) Choice of modulo

By choosing a non-Fermat or Mersenne number but a prime modulo (while holding the NTT conditions sets in chapter 3), it is possible to reduce the word length while retaining a modest transform length.

ii) Multi-dimensional transform

There is a need for exploiting the properties of multi-dimensional transform techniques for both high computational power and long transform lengths.

2) Overflow detection and wordlength

The overflow problem can be overcome by choosing an appropriate word length, usually the number of bits used for representing the data samples should be half of the word length. However, the implementation of systems with large word length could be costly. There are however, a number of steps that could be taken in order to solve the word length problem:

i) Segmentation of data samples

A scheme to overcome the problem of word length is to segment the input words into several sets of smaller words and process each of these separately and combine the results.

ii) Other transform techniques

Because many NTT algorithms require large word length, it is necessary to study other orthogonal transforms and their properties regarding the adaptive digital filtering with short word length such as Walsh transform and their realisation in VLSI.

b) Frequency adaptive digital lattice filtering

We have considered in our work frequency adaptive digital filtering for finite impluse response (non-recursive) filters. However, these are not the only possible types of filtering. Many authors [108-110] have presented another class of adaptive filter structure known as lattice digital adaptive filters, where the adaptation process has been performed in the time domain. Hence, another interesting area for future work would be the implementation of these filters in frequency domain utilising the NTT.

c) VLSI architecture

There are various changes that could be made to the design of the system presented in this thesis that would increase the system performance i.e. time/area complexity:

i) The use of decoders at the PLA inputs, and the PLA folding technique.

ii) The use of other coding techniques rather than the binary representation of the residue numbers for table look-up operation which has been used in this work.

iii) Custom cell design for the processing elements. It is highly desirable to design a set of custom hand crafted cells for the required processing elements in order to minimise the silicon area. These cells can then be part of a standard cell library which can be used for the implementation of a given semi-custom design.

iv) Other pipeline architectures.

Appendix A

Convergence proof of FLMS

The approach taken in the proof is to show that the weight vector converges in frequency in the mean to the Wiener weight vector, as the data block sequence approaches infinity. The proof is based upon [Widro,33].

For the purpose of the following proof, we assume:

1) The time between succesive iteration of FLMS is long,so the succesive data sequences are uncorrelated.

2) Because the weight vector is only the function of previous data samples, therefore the next weight vector is independant of present data samples.

3) all inputs are satationary.

Recall Eq. (3.2.5)

$$|W^{j+1}|_M = |W^j|_M + 2\cdot\mu_F\cdot|X^{\tau^j}|_M\cdot|E^j|_M \tag{A.1}$$

Expanding further and take expectation of both side:

$$\epsilon[|W^{j+1}|_M] = \epsilon[|W^j|_M] + 2\cdot\mu_F\cdot\epsilon[|X^{\tau^j}|_M\cdot|E^j|_M]$$

$$= \epsilon[|W^j|_M] + 2\cdot\mu_F\cdot\epsilon[|X^{\tau^j}|_M\cdot|D^j|_M] - 2\cdot\mu_F\cdot\epsilon[|X^{\tau^j}|_M\cdot|X^j|_M\cdot|W^j|_M]$$

Using condition 2, the Eq. (A.2) becomes:

$$= \epsilon[|W^j|_M| + 2\cdot\mu_F\cdot\epsilon[|X^{\tau^j}|_M\cdot|D^j|_M] - 2\cdot\mu_F\epsilon[|X^{\tau^j}|_M\cdot|X^j|_M]\cdot\epsilon[|W^j|_M]$$

$$= \epsilon[|W^j|_M] + 2\cdot\mu_F\cdot|\phi_2|_M - 2\cdot\mu_F\cdot|\phi_1|_M\cdot\epsilon[|W^j|_M]$$

Using the properties of $|\phi_1|_M$ and $|\phi_2|_M$,a vector difference equation in the expected value of the weight vector is obtain.

$$\epsilon[|W^{j+1}|_M] = [I-2\cdot\mu_F|\phi_1|_M]\cdot\epsilon[|W^j|_M] + 2\cdot\mu_F\cdot|\phi_2|_M$$

Where "I" is the identity matrix.

Our analysis is based on the scalar case of vector operation, thus, Eq. (A.3) is the first order linear differece equation in the weight, whose explicit solution [36] is given by:

$$\epsilon[\,|W^{j+1}|_M\,] = [I-2\cdot\mu_F\cdot|\phi_1|_M]^{j+1}\cdot|W^0|_M$$

$$+\ 2\cdot\mu_F\,|\phi_2|_M\cdot\sum_{i=0}^{j}[I-2\cdot\mu_F\cdot|\phi_1|_M]^i \qquad (A.4)$$

Where $|W^0|_M$ is the initial weight vector.

Eq (A.4) can be written in diagonal form by using appropriate " SIMILARITY TRANSFORM P " [37,38] for the matrix $|\phi_1|_M$, that is:

$$|\phi_1|_M = \epsilon[\,|X^{\tau^j}|_M\cdot|X^j|_M\,] = |P|_M\cdot|\Lambda|_M\cdot|P|_M^{-1} \qquad (A.5a)$$

where

$$\Lambda = \begin{bmatrix} \Lambda_1 & 0 & 0 & 0 \\ 0 & \Lambda_2 & 0 & 0 \\ 0 & 0 & . & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \Lambda_j \end{bmatrix} \qquad (A.5b)$$

is the diagonal matrix of eigenvalue of matrix $|\phi_1|_M$. Because $|\phi_1|_M$ is a positive definite, thus all the eigenvalues are positive.

Rewrite Eq (A.4)

$$\epsilon[\,|W^{j+1}|_M\,] = [I-2\cdot\mu_F\cdot|P|_M\cdot|\Lambda|_M\cdot|P^{-1}|_M]^{j+1}\cdot|W^0|_M$$

$$+\ 2\cdot\mu_F\cdot|\phi_2|_M\cdot\sum_{i=0}^{j}[I-2\cdot\mu_F\cdot|P|_M\cdot|\Lambda|_M\cdot|P^{-1}|_M]^i$$

$$=\ |P|_M\cdot[I-2\cdot\mu_F\cdot|\Lambda|_M]^{j+1}\cdot|P^{-1}|_M\cdot|W^0|_M$$

$$+\ 2\cdot\mu_F\cdot|\phi_2|_M\cdot|P|_M\cdot\sum_{i=0}^{j}[I-2\cdot\mu_F\cdot|\Lambda\ |_M]^i\cdot|P^{-1}|_M \qquad (A.6)$$

The key to Eq (A.6) is to show that:

$$[I-2\cdot\mu_F\,|P|_M\cdot|\Lambda|_M\cdot|P^{-1}|_M]^{j+1} = |P|_M\cdot[I-2\cdot\mu_F\,|\Lambda|_M]^{j+1}\cdot|P^{-1}|_M$$

Properties used in this development are the Binomial formula and the fact that for diagonal forms [A.4] it is true that

$$(|P|_M \cdot |\Lambda|_M \cdot |P^{-1}|_M)^l = |P|_M \cdot |\Lambda^l|_M \cdot |P^{-1}|_M$$

Thus, as long as the diagonal matrix of eigenvalues outside Galios field is less than unity, the first term of Eq (A.6) is zero as $j \rightarrow \infty$

$$\lim_{j\to\infty} \sum_{i=0}^{j} [I - 2\cdot\mu_F \cdot |\Lambda|_M]^{j+1} \rightarrow 0 \qquad \text{(A.7)}$$

and vanishes as number of iteration increases.

Consider the second term in Eq (A.6). Becuse $|\Lambda|_M$ is diagonal , the N summation of the geometric formula gives:

$$\lim_{j\to\infty} \sum_{i=0}^{j} [I - 2\cdot\mu_F \cdot |\Lambda|_M]^i = \frac{1}{2\cdot\mu_F \cdot |\Lambda^{-1}|_M} \qquad \text{(A.8)}$$

Thus, in the limit the Eq (A.6) becomes:

$$\lim_{j\to\infty} \epsilon[\,|W^{j+1}|_M] = |P|_M \cdot |\Lambda^{-1}|_M \cdot |P^{-1}|_M \cdot |\phi_2|_M$$

$$= |\phi_1|_M^{-1} \cdot |\phi_2|_M \qquad \text{(A.9)}$$

Compare this with Wiener-Hopf equation

$$W^* = R^{-1} \cdot P'$$

shows that as the number of iteration increases without limit the expected value of weight vector converges to Wiener solution.

The convergence insured if and only if μ_F is set within a certain bound:

$$|[I - 2\cdot\mu_F \cdot \Lambda_{\max}]| < 1$$

$$0 < \mu_F < \frac{1}{\Lambda_{\max}} \qquad \text{(A.10)}$$

The condition in Eq (A.10) is sufficient for convergence but the individual eigenvalue of $|\phi_1|_M$ are rarely known and thus, Eq (A.10) is not always easy to apply. Another quantity that is more easily measured is the 'TRACE" of $|\phi_1|_M$:

$$tr(|\phi_1|_M) = \sum_{i=0}^{N-1} \Lambda_i = totalinputpower$$

the fact that $tr(|\phi_1|_M) > \Lambda_{max}$ suggest that Eq (A.10) becomes:

$$0 < \mu_F < \frac{1}{tr(|\phi_1|_M)} \qquad \text{A(11)}$$

APPENDIX B

ADAPTATION SPEED

Recall equation 4.2.1.5

$$\epsilon[\,|E_j^T|_M \,.\, |E_j|_M] = \epsilon[\,|D_j^T|_M \,.\, |D_j|_M] - 2\,|\phi_2|_M\,|W_j|_M + |W_j^T|_M \;\; |\phi_1|_M \;\; |W_j|_M$$

The FMSE gradient for weight vector is ;

$$\nabla_F = -2\,|\phi_2|_M + 2\,|W_j^T|_M \;\; |\phi_1|_M \tag{B.1}$$

The minimum MSE can be obtain from orthogonality principle [35] which is ;

$$E_{\min} = \epsilon[\,|D_j^T|_M \,.\, |E_j|_M]$$

$$= \epsilon[\,|D_j^T|_M(|D_j|_M - |X_j|_M \,.\, W_j^*)]$$

$$= \epsilon[\,|D_j^T|_M \,.\, D_j] - \epsilon[\,|D_j^T|_M \,.\, |X_j|_M \,.\, W_j^*]$$

$$= \epsilon[\,|D_j^T|_M \,.\, D_j] - |\phi_2|_M \,.\, W_j^*$$

$$= \xi_{\min}$$

Substiute B.3 in 4.2.1.5 yields an alternative formula;

$$\epsilon[\,|E_j^T|_M \,.\, |E_j|_M] = E_{\min} + (|W_j|_M - W_j^*)^T \,.\, |\phi_1|_M \,.\, (|W_j|_M - W_j^*) \tag{B.4}$$

Let us define some quantity that transform FMSE into diagonal form ;

$$|V_j|_M = |W_j|_M - W_j^* \tag{B.5}$$

Thus equation(B.4) becomes :

$$E = E_{\min} + |V_j^T|_M \,.\, |\phi_1|_M \,.\, |V_j|_M \tag{B.6}$$

Another form of gradient is formed by differentiating eq(B.6)

$$\nabla_F = 2 \,.\, |\phi_1|_M \,.\, |V_j|_M \tag{B.7}$$

Using the Similarity transform

$$|\phi_1|_M = P \,.\, |\Lambda|_M \,.\, P^{-1} = P \,.\, |\Lambda|_M \,.\, P^T \tag{B.8}$$

Where P is the orthomormal model matrix of $|\phi_1|_M$. Thus ;

$$E = E_{\min} + |V_j^T|_M \cdot P \cdot |\Lambda|_M \cdot P^{-1} \cdot |V_j|_M \tag{B.9}$$

By linear transformation ;

$$|V_j'|_M = P^{-1} \cdot |V_j|_M \qquad |V_j|_M = P \cdot |V_j'|_M \tag{B.10}$$

Using this new corrdinate system , eq(B.9) becomes ;

$$E = E_{\min} + |V_j'^T|_M \cdot |\Lambda|_M \cdot |V_j^7|_M \tag{B.11}$$

A new weight vector is obtain by appling the same transform corrdinates ;

$$W_j' = P^{-1} \cdot W_j \qquad W_j = P \cdot W_j' \tag{B.12}$$

THE METHOD OF STEEPNESS

Recall equation 4.3.1

$$|W_{j+1}|_M = |W_j|_M + \mu_F \cdot \nabla_F \tag{B.13}$$

Taking the advantages of equations B.7-B.12 then eq(B.13) can be written as;

$$|V_{j+1}'|_M - (I - 2 \cdot \mu_F \cdot |\Lambda|_M) \cdot |V_j'|_M = 0 \tag{B.14}$$

This diagonal homogenous vector difference equation has a geometric solution ;

$$|V_j'|_M = (I - 2 \cdot \mu_F \cdot |\Lambda|_M)^j \cdot |V_0|_M \tag{B.15}$$

Where V_0 is the initial vector weight .

The transient solution of the eq(B.14) is geometric with geometric ratio of p^{th} mode being ;

$$r_{pF} = (1 - 2 \cdot \mu_F \cdot |\Lambda_p|_M) \tag{B.17}$$

To define a time constant for the solution , an exponential envelope can be fitted to the geometric ratio . Call the frequency time constant T_{pF}' , thus;

$$r_{pF} = [1 - 2 \cdot \mu_F \cdot |\Lambda_p|_M] = e - \frac{1}{T_{pF}'}$$

$$1 - \frac{1}{T_{pF}'} + \frac{1}{4 \cdot T_{pF}'} - \cdots\cdots\cdots \tag{B.18}$$

r_{pF} can be estimated by first two terms of eq(B.18)

$$r_p = 1 - 2 \cdot \mu_F \cdot |\Lambda_p|_M = 1 - \frac{1}{T_p} \tag{B.19}$$

Thus;

$$T_p = \frac{1}{2 \cdot \mu_F \cdot |\Lambda_p|_M} \tag{B.20}$$

TIME CONSTANT FOR LEARNING CURVE

Consider the FMSE during the adaptation as the weight vector adapts towards Wiener solution . Recall eq(B.11) ;

$$|E_j|_M = E_{\min} + |V_j'^{\tau}|_M \cdot \Lambda \cdot |V_j'|_M \tag{B.21}$$

Assume no noise in weights vector and using eq(B.15)

$$|E_j|_M = E_{\min} + |V_0'^{\tau}|_M \cdot |\Lambda|_M (I - 2 \cdot \mu_F \cdot |\Lambda|_M)^{2j} \cdot |V_0'|_M$$

$$= E_{\min} + |V_0'^{\tau}|_M (I - 2 \cdot \mu_F \cdot |\Lambda|_M)^j \cdot |\Lambda|_M \cdot (I - 2 \cdot \mu_F \cdot |\Lambda|_M)^j \cdot |V_0'|_M$$

$$= E_{\min} + \sum_{p=0}^{N} (1 - 2 \cdot \mu_F \cdot |\Lambda_p|_M)^{2j} \cdot |\Lambda_p|_M \cdot (|V_{p0})^2|_M \tag{B.22}$$

Where $|\Lambda_p|_M$ is the pth eigenvalue of $|\phi_1|_M$.

and V_{p0}' is the pth component of V_0' .

The FLMS decays with a geometric ratio for the pth mode ;

$$r_{FMSE_p} = (1 - 2 \cdot \mu_F \cdot |\Lambda_p|_M)^2 \tag{B.23}$$

Where μ_F is chosen so that FLMS algorithm converges .

Let us assume :

$$\lim_{j \to \infty} E_{\min} = \xi_{\min}$$

This follows that

$$\lim_{j \to \infty} (I - 2 \cdot \mu_F \cdot |\Lambda|_M)^j = 0$$

Thus , the geometric ratio r_p can be used to define a time constant :

$$r_{FMSE_p} = e^{\frac{-1}{T_{MSE}}} = (1 - 2 \cdot \mu_F \cdot |\Lambda_p|_M)^2$$

$$= r_p^2 = e^{\frac{-2}{T_p}}$$

$$T_{FMSE} = \frac{1}{2 \cdot T_p} = \frac{1}{4 \cdot \mu_F \cdot |\Lambda_p|_M} \tag{B.24}$$

In special case where all eigenvalues are equall;

$$T_{FMSE} = \frac{1}{4 \cdot \mu_F \cdot |\Lambda|_M} \tag{B.25}$$

$$T_{FMSE} = \frac{1}{4N \cdot \mu_F \cdot trace(|\phi_1|_M)}$$

Where :

$$trace = input\ \ power = \frac{1}{N} \cdot \sum_{j=1}^{N} |\Lambda_j|_M$$

Appendix C

TOPICS IN NUMBER THEORY

Congruence and Equivalence relation

Definition 1 : If the difference of two integers a,b is divisible by M , we shall say that a and b are congruent modulo M and can be written as :

$$a \equiv b \mod M$$

Example : Let us consider the set H = {h} consisting of those positive rational integers that of the form h = 4n +1 . When these integers are divided by 4 , they leave a remainder equal to 1 . Such i ntegers are said to be congruent to 1 modulo 4 .

THEROEM 1 : The following congruences are equivalent modulo any integer M (that is each implies and implied by each one of the other three):

$$a \equiv b \quad b \equiv a \quad a-b \equiv 0 \quad b-a \equiv 0$$

Definition 2 : If x is an integer and $b \equiv x \mod M$, then b is said to be a Residue of x modulo M . If $0 \le b < M$ then b is called LEAST POSITIVE RESIDUE of x modulo M . If $\frac{-M}{2} < b \le \frac{M}{2}$ then b is called a LEAST RESIDUE of x modulo M .

Definition 3 : A set of integers is called a complete set of residues , if no two of them are congruent and if every ratio nal integer is congruent to one of them .

Definition 4 : i) Given a set S of elements (not necessarily integers) , any set R = {(a,b)} of ordered pairs , a $\in$ S , b $\in$ S is called a " RELATION " if a $\in$ S , b $\in$ S and (a,b) $\in$ R . We say that "a" is in relation R to "b" and write a R b .

ii) A relation R among the elements of a set S is said to be :

a) REFLEXIVE if a S a R a

b) SYMMETRIC if a,b S a R b b R a

c) TRANSITIVE if a,b,c S a R b , b R c a R c

iii) A relation R which is reflexive , symmetric , transitive is said to be an " EQUIVALENCE RELATION " .

iv) If R is an equivalence relation and a R b then a is said to be equivalent to b under R .

THEROEM 2 : For any integer M , the congruence modulo M , is an equivalence relation.

proof : By Theroem 1 and definition 1 modulo any integer M , $a \equiv a$ and also $a \equiv b \;\; b \equiv a$ so we left to check the transitivity if

$$a \equiv b \;\; b \equiv c \;\; then \;\; a \equiv c$$

a - b = kM b - c = lM hence a - c = (a - b) + (b - c) = (k + l)M implies $M\frac{M}{(a-c)}$ $a \equiv c$

Definition 5 : Given a set S and a relation R on S , all elements equivalent under R to a given one are said to form an " EQUIVALENCE CLASS " .

Definition 6 : The equivalence classes induced by the congruence modulo M are called " RESIDUE CLASSES MODULO M " .

THEROEM 3 : The sets $0 \le r \le M$ and $\frac{-M}{2} \le r \le \frac{M}{2}$ form complete sets of residues.

Definition 7 : the set $0 \le r \le M-1$ is called a complete set of least positive residue ; and the $\frac{-M}{2} < r \le \frac{M}{2}$ is called a complete set of least residue .

THEROEM 4 : The following statements hold ;

1) $a \equiv b \qquad ca \equiv cb$

2) $a \equiv b, c \equiv d \qquad a + c \equiv b + d$

3) $a \equiv b, c \equiv d \qquad ar + cs \equiv br + ds$

4) $A \equiv b, c \equiv d \qquad ac \equiv bd$

5) $a \equiv b \qquad a^n \equiv b^n$

THEROEM 5 : If P(x) is a polynomial with integer coefficient , and $a \equiv b$, then $P(a) \equiv P(b)$.

So far , the properties of congruence appear to be almost identical with the corresponding properties of ordinary equality. This observation illustrates well the point , that to some extent the properties of an equivalence relation are d ue to the fact that it is an equivalence relation . Ordinary equality and congruence (modulo M) , being both equivalence relations , share , of course , all those properties due precisely to the fact that they are equivalence relations . It might , therefore , be appropriate to point out at least one difference between two relations ;

if c a = c b and $c \neq 0$ then we may cancel the common factor c and infer that a = b . However if $c \equiv 0$ (mod M) means c is not prime to M and $ca \equiv cb$ we can not in general conclude that $a \equiv b$.

However , it is comforting to know that a common factor may be canceled in a congruence ,provided that it is coprime to the modulo .

Operations with Residue Classes

Let us consider the set { 0,1,....., M-1 } of least positive residues modulo M. By Theroem 3 , each of these integers belongs to exactly one residue class . All congruences being understood modulo M , let A be a residue class to which belongs the least positive residue r_1

; then $A = \{a \mid a \equiv r_1\}$. Similarly , let $B = \{b \mid b \equiv r_2\}$. By theroem 4 , a A , b B , then $a + b \equiv r_1 + r_2$ and $a \cdot b \equiv r_1 \cdot r_2$.

If r_3 and r_4 are least positive residues such that $r_1 + r_2 \equiv r_3$ and $r_1 \cdot r_2 \equiv r_4$ then for every element a of A and b of B , one has $a + b \equiv r_3$ and $a \cdot b \equiv r_4$.

Moreover , if we define the residue classes $C = \{c \mid c \equiv r_3\}$ and $D = \{d \mid d \equiv r_4\}$, then $a + b \equiv c$ and $a \cdot b \equiv d$ hold , regardless of the particular choice of elements within their residue classes . This shows that the residue class of a sum or of a product does not depend at all on the summend , and factors themselves , but only on their respective resi due classes .

THEROEM 8 : The operation of addition and multipication of residue classes are

well defined . The set of residue classes is closed under both operations .

The set residue classes modulo a prime integer will form a " FIELD " , while those modulo a composite integer will form a " COMMUTATIVE RING " with divisors of zero . The simplest way to aviod these divisors of zero is to restrict our attention to residue classes that are relatively prime to the modulos ; this motivates the following definitions ;

Definition 8 : A residue class $A \quad a \mid a \equiv r \mod M$ is called a prime residue class if

$$(r , M) = 1$$

Definition 9 : A complete set of reduced (or prime) residues is a set $S = r_i$ satisfing the followin g conditions :

i) $i \neq J \qquad r_i \equiv r_j$

ii) r S $\qquad (r , M) = 1$

iii) (a , M) = 1 r S a = r

If , in addition , $0 < r \leq M$, then S is called a reduced set of least positve residues , if $\frac{-M}{2} < r < \frac{M}{2}$ then S is a reduced set of least residues .

In words , a complete set of reduced residues consist of a set of mutually incongruent integers , all coprime to the modulo, and such that every integer coprime to the modulo is conguent to one of them .

1) D.E.Knuth : " The art of programming : Seminumerical algorithm ", vol 2,2nd edition, Addison-Wesley, New York, 1981.

2) G.H.Harday ,E.M.Wright : " An introduction to the theory of numbers ", 4th edition, Oxford Uni. press, London, 1960.

3) I.M.Vinogradov : " An introduction to theory of numbers ", Pergamon press, London, 1955.

4) N.S.Szabo, R.I.Tanaka : " Residue arithmetic and its application to computer tecnology ", McGraw-Hill, New York, 1967.

5) J.M.McClellan, C.M.Rader : " Number theory in Digital Signal Processing ", Prentic-Hall, New York, 1979.

6) N.J.Nussbaumer : " Fast Fourier Transform and convolution algorithms ", Spring-Verlang, Berlin, 1981.

7) T.G.Stockham : " High speed convolution and correlation ", in 1966 Spring Joint Computer Confrence, AFIPS, proc. 28, pp 229-233.

8) R.C.Agarwal, J.W.Cooley : " New algorithms for digital convolution ", IEEE Trans. on Acoustic Speech and Signal Processing, ASSP-25, No 2, 1977, pp 329-410.

9) I.J.Good : " The relationships between two Fast Fourier Transforms ", IEEE Trans. on Computer, C-20, 1971, pp 310-317.

10) H.J.Nussbaumer, P.Quandalle : " Computation of convolution and Discrete Fourier Transform by polynomial transform ", IBM J. Res. Dev., 22, 1978, pp 134-144.

11) P.W.Cheney : " A digital correlator based on the residue number system " IRE trans. on Elecctronic Computers, 1961, pp 63-70.

12) L.R.Rabiner, B.Gold : " Theory and application of digital signal processing, Prentice-Hall, New York, 1975.

13) W.K.Jenkins, B.J.Leon : " The use of residue number system in the design of Finite Impulse Responce digital filters ", IEEE Trans. on circut and system, vol CAS 24, No 4, 1977, pp 191-201.

14) A.Peled and B.Liu : " A new hardware realization of digital filters ", IEEE Trans. on Acoustic, Speech and Signal Processing, vol ASSP-22, 1974, pp 456-462.

15) F.J.Taylor, A.S.Ramnarayanan : " An efficient Residue-to-Decimal converter ", IEEE Trans. on circuit and system, vol CAS-28, No 12, 1981, pp 1164-1169.

16) J.B.Martens : " Number Theoretic Transforms for calculation of convolution ", IEEE Trans. on Acoustic, Speech and Signal Processing, vol ASSP-31, No 4, 1983, pp 969-978

17) P.C.Balla, A.Antoniou : " Number Theoretic Transform based on Ternary arithmetic and its application to cyclic convolution ", IEEE Trans. on Circuit and

System, vol CAS-30, No 7, 1983, pp 504-505

18) M.D.Wagh, S.D.Morgera : " A new structured design method for convolution over Finite Fields ", IEEE Trans. on Information Theory, vol IT-29, No 4, 1983, pp 583-595

19) J.J.Thomas, G.N.Larsen, J.M.Keller : " Number Theoretic Transform with independent lenght and modulo ", IEEE Trans. on Acoustics, Speech and Signal Processing, vol ASSP-31, No 1, 1983, pp 215-217

20) T.K.Truong, K.Y.Liu, I.S.Reed : " A paralell-pipeline architecture of the fast polynomial transform for computing a two-dimentional cyclic convolution ", IEEE Trans. on Computers, vol C-32, No 3, 1983, pp 301-306

21) C.R.Rader : " Discrete convolution via Mersenne transforms ", IEEE Trans. on Computers, vol C-21, No 12, 1972, pp 1269-1273

22) R.C.Agarwal, C.S.Burrus : " Fast convolution using Fermat Number Transform with application to digital filtering " , IEEE Trans. on Acoustics, Speech, Signal processing, vol ASSP-22, No 2 ,1974, pp 87-97

23) R.C.Agarwal, C.S.Burrus : " Number Theoretic Transform to implement fast digital convolution ", Proceedings of the IEEE, vol 63, No 4, 1975, pp 550-560

24)) S.R.Reed, T.K.Troung : " The use of finite fields to compute convolution ", IEEE Trans. on Imformation Theory, vo l IT-21, No 2, 1975, pp 208-213

25) D.F.Elliott, K.R.Rao : " Fast Transforms Algorithms, analyses, applications

", Academic press, New York, 1982

26) J.M.Pollard : " The Fast Fourier Transform in a finite field ", Mathematics of Computation, vol 25, No 114, 1971 , pp 365-372

27) R.C.Agarwal, C.S.Burrus : " Fast digital convolution using Fermat Number Transform ", in South West IEEE conf., 1 973, pp 538-543

28) J.H.McClellan : " Hardware realisation of a Fermat Number Transform " , IEEE Trans. on Acoustics, Speech, Signal Processing, vol ASSP-24, No 3, 1976 ,pp 216-225

29) L.M.Leibowitz : " A simplified binary arithmatic for the Fermat Number Transform ", IEEE Trans. on Acoustics, Speech , Signal Processing, vol ASSP-24, No 5, 1976, pp 356-359

30) R.C.Agarwal, C.S Burrus : " Fast one dimentional digital convolution by multidimentional techniques ", IEEE Trans. on Acoustics, Speech, Signal Processing, vol ASSP-22, No 1, 1974, pp 1-10

31) N.J.Nussbaumer : " Digital fieltering using Complex Mersenne Transforms ", IBM J. Res. Dev., Vol 20, No 50, 1976 , pp 498-504

32) N.J.Nussbaumer : " Digital filtering using Pseudo Fermat Transform ", IEEE Trans. on Acoustics, Speech, Signal Pr ocessing, vol ASSP-25, No 1, 1977, pp 79-83

33) B.Widrow et.al : " Adaptive noise cancelling ; principle and applications ", Proceedings of IEEE, vol 63, No 12 , 1975, pp 1692-1716

34) B.Widrow et.al : "Stationary and non-stationary learing characteristics of LMS adaptive filters ", Proceedings of IEEE, vol 64, No 8, 1976

35) A.P.Sage, J.L.Melsa : "Estimation theory with application to communication and control ", Mc Graw-Hill 1971

36) C.W Celia : "An introduction to numerical analysis", McGraw-Hill, London, 1969

37) G.W.Stewart : "Introduction to matrix computations ", Academic press, 1973 ,USA, pp 275-289

38) N.J.Pullman : " Matrix theory and its applications ", Marcel Dekker, 1976 , USA, pp 5-19

39) G.Strang : " Linear Algebra and its applications ", Academic press, 1980, New York

40) C.F.N.Cowan, J Mavor : " Miniture CCD-based anolog adaptive filters ", International Conference on Acoustic, Speech, Signal processing, 1980, pp 474-477

41) B.Gold, A.Oppenhiem : " Digital signal processing ", 1969

42) B.Widrow el.at : " A comparison of adaptive algorithms based on the method of steepest descent and random search ", IEEE Trans. Antenna and Propagation,

vol AP-24, No 5, 1976

43) A.V.Oppenhiem : " Applications of digital signal processing ", Prentice-Hall, USA, 1978

44) E.E.Swartzlander et.al : " Inner product computers ", IEEE Trans. on Computers, vol C-27, No 1, 1978, pp 21-31

45) W.T.Rhodes : " Acoustic-optical signal processing ", Proceedings of IEEE, vol 69, No 1, 1981, pp 65-76

46) T.M.Turpin : " Spectrum analysis using optical processing ", Proceedings of IEEE, vol 69, No 1, 1981, pp 79-92

47) P.Kellman et.al : " Integrated acoustic-optic channelised receiver ", Proceedings of IEEE, vol 69, No 1, 1981, pp 93-100

48) P.A.Avon : Ph.D thesis, University of kent, Canterbury, 1983

49) A.Brine : Ph.D thesis, University of kent, Canterbury, 1983

50) R.White , H.T.Nagle : " Digital realisation using special purpose stored-program computer ", IEEE Trans. on Audio and Electronics, vol AU-20, No 4, 1972, pp 289-294

51) H.L.Groginsky , G.A.Works : " A pipline fast Fourier transform ", IEEE Trans. on Computers, vol C-19, No 11, 1970, pp 1015-1019

52) S.L.Freeny : " Special purpose hardware for digital filtering ", Proceedings of IEEE, vol 63, No 4, 1975, pp 633-648

53) S.S.Magar , D.A.Rubinson : " Microprogrammable arithmatic element and its applications to signal processing ", Proceedings of IEE, vol 127, part F, No 2, 1980, pp 99-106

54) B.Gold , T.Bially : " Parallelism in fast Fourier transform hardware ", IEEE Trans. on Audio and Electronics, vol AU-21, No 1, 1973, pp 5-16

55) K.Murano , S.Unagami , T.Tusda : " LSI processor for digital signal processing and its applications to 4800 bit/s modem ", IEEE Trans. on Communications, vol COM-26, No 5, 1978, pp 499-506

56) S.C.Sfetcu , J.Doyle : " A low cost real-time service digital signal processor ", IEEE Trans. on Communications, vol COM-26, No 5, 1978, pp 626-631

57) Y.S.Wu : " Architecture considrations of a signal processing under microprogram control ", Spring Joint Computer Conference, 1972, pp 675-683

58) H.Aiso et.al : " A very high-speed microprogrammable pipline signal processor ", IFIP Conference, 1974, pp 60-64

59) P.Chow , Z.Vranesic , J.Yen : " Microprocessor implementation of discrete Fourier transform ", International Conference on Acoustics, Speech, Signal processing, 1979, pp 316-320

60) J.A.Feldman , E.M.Hofstetter , M.L.Maplass : " A compact flexible LPC vocoder based on commercial signal processing microcomputer ", IEEE Trans. on Acoustics, Speech, Signal processing, vol ASSP-31, No 1, 1983, pp 252-257

61) R.Demori , S.Rivoira , A.Serra : " A special purpose computer for digital signal processing, IEEE Trans. on Computers, vol C-24, No 12, 1975, pp 1202-1211

62) R.F.Lyon : " Two's complement pipline multipliers ", IEEE Trans. on Communications, vol COM-24, No 3, 1976, pp 418-425

63) A.Habibi , P.A.Wintz : " Fast multipliers ", IEEE Trans. on Computers, No 1 , 1970, pp 153-157

64) A.Peled , B.Liu : " A new hardware realisation of digital filters ", IEEE Trans. On Acoustics, Speech and Signal processing, vol ASSP-22, No 6, 1974, pp 456-462

65) A.Peled , B.Liu : " A new hardware realisation of high speed fast Fourier transforms ", IEEE Trans. on Acoustics, Speech and Signal processing, vol ASSP-23, No 6, 1975, pp 543-547

66) A.peled , B.Liu : " Implementation of dedicated hardware special purpose digital signal processor ", In Digital signal processing, Theory, Design, and implementation, John Wiley, 1976, London, pp 212-238

67) A.Peled , B.Liu : " On the hardware implementation of digital signal processing ", IEEE Trans. on Acoustics, Speech and Signal processing, vol

ASSP-24, No 1, 1976, pp 76-86

68) A.V.Oppenhiem , R.W.Schafer : " Digital signal processing ", Prectice-Hall, USA, 1975

69) C.F.Cowan , J.Mavor : " New digital adaptive filter implementation using distributed arithmetic technique ", Proceeding of IEE, vol 128, part F, No 4, 1981, pp 225-230

70) W.Y.Dere , D.J.Sakrison : " Berkeley array processor ", IEEE Trans. on Computers, vol C-19, No 4, 1970, pp 444-447

71) G.D.Hornbukle , E.I.Ancona : " The LX-1 microprocessor and its application to real-time signal processing ", IEEE Trans. on Computers, vol C-19, No 8, 1970, pp 710-720

72) P.E.Blankenship , A.H Huntoon , V.J.Sferrion : " LSP/2 programmable signal processor ", Proceeding of Nat. Electronic Conf. Oct 16-18, 1974, pp 416-421

73) J.R.Fisher : " Architecture and applications of the SPS-41 and SPS-81 programmable digital signal processor ", EASCON Conf. Oct 7-9, pub. IEEE, 1974, pp 674-678

74) J.V.Harshman : " Architecture of a programmable digital signal processor ", Nat. Telecommunication Conf. record Dec 2-4, 1974, pp 496-500

75) P.Thirion : " Digital signal processing with program synchronization between two microprocessor ", IEEE Trans. on Communication, vol COM-26, No 5, 1978, pp513-517

76) K.Watanabe , K.Inoue , Y.Sato : " A 4800 bit/s microprocessor data modem ", IEEE Trans. on Communications, vol COM-26, No 5, 1978, pp 493-498

77) P.D.Stigall , R.E.Ziemer , V.T.Pham : " Performance studies of microcomputer-implemented Fast Fourier Transform ", International mini and micro computer Conf., pub. IEEE, 1979, pp 187-190

78) K.J.Thurber , D.Bennett , L.Smith , Y.Kim : " Comparison of computer architecture for radar signal processing ", SIPE, vol 180, 1979, pp 80-97

79) M.Townsend , M.Hoff , R.Holm : " An NMOS microprocessor for analog signal processing ", IEEE Trans. on Computers, vol C-29, No 2, 1980, pp 97-101

80) J.Zeman , H.Nagle : " A high speed microprogrammable digital signal processor employing distributed arithmetic ", IEEE Trans. on Computers, vol C-29, No 2, 1980, pp 134-144

81) W.Luk , H.Li : " Microcomputer based real-time /online FFT processor ", Proceeding of IEE, vol 127, part E, No 1, 1980, pp 18-23

82) J.Hesson , F.Gallagher , D.Harrington : " A 32 bit programmable signal processor for a multiprocessor system enviroment ", IEEE Trans. on Acoustics, Speech and Signal Processing, vol ASSP-31, No 4, 1983, pp 912-921

83) J.S.Thompson , S.K.Tewksbury : " LSI signal processor architecture for telecommunication application ", IEEE Trans. on Acoustics, Speech, and Signal Processing, vol ASSP-30, No 4, 1982, pp 613-632

84) J.R.Boddie et.al : " Digital signal processor, a programmable integrated circuit ", The BELL system technical journal, vol 60, No 7, part 2, 1981, pp 1431-1563

85) F.Mintzer , K.Davies , A.Peled , F.Ris : " The Real-Time Signal Processor ", IEEE Trans. on Acoustics, Speech, and Signal Processing, vol ASSP-31, No 1, 1983, pp 83-97

86) K.Inove et.al : " A single CMOS speech synthesis chip and new synthesis techniques ", IEEE Trans. on Acoustics, Speech, and Signal Processing, vol ASSP-31, No 1, 1983, pp 335-338

87) Y.Hagiwara et.al : " A single chip digital signal processor and its application to real-time speech analysis ", IEEE Trans. on Acoustics, Speech, and Signal Processing, vol ASSP-31, No 1, 1983, pp 339-347

88) K.Muller-Glasser , T.Canzler , P.King , H.Schulete : " A 24-bit microprocessor for data communication system designed on the basis of a general cell library ", IEEE Journal on Solid-State circuits, vol SC-18, No 3, 1983, pp 250-260

89) J.M.Flynn : " Very high-speed computing system ", Proceedings of IEEE, vol 54, 1966, pp 1901-1906

90) H.S.Stone : " Introduction to computer architecture ", SRA computer scince series, USA, 1975

91) C.Mead , L.Conway : " Introduction to VLSI system ", Addison-Wesley, USA, 1980

92) H.T.Kung : " The structure of parallel algorithms ", in Advances in computers, vol 19, Academic Press, 1980, pp 65-112

93) H.T.Kung : " Why systolic architectures ", Computer, vol 15, No 1, 1982, pp 37-47

94) H.T.Kung : " Let's design algorithms for VLSI systems ", Proceeding Conf. Very Large Scale Integration, California Institue of Technology, 1979, pp 65-90

95) M.J.Foster , H.T.Kung : " The design of special-purpose VLSI chips ", IEEE computer magazine, vol 13, No 1, pp 26-40

96) R.A.Evans et.al : " A CMOS implementation of a systolic multi-bit convolver chip ", In VLSI design of digital system, Ed. F.Anceau , E.Aas, North-Holland 1983, pp 227-237

97) P.R.Cappello , K.Steiglitz : " Digital signal processing applications of systolic algorithms ", In VLSI systems and computations, Ed. H.T.Kung et.al, Spring verlang, Berlin 1981, pp 245-255

98) H.T.Kung, L.M.Ruane, D.W.Yen : " A two-level pipelined systolic array for convolutions ", In VLSI systems and computations, Ed. H.T.Kung et.al, Spring-

verlang, Berlin 1981, pp 255-264

99) A.L.Fisher, H.T.Kung : " Synchronizing large systolic arrays ", SPIE, vol 341 Real time signal processing, 1982, pp 44-52

100) G.A.Clark, et.al : " Block implementation of adaptive digital filters ", IEEE Trans. Acoustics, Speech, and Signal Processing, vol ASSP-29, No 3, 1981, pp 744-752

101) F.Reed, P.L.Feintuch : " A comparison of LMS adaptive cancellers implemented in the frequency and the time domain ", IEEE Trans. on Acoustics, Speech, and Signal Processing, vol ASSP-29, No 3, 1981, pp 770-775

102) D.Mansour, A.H.Gray : " Unconstrained frequency adaptive filter ", IEEE Trans. on Acoustics, Speech, and Signal Processing, vol ASSP-30, No 5, 1982, pp 726-734

103) M.Dention, J.McCool : " Adaptive filtering in frequency domain ", Proceedings of IEEE, vol 66, 1978, pp 1658-59

104) F.J.Taylor : " Large moduli multipliers for signal processing ", IEEE Trans. on Circuit and System, vol CAS-28, No 7, 1981,pp 731-736

105) F.J.Taylor : " Memory intensive multipliers for signal processing ", IEEE Trans. on Acoustics, Speech, and Signal processing, vol ASSP-31, No 6, 1983, pp 1579-82

106) T.Kailatn : " A view of three decades of linear filtering theory ", IEEE Trans. on Information Theory, vol IT-20, no 3, 1974, pp 145-181

107) B.Widrow et.al : " Adaptive antenna systems ", Proceddings of IEEE, vol 55, No 12, 1967

108) D.T.Lee at.el : " Recursive least square ladder estimation algorithms ", IEEE Trans. on Acoustics, Speech, and Signal processing, vol ASSP-29, No 3, 1981, pp 627-642

109) M.L.Honig , D.G.Messerschmitt : " Convergence properties of an adaptive digital lattice filter ", IEEE Trans. on acoustics, Speech, and Signal processing, vol ASSP-29, No 3, 1981, pp 642-654

110) J.Makhoul : " A class of all-zero lattice digital filters ", IEEE Trans. on Acoustics, Speech, and Signal processing, vol ASSP-26, No 4, 1978, pp 304-314

111) J.Angus : Ph.D thesis, University of Kent, Canterbury, 1984

112) J.Bagherli : Mini-user manual, Dept. of Electronics, University of Kent, 1983