Number theoretic transform implementation using microprocessors

Martin, Sean C.

NUMBER THEORETIC TRANSFORM IMPLEMENTATION USING MICROPROCESSORS

Sean C. Martin

B.Sc.(Dunelm).

Department of Applied Physics and Electronics

University of Durham

September 1980

A thesis submitted for the degree of

Doctor of Philosophy of the University of Durham

ABSTRACT

NUMBER THEORETIC TRANSFORM IMPLEMENTATION USING MICROPROCESSORS

Sean C. Martin

Since 1974 considerable interest has been shown in the literature in the topic of number theoretic transforms. These transforms provide an efficient integer processing technique for convolution. Microprocessors are suited to integer processing particularly for applications where the required processing load is small. It was therefore a natural step to investigate and tailor the properties of number theoretic transforms to the capabilities of microprocessors to provide cheap and compact processors using efficient signal processing algorithms.

It was found that efficient number theoretic transforms could be defined using the Modulus M = 65521 and this is especially convenient for a microprocessor implementation. Relevant aspects of modular arithmetic are investigated. The techniques developed are extended to allow for complex signal processing.

In conclusion it is shown that number theoretic transforms can be used to encode and decode Reed-Soloman error correcting codes.

ACKNOWLEDGEMENTS

I would like to thank Dr. B. J. Stanier for his constant supervision of the work conducted during this project, and I would also like to thank Mrs. J. Doherty for advice on the presentation of this thesis. I am grateful to the Wolfson Microelectronics Institute at the University of Edinburgh for the use of wordprocessing facilities.

This work was supported by a University of Durham research scholarship.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

With recent advances in solid state technology digital signal processing is becoming more and more important as a substitute for analogue processing in various areas of communications, control engineering and general signal processing. Digital components are more reliable, have a lower failure rate, can achieve arbitary accuracy and in certain applications consume less power and occupy less space than analogue components. But, in order to take full advantage of this, efficient signal processing algorithms must be developed to suit the capabilities of the available hardware. Microprocessors combine all the assets described to provide cheap compact processors for applications where computation speed is not critical. The aim of this work has been to adapt recently developed signal processing algorithms to the capabilities of microprocessors.

Convolution is the basic operation of signal processing; however a direct implementation of convolution is inefficient. The Fourier transform possesses a special property known as the cyclic convolution property (CCP). This attribute identifies multiplication of frequency domain signals with the convolution of signals in the time domain. It is well known that the convolution of two signals may be achieved by taking the Fourier transform of both signals and then taking the inverse Fourier transform of the product of the frequency domain signals. With the advent of the fast Fourier transform (FFT) algorithm, convolution may now be performed considerably more efficiently by Fourier transform techniques.

The Fourier transform is defined over the complex domain and during its computation it requires multiplications by complex irrational roots of unity. With any finite precision these coefficients cannot be represented accurately and so the Fourier transform is subject to accumulative round off error. In signal processing applications data is often sampled using analogue to digital convertors and therefore without loss of generality such signals can be considered to be integer with some upper bound and scale factor applied. One may therefore consider signals appertaining to the real world to be essentially integer in character. It follows that in principle, if convolutional signal processing is computed via the Fourier transform then complex arithmetic has to be employed even when the data is often purely integer in nature. It is therefore a natural step to consider performing signal processing by an integer technique.

Algebraic studies of the Fourier transform reveal that certain structural attributes are responsible for it possessing the cyclic convolution property. It is possible to define transforms with these attributes over finite rings of integers. These transforms are collectively known as number theoretic transforms (NTTs), and it will be seen that these transforms have some advantages over the Fourier transform. Modular integer arithmetic is more attractive than complex floating point arithmetic for simple computers. This arithmetic is also exact since no round off error occurs during the computation of the transform, and it is for these reasons that much interest has been shown in number theoretic transforms.

Of late microprocessors have become available providing small and cheap processors. These processors do not offer the sophisticated facilities associated with more expensive machines and so it seems that they are more suited to integer arithmetic rather than complex arithmetic. It was therefore considered worthwhile to try and implement number theoretic transforms using microprocessor systems.

There are many excellent references on number theoretic transforms e.g. (1)-(5), which introduce the fundamental concepts of the topic. However one may find an introduction to number theory enlightening, for example Ore (68). The reader's attention is drawn particularly to a fairly recent publication by McClellan and Rader (122) entitled 'Number theory in Digital Signal Processing'. This book provides a thorough overview of the topic of number theoretic transforms. Although not all of the references have been directly referred to in the text they have been included to provide a fairly comprehensive list of related work.

Agarwal and Burrus (3),(5) have shown that in order that the transform support the cyclic convolution property certain structural constraints limit practical choices of number theoretic transform. A finite field of M elements (Z_M) will support only certain values of transform length (N), and for a given M and N an element α of order N must be found in Z_M. The transform and its inverse are defined by the following relations:

$$\begin{aligned} X(j) &= \sum_{i=0}^{N-1} x(i)\,\alpha^{ij} \quad \text{mod } M \qquad j \in (0,N-1) \\ x(i) &= N^{-1} \sum_{j=0}^{N-1} X(j)\,\alpha^{-ij} \quad \text{mod } M \qquad i \in (0,N-1) \end{aligned} \qquad (1.1)$$

The particular choices of M, N and α affect the ease of computation of the transform. These constraints are summarized as follows:

(a) N must divide O(M), where O(M) is defined to be the greatest common divisor of the set of prime divisors $(p_i - 1)$ of M i.e. where $O(M) = \text{g.c.d.}\ (p_i - 1)$

(b) α must be an element of order N i.e. $\alpha^N = 1 \bmod M$ and $\alpha^r \neq 1 \bmod M \quad r \in (1, N-1)$.

(c) N^{-1}, a multiplicative inverse of N must exist in the ring Z_M.

(d) N should be well factored for fast transform algorithms to exist.

(e) To facilitate fast and simple arithmetic mod M, M must have a simple binary representation, and to facilitate fast multiplication by powers of α, α must also have a simple binary representation.

Agarwal and Burrus have shown that constraint (a) is equivalent to constraints (b) and (c). Constraint (a) is in a useful form for finding a suitable modulus for a given transform length, while constraints (b) and (c) state the conditions which the transform parameters α and N^{-1} must satisfy once M and N have been determined. For this reason both sets of constraints are presented.

Let us consider $M = 2^k$. This modulus does have a simple binary pattern but obviously it has a prime factor of two and so $O(2^k) = 1$ and hence the maximum transform length is also one.

For $M = 2^k-1$ where k is composite, then let $k = PQ$ where P is prime. It can be seen that $2^P - 1$ divides $2^{PQ} - 1$ and hence the maximum transform length will be governed by the maximum possible for $M = 2^P - 1$. Therefore only the primes P need be considered and such numbers $M = 2^P - 1$ are known as Mersenne numbers. It is this property that gives rise to the study of Mersenne number transforms.

For $M = 2^k + 1$ where k is odd, then 3 divides $2^k + 1$ and so the maximum transform length is 2; therefore we need only consider even k. Let k be $s2^t$ where s is odd, then $2^{2^t} + 1$ divides $2^{s2^t} + 1$ and the length of the possible transform will be governed by the length possible for $M = 2^{2^t} + 1$. Numbers of this form are known as Fermat numbers and this gives rise to the study of Fermat number transforms.

Both these transforms have moduli and α with simple binary representations which facilitate fast and simple arithmetic.

However the Mersenne number transforms do not support highly factored transform lengths and so fast transform algorithms are not available for this class of number theoretic transform.

The Fermat number transform does however support radix 2 FFT algorithms and hence this appears to be an attractive transform. The nature of this transform is examined in more detail in chapter 2 where it is shown that the transform does suffer from certain limitations which make it unsuitable for a microprocessor implementation. It is also possible to perform complex convolutions using Fermat number transforms using a technique presented by Nussbaumer. In order to define the nomenclature this technique is presented in chapter 2 as it will later be generalized and used widely in later chapters.

Number theoretic transforms are alike in structure to the Fourier transform, and hence in principle any fast Fourier transform algorithm can be applied to number theoretic transforms. Therefore number theoretic transforms can be computed using standard fast Fourier transform (FFT) algorithms, which are most efficient for transform lengths which are a power of 2. In 1976 Winograd (113),(114) and (101) proposed a new class of Fourier transform algorithms (WFTA) and these too may be adapted to derive comparable fast algorithms for number theoretic transforms. Winograd applied field theory to derive short transform length algorithms in the range 2 to 16. He has proposed a technique for combining these algorithms in a nested structure to derive efficient algorithms for greater transform lengths. By employing this technique it is possible to derive transform algorithms for a wider variety of transform lengths than would be available with standard radix 2 FFT algorithms.

In a comparison between the Winograd algorithm and the FFT, one finds that the number of multiplications required to compute the transform by the Winograd algorithm is reduced below that required for the FFT while the number of additions required is comparable. This reduction of arithmetic load is gained at the sacrifice of algorithm complexity.

There is dispute in the literature as to the relative merits of the WFTA and the FFT (101),(49). However it is certain that the WFTA does provide algorithms which are as efficient as comparable FFT algorithms with the primary advantage that algorithms for a greater variety of transform lengths are obtainable. It is this last point which enabled the development of number theoretic transforms which are suited to a microprocessor implementation and this work is covered in chapter 3.

Preliminary investigations showed that the Fermat number transform, which is best implemented using a radix 2 FFT algorithm, was not particularly suited to a microprocessor system. Since this class of transform was the optimum choice for the FFT other classes of transform were considered with a view to utilizing the greater variety of transform available with Winograd transform algorithms. It was found that the multiplication coefficients required for these algorithms do not have simple binary patterns unlike the corresponding coefficients for Fermat number transforms. However microprocessors are becoming available with fast multiply instructions and for those that do not, hardware multiply chips are available, and this allows fast general integer multiplications. This property can be seen to relax constraint (e) that the multiplication coefficients required for the transform computation need have a simple binary pattern. For such processors the fact that the Winograd multiplication coefficients do not have simple binary patterns is not a serious limitation and this allows wider classes of number theoretic transforms.

A search was made for choices of modulus which would be suited to Winograd algorithms. Since most microprocessors can efficiently accomodate only 16 bit arithmetic, without excessive software penalties, the search was conducted below 2^{16}. It was found that the choice of M = 65521 is optimum.

The technique proposed by Nussbaumer was adapted to allow complex convolutions to be performed using these number theoretic transforms. In some cases a given modulus will provide insufficient dynamic range and a larger modulus should be chosen. A direct implementation of such a scheme would require arithmetic with greater precision and this seems unattractive. The Chinese remainder theorem may be employed to circumvent this problem. If the results of an arithmetic operation are evaluated with respect to two or more relatively prime moduli then the result may be determined with respect to the modulus which is the product of those moduli. This approach is particularly suited to a parallel processing technique. A search was made for moduli which will combine with M = 65521 over specific Winograd transform lengths. The material relating to deriving number theoretic transforms suited to a microprocessor implementation is outlined in chapter 3.

In chapter 4 certain aspects of modular arithmetic are discussed. The technique of filter design for finite field arithmetic is examined more closely. As has been previously noted, in cases when the dynamic range of a given modulus is insufficient to meet the filter design constraints, then a larger modulus should be chosen. The mechanism by which the Chinese remainder theorem allows recombination of results wirh respect to two sub moduli is examined. Preliminary investigations showed that modular integer arithmetic is considerably slower than normal integer arithmetic and a hardware modular multiplier was designed in order to reduce the computational load upon the processor.

Reed, Truong et al have considered defining transforms over quadratic finite fields. Arithmetic over these fields resembles arithmetic defined in the complex domain and using this type of transform it is possible to convolve complex sequences directly. In the manner previously outlined for simple number theoretic transforms, a search was conducted for a suitable modulus to support Winograd algorithms for complex transforms. It was found that the modulus M = 65519 is an optimum choice.

It was observed that the two optimum choices for modulus, 65521 and 65519, form a prime pair. This result was investigated more closely and generalized for other bit lengths. Aspects of complex transforms relevant to a microprocessor implementation are discussed in chapter 5.

Recent interest has been shown in employing number theoretic transforms to encode and decode a special class of error correcting code, known as a Reed-Soloman code. This technique was adapted to use the algorithms derived earlier. This work is discussed in chapter 6.

Finally in chapter 7 some areas of further investigation are suggested which did not receive enough attention in this thesis.

It is shown that microprocessor systems can be used effectively to provide the hardware for small scale convolution requirements and therefore such systems can be used wherever digital convolution is required. The range of possible applications is broad, covering many areas, from the computation of auto and cross correlation and power spectra, and non recursive and recursive digital signal processing; to obtaining the solution of difference equations.

CHAPTER 2

FERMAT NUMBER TRANSFORMS

Agarwal and Burrus (1) have shown that exact cyclic convolution can be performed by using number theoretic transforms defined over rings of integers modulo Fermat numbers F_t given by $F_t = 2^{2^t} + 1$. Such transforms can be computed without requiring any multiplications to be performed but requiring only repeated bit shifts, additions and subtractions.

These transforms also support radix 2 FFT algorithms and hence at first sight they would appear to be attractive for signal processing by transform techniques.

The Fermat number transform is examined in more detail in this chapter, and when this transform is considered for a microprocessor implementation two limitations become apparent. It is found that a binary representation problem exists and it is felt that an efficient microprocessor implementation would generate computational errors which would be likely to seriously affect complete data blocks. It is primarily for this reason that Fermat numbers were considered unsuitable moduli for a microprocessor implementation.

It is also shown that the transforms lengths available with Fermat number transforms are limited and therefore other classes of number theoretic transform were considered for a microprocessor implementation.

The chapter concludes with a section describing work by Nussbaumer (51) which enables complex convolutions to be computed via Fermat transforms. It was felt necessary to include this material inorder to define the nomenclature and to express the subleties of this technique in a manner which aids further development in chapter 3.

2.1 FERMAT NUMBER TRANSFORMS (FNTs)

Numbers of the form $F_t = 2^b + 1$, $b = 2^t$ are known as Fermat numbers and F_t is known as the t^{th} Fermat number. Transforms of length N can be defined using Fermat moduli provided an element α of order N and an element N^{-1} can be found. The choice of the modulus F_t leads to the observation that

$$2^b = -1 \qquad \text{mod } F_t \text{ where } b = 2^t$$

or

$$2^{2b} = 1 \qquad \text{mod } F_t$$

and it can be seen that the element 2 is of order 2^{t+1} in the ring F_t. For prime Fermat numbers it can be seen that

$$O(F_t) = 2^{2^t} \qquad (2.1.1)$$

and so these moduli are particularly suited to radix 2 FFT algorithms, for which the element 2 provides a suitable α with a simple binary pattern, supporting a transform length which is proportional to the wordlength employed.

Having found that the first four Fermat numbers were prime, Fermat conjectured that all such numbers would be likewise, however no known Fermat prime above F_4 exists. It is now well known (1) that any prime factors of composite Fermat numbers are of the form $(k2^{t+2} + 1)$ and therefore (2^{t+2}) divides $O(F_t)$. In particular $O(F_t) = 2^{t+2}$ for the Fermat numbers F_5 and F_6. It can therefore be seen that Fermat numbers larger than F_4 are also suited to radix 2 FFT algorithms.

In the light of this discussion it can be seen that FNTs support radix 2 FFT algorithms with the choice of $\alpha = 2$ and this is a desirable situation. For Fermat moduli the element 2 is of order 2^{t+1} and so the transform length for such simple transform algorithms is proportional to the wordlength employed. This is a limitation of such transforms.

The element $\surd 2$ is of order 2^{t+2} where the symbol $\surd 2$ denotes the element for which $(\surd 2)^2 = 2$. With the choice of $\alpha = \surd 2$ the transform length can be doubled over that available with $\alpha = 2$. Multiplication by odd powers of $\surd 2$ are required in only one half of one pass of the fast algorithm and in most cases this is not too serious a penalty.

A selection of Fermat moduli, permissible transform lengths and corresponding αs are shown in table 2.1.

In computing the FNT arithmetic is perfomed modulo $2^b + 1$. In this arithmetic the allowed integers are 0 .... 2^b. However using b bit arithmetic only integers in the range 0 .... $2^b - 1$ can unambigously be represented. Therefore there exists a problem in the representation of the element 2^b which is itself congruent to -1. For example consider the case of $F_4 = 65537$. 17 bits are required for unambigous representation of all the valid elements of Z_{F_4}. However the most significant bit is required only to represent the element (-1). Therefore great redundancy is incurred in the use of 17 bit arithmetic while 16 bit arithmetic is just inadequate.

Using b bit arithmetic truncation errors will occur whenever the element -1 is required to be represented. It is worthwhile to distinguish between two clearly distinct occasions when this representation error may occur. During initial analogue to digital conversion the element -1 may be required and the truncation will generate the representation -2 or 0. This case is not serious since this can be seen to be just an abnormally large quantisation error and in any case this situation can be avoided by careful choice of the input quantisation levels. Serious errors will arise, if during processing the representation error occures when data values do not

TABLE 2.1

PARAMETERS FOR A SELECTION OF FNTs

t	b	F_t	N $\alpha=2$	N $\alpha=\sqrt{2}$	N_{max}	α for N_{max}
3	8	2^8+1	16	32	256	3
4	16	$2^{16}+1$	32	64	65536	3
5	32	$2^{32}+1$	64	128	128	2
6	64	$2^{64}+1$	128	256	256	2

TABLE 2.2

PROBABILITY OF ERROR FOR A LENGTH N FNT CONVOLUTION

t	b	F_t	N $\alpha=2$	P_e
4	16	$2^{16}+1$	32	$8.3.10^{-3}$
5	32	$2^{32}+1$	64	$2.9.10^{-7}$
6	64	$2^{64}+1$	128	$1.6.10^{-16}$

have their usual physical significance; but rather they represent elements of Z_{F_4}, and in such cases a single error in any data value can seriously affect the entire data block. This is potentially an area of concern.

During processing itself the carry flag can be used to provide the extra bit and so the result of each arithmetic operation would be exact, however unless some provision were made for (b + 1) bit memory then the possibility of erroneous results is introduced.

If the data are uncorrelated then the probability of this error occurring is approximatly 2^{-b} per memory write operation. Based upon this assumption the probability of any error (p_e) occuring in a length N FNT convolution has been estimated and is shown in table 2.2.

For F_5 and F_6 this probability is small and may be acceptable, however for F_4 the probability of errror is unacceptably high. This limitation is a serious shortcoming of FNTs particularly for F_4 which is the Fermat modulus perhaps best suited for a microprocessor implementation.

Agarwal and Burrus (1) have compared the performance and hardware requirements of the FNT with a fixed point DFT technique with a view for an implementation on an IBM 370. They observe that the bit length required to obtain an accurate result using the FNT is roughly twice that required for the DFT. However for the DFT every data point requires a real and an imaginary word to represent the complex values and so the hardware requirement is comparable for the two techniques. They have shown that the complexity of corresponding addition and subtraction operations for the two techniques is also comparable. The DFT employs multiplications by powers of $W = e^{-j2\pi/N}$ and this is a comparatively slow operation and they observe that the bit shift and subtract in carry procedure for the FNT is faster for an IBM 370 machine. FNT algorithms have two other advantages; they do not require storage of powers of W as is necessary for an efficient DFT algorithm, and secondly they are not subject to accumulative round off error and so produce exact results. The only source of error is the initial A/D quantisation. They report that for the IBM 370 the FNT is faster than a corresponding DFT by a factor of up to 5.

As has previously been observed the maximum transform lengths for efficient FNTs are limited and for longer convolution lengths multidimensional techniques have to be employed. For these long convolution lengths the multidimensional techniques (2) available at the time of this comparison (1974) were not efficient and so the comparison was degraded. Agarwal and Cooley have subsequently published more efficient technniques specifically for this application (7).

McClellan has constructed a hardware realization of an FNT processor for radar signal processing (47). For real signal processing and particularly with modest convolution lengths (e.g. 64) he has shown that the FNT requires less hardware than the corresponding DFT pipeline processor. It can readily be seen that such a hardware realization of a Fermat number transform would be most efficient if implemented with a radix 2 FFT algorithm and the choice $\alpha=2$. This is enhanced by the use of subtract in carry hardware. However preliminary invesitgations showed that such an approach would not be efficient for a microprocessor implementation.

Melhuish (48) has used a minicomputer to perform FNTs in a signal processing application, with results, which although promising indicate that the optimum transform length relation with the wordlength is a limitation of such transforms.

It has been the purpose of this section to present the Fermat number transform as a convolutional technique which is particularly suited to radix 2 FFT algorithm requiring only bit shifts and additions for its computation. When compared with the DFT techniques the FNT provides a fast and potentially efficient scheme particularly as the general complex multiplications required for the DFT are no longer needed.

However two limitations have become apparent. The optimum transform length is restricted and there is a problem of unambiguous representation of data. The latter can lead to serious errors affecting an entire data block. This is a greater problem with short wordlengths and this has been considered to be an unnaceptable technique for the wordlength of 16 bits which is that best suited to an efficient microprocessor implementation.

2.2 COMPLEX CONVOLUTION VIA FERMAT NUMBER TRANSFORMS

Nussbaumer has proposed a novel technique for performing complex convolutions over rings modulo Fermat numbers. The structure of such rings does not allow the concepts of 'real', 'imaginary' and 'complex' to be defined in the normal way and hence this is an intruiging technique.

He observed that with arithmetic over such rings one may find an element j for which $j^2 = -1$. It is this property which gives a strong structural relation with the familiar complex field and by utilizing this property one may impart some complex nature to such rings, and by so doing it becomes possible to perform complex convolutions by Fermat number transforms.

In section 3.4 this technique is later generalized for moduli other than Fermat number moduli and this permits complex convolutions to be performed with the number theoretic transforms which are derived in chapter 3 for a microprocessor implementation.

For Fermat number moduli it has already been observed that

$$2^b = -1 \quad \bmod F_t$$

and so $$(2^{b/2})^2 = -1 \quad \bmod F_t$$

The element $2^{b/2}$ we will denote as the element 'j' since it can be seen that $j^2 = -1$. Therefore this element has the mathematical properties of the element j in the complex domain. It is this relation which allows complex convolutions to be performed efficiently with Fermat number transforms. The mathematical nature of the number theoretic 'j' will be examined in section 3.4.

Let $$a_i = a_i' + j\, a_i'' \tag{2.2.1}$$

Where a is a complex sequence

a_i' = real part of a_i

a_i'' = imaginary part of a_i

j is an element such that $j^2 = -1 \mod M$

Consider performing the convolution $y_i = x_i * h_i$ by a direct technique

$$\begin{aligned} y_i' &= x_i' * h_i' - x_i'' * h_i'' \\ y_i'' &= x_i' * h_i'' + x_i'' * h_i' \end{aligned} \tag{2.2.2}$$

and this can be seen to require 4 length N convolutions and 2N additions. By using Golub's algorithm (57) this may be reduced to 3 length N convolutions and 5N additions.

Nussbaumer (57) has proposed a new technique for finite field arithmetic. First form inphase and quadrature components of the sequences x and h:

$$a_{xi} = x'_i + j\,x''_i, \quad a_{hi} = h'_i + j\,h''_i \quad \text{mod } M \qquad (2.2.3)$$

$$b_{xi} = x'_i - j\,x''_i, \quad b_{hi} = h'_i - j\,h''_i \quad \text{mod } M \qquad (2.2.4)$$

and it follows that

$$y'_i + j\,y''_i = a_{xi} * a_{hi} \quad \text{mod } M \qquad (2.2.5)$$

$$y'_i - j\,y''_i = b_{xi} * b_{hi} \quad \text{mod } M \qquad (2.2.6)$$

and so

$$\begin{aligned} y'_i &= 2^{-1} \,[\, a_{xi} * a_{hi} + b_{xi} * b_{hi} \,] \\ y''_i &= (2j)^{-1} \,[\, a_{xi} * a_{hi} - b_{xi} * b_{hi} \,] \end{aligned} \qquad (2.2.7)$$

This technique can be seen to require only two length N convolutions 4N multiplications and 6N additions. Therefore the number of convolutions required has been reduced from 3 to 2, and since the bulk of the computational load is incurred by the convolution operator it can be seen that this technique proffers computational advantages. These convolutions may of course be computed by transform techniques.

For FNTs the elements 2^{-1} and $(2j)^{-1}$ can be expressed as positive powers of the element 2 and so fast multiplication techniques can be applied.

It has been the purpose of this section to introduce the technique whereby complex convolutions can be efficiently performed by number theoretic transform techniques by utilizing a structural property of the rings over which the transforms are defined.

CHAPTER 3

MICROPROCESSOR IMPLEMEMTATION OF NUMBER THEORETIC TRANSFORMS

Number theoretic transforms can be computed requiring only bit shifts and additions, however it has been found that such number theoretic transforms are not particularly suited to a microprocessor implementation.

Fast multiply instructions available on some microprocessors or the use of external multipliers relax the basic constraints on the choice of a particular number theoretic transform, and so other classes of NTT become practicable.

A search was therefore conducted for suitable moduli and it was found that the choice of M = 65521 seemed optimal. It is shown how complex convolutions may be performed by adapting Nussbaumer's technique. The chapter concludes with a section describing a selection of moduli which will combine using the Chinese remainder theorem with M = 65521 over specific transform lengths to obtain increased dynamic range.

3.1 A REVIEW OF NUMBER THEORETIC TRANSFORMS

Agarwal, Burrus, Rader and others have shown that error free exact convolutions can be performed using number theoretic transforms. It is possible for such transforms to be defined so that few or no generalized multiplications are required for their computation. These transforms can be performed requiring only bit shifts and additions. Both the Fermat number transform and the Mersenne number transform are such transforms. However the transform lengths corresponding to Mersenne number transforms are not well factored, and as can readily be seen it is desirable for the transform lengths which are supported to be well factored for fast transform algorithms to exist. Agarwal and Burrus (3) have shown that the following constraints limit practical choices of number theoretic transforms:

(a) N must divide O(M), where O(M) is defined to be the greatest common divisor of the set of prime divisors $(p_i - 1)$ of M i.e. where $O(M) = \text{g.c.d.} (p_i - 1)$

(b) α must be an element of order N i.e. $\alpha^N = 1 \bmod M$ and $\alpha^r \neq 1 \bmod M \quad r \in (1, N-1)$.

(c) N^{-1}, a multiplicative inverse of N must exist in the ring Z_M.

(d) N should be well factored for fast transform algorithms to exist.

(e) To facilitate fast and simple arithmetic mod M, M must have a simple binary representation, and to facilitate fast multiplication by powers of α, α must also have a simple binary representation.

Agarwal and Burrus have shown that constraint (a) is equivalent to constraints (b) and (c). Constraint (a) is in a useful form for finding a suitable modulus for a given transform length, while constraints (b) and (c) state the conditions which the transform parameters α and N^{-1} must satisfy once M and N have been determined. For this reason both sets of constraints are presented.

From all these aspects the Fermat number transform provides an optimum number theoretic transform but as previously discussed this transform suffers from two limitations. For a simple Fermat number theoretic transform the transform length is proportional to the wordlength employed. For a microprocessor implementation the optimum wordlength is 16 bits and as has been seen the corresponding Fermat number transform suffers from a representation problem which is potentially serious for this choice of wordlength.

In conclusion it can be said that for a microprocessor implementation there is no single class of efficient error free number theoretic transform which can be computed requiring only bit shifts and additions.

3.2 A SEARCH FOR NUMBER THEORETIC TRANSFORMS SUITABLE FOR A SIMPLE MICROPROCESSOR IMPLEMENTATION

The concept of the Winograd Fourier transform algorithm has been introduced. Unlike the Fermat number transform the multiplication coefficients for this class of transform algorithm do not have a simple binary representation. Therefore the Winograd algorithms would not appear to be attractive for a microprocessor implementation. However microprocessors are becoming available with fast multiply instructions and for those that do not have this facility, fast hardware multiplier chips are available (117). These trends allow fast generalized multiplications and make such operations competetive with repeated bit shift and subtract in carry operations such as are required with Fermat number transforms. Therefore for a microprocessor implementation constraint (e) may be waived allowing binary non-simple moduli and αs and so many more number theoretic transforms become practicable.

There are two main parts for a search for number theoretic transforms suitable for a microprocessor implementation. Having decided upon a transform length, a suitable modulus must be found to satisfy $N \mid O(M)$ and secondly an element α in Z_M of order N must be found. Work by Bailey (9) based upon these ideas was performed concurrently.

It was therefore decided to implement a computer search for suitable combinations of N, M, and α. It is apparent that radix 2 FFT algorithms have already been well dealt with in the literature in this context and so Winograd transform lengths were considered. Silverman (101) has presented small-N Winograd algorithms for the following transform lengths (2, 4, 8, 16), (3, 9), 5 and 7. In principle it is possible for such algorithms to be derived for other transform lengths e.g. 11 and 13 however these do not promise to be as efficient and additionally they have not yet been described in the literature. Therefore for the existing transform algorithms it can be seen that any composite transform length will divide 5040 (which is the product of 16 x 9 x 5 x 7). Hence any modulus which supports a transform length of 5040 will also support any of the other Winograd transform lengths composed of Silverman's algorithms.

Bailey (9) suggested searching over prime p until the condition $N \mid (p - 1)$ was satisfied. However the approach which was adopted was to search over odd values of M until the condition $N \mid O(M)$ was met. This is a more general condition allowing composite moduli to be used. Since data is handled most efficiently in full bytes this search was conducted from 2^{16} downwards for moduli which would support a transform length of 5040.

This search quickly revealed that the choice of modulus M = 65521 was optimum for two reasons:

(a) O(65521) = 13 x 5040 and so this modulus will support any Winograd transform algorithm

(b) 65521 is very close to 2^{16} and so little redundancy is incurred in the use of 16 bit arithmetic. (The redundancy is 0.023%).

By way of interest it may be noted that 65521 is the first prime below 2^{16}.

In order to implement number theoretic transforms with this choice of modulus an element of order 5040 must be determined. Elements of orders which divide 5040 can easily be determined from such an element. Bailey (9) suggests a technique whereby if an element of order N is required then a search over the elements of Z_M should be conducted until an element is found of an order which is equal to N or a multiple of N. However a search procedure will be outlined to find a primitive element in Z_M. The order of such an element is given by Euler's function $\emptyset(M)$ (see (68)). Much more information is gained from a primitive element than from an element of arbitary order, since elements of any order which divide $\emptyset(M)$ can quickly be derived from the primitive element.

Euler's theorem (68) states that for every element b in the ring Z_M which is relatively prime to M the following relation holds:

$$b^{\emptyset(M)} = 1 \qquad \text{mod } M \qquad (3.2.1)$$

It is easily shown that the order of b must divide $\emptyset(M)$ and it is this point which gives the key to an efficient search technique. First find $\emptyset(M)$ and reduce it to its prime factored form.

$$\emptyset(M) \neq p_1^{r1}.\ p_2^{r2}.\ p_3^{r3}....\ p_l^{rl} \qquad (3.2.2)$$

Then for any non primitive element b, the order of b must divide one of the following:

$$\frac{\emptyset(M)}{p_1} \ , \quad \frac{\emptyset(M)}{p_2} \ , \quad \frac{\emptyset(M)}{p_3} \ ,, \quad \frac{\emptyset(M)}{p_l}$$

Therefore if for an element b the following relations hold

$$b^{\emptyset(M)/p_i} = 1 \qquad \text{mod } M \text{ i e } (1,1) \qquad (3.2.3)$$

and provided b is coprime with M then b must be a primitive element of Z_M. From such an element b_p an element α_N of order N can quickly be found by:

$$\alpha_N = b_p^{\emptyset(M)/N} \qquad (3.2.4)$$

This method is in principle similar to that presented in reference (122).

The elements of Z_M can be scanned from 2 ⟶ M-2 for a suitable primitive element. The computer search time for finding a primitive element in Z_M where M is of the order of 2^{16} is not prohibitively long by conventional techniques. However as will be shown in a later chapter the corresponding search time for a suitable element for complex transforms is prohibitively long and in such cases use must be made of an efficient technique such as that described.

In summary, this section presents the ideas behind trying to derive number theoretic transforms suitable for a microprocessor implementation. The availability of fast multiply instructions on some microprocessors relax the basic constraints upon the choice of a particular number theoretic transform. Since radix 2 FFT algorithms have already been well exploited, and it has been shown that these tend to prefer Fermat number transforms, which for the purposes of a microprocessor are impractible; Winograd transform algorithms were in principle adopted for a proposed transform technique.

It was found that the choice of M = 65521 was optimum and the corresponding transform algorithm development will be described in the next section.

3.3 WINOGRAD ALGORITHM DEVELOPMENT

Algorithm development was conducted in FORTRAN on an IBM 370 where the mainframe support facilities were found to be benificial. Before any specific algorithm development could be attempted, certain simple modular arithmetic routines had to be defined. A selection of these are shown in Appendix A.

In the search for a suitable modulus to satisfy the condition $5040 \mid O(M)$ a routine had to be defined to derive the value of $O(M)$. This function OH(M) employs an efficient factoring subroutine FACTOR and these two allowed the basic modulus search to be conducted. Having found a suitable modulus, modular arithmetic functions had to be defined. The function MODO reduces an arithmetic value mod M, where M has previously been declared by the use of subroutine MSET.

It was found in many cases that expressions of the form a^b mod M were required to be evaluated. Direct evaluation of the expression would often cause integer overflow and so a different technique was implemented. Let b be represented by

$$b = \sum_{i=0}^{lim} b_i 2^i \qquad b_i \in (0,1) \quad 0 < b < 2^{lim+1} \qquad (3.3.1)$$

and evaluate the expressions

$$P_i = (P_{i-1})^2 \mod M, \qquad P_0 = 1, \qquad i \in (1,lim) \qquad (3.3.2)$$

then

$$a^b = \sum_{i=0}^{lim} E\,(b_i P_i) \mod M \qquad (3.3.3)$$

where $E(0) = 1$ and $E(x) = x$ otherwise.

These relations provide an efficient procedure for evaluating a^b mod M and the function EXPM achieves this result.

A procedure named INV was written to derive the multiplicative inverse of a number x by an application of Euler's theorem:

$$x^{-1} = x^{\emptyset(M)-1} \qquad (3.3.4)$$

and this function employs the routine EXPM.

The short N algorithms presented by Silverman (101) express the multiplication coefficients in terms of the trigonometrical functions COSINE and SINE. In modular arithmetic the concepts of magnitude and phase are not defined and so the meaning of these functions has to be reinterpreted.

$$u = \cos u + j \sin u \qquad (3.3.5)$$

$$u^{-1} = \cos u - j \sin u \qquad (3.3.6)$$

and so

$$\cos u = 2^{-1} (u + u^{-1}) \qquad (3.3.7)$$

$$\sin u = (2j)^{-1} (u - u^{-1}) \qquad (3.3.8)$$

These relations allow the COSINE and SINE functions to be interpreted in the number theoretic sense. The function MCOS generates expressions for cos iu and sin iu, $i \in (1, 4)$ and this allows the Winograd algorithms to be developed.

Since any specific-N transform algorithm would be computationally more efficient than a general N algorithm, various algorithms were derived for specific transform lengths. However it was found that a general N program was an invaluable tool for such algorithm development since during its execution this program derives the multiplication coefficients, the pointers into workspace arrays and the permutation sequences required for specific N algorithms.

A modified version of the program was written which when executed would automatically derive and print out all such useful parameters. Although not as powerful, this technique is based on the same principle of the 'autogen' software developed by Morris (49) for his comparison between the WFTA and FFT algorithms. With more design effort the general N program could have been made completely autogenerative.

Certain points are of interest concerning the general N program and the subsequent specific N algorithms.

(a) Since the ultimate aim is to derive algorithms for a microprocessor environment where memory workspace should be minimized, the suggestions given by Silverman (101) for reducing memory requirement were heeded.

(b) Arithmetic for the Fourier transform would generally be complex because the Fourier transform is defined in the complex domain. However the algorithms presented by Silverman (101) involve only purely real or purely imaginary data, and so in the inner stages of their computation, certain savings are made by keeping flags to denote the data type. The concepts of real, imaginary and complex do not strictly apply in the number theoretic sense, and so it was not necessary for such flags to be kept.

(c) We have already shown how Silverman's trigonometrical expressions may be interpreted in the number theoretic sense. For the inverse transform the normalizing factor N^{-1} may be incorporated into the multiplication coefficients and by so doing the forward and inverse transforms are made equally efficient. These two points have also been made by Bailey (9).

The general N program was run on an IBM 370 and the results of convolutions performed by such number theoretic transforms with modulus M = 65521 were compared with reference techniques. For short convolution lengths the reference technique was a direct integer procedure and for longer lengths a Fourier transform technique was employed. The Fourier transform subroutine used was from the NAG library.

The test convolutions were of two types. Preliminary tests were taken with cross convolutions of simple sequences of sinusiods, square waves and exponential functions. The later tests involved sampling a laboratory signal and convolving it with the impluse response of a band stop filter. In all cases the NTT convolutions gave results in exact agreement with the direct convolution technique. The Fourier transform technique produces 'real' results subject to round off error and within the limits of accuracy of such a technique the results of the NTT convolutions were also in exact agreement.

Specific transform algorithms were written using the autogen technique described above for the transform lengths of 60 and 240. These algorithms for M = 65521 are presented in Appendix B. The length 60 algorithm was transcribed into assembler code for an Intel 8080 microprocessor using the FORTH programming technique (118). The source code is presented in Appendix C.

The algorithms have been designed for implementation on a microprocessor with either a fast multiply instruction or with added hardware multipliers. However the development system available for the Intel 8080 had neither facility and the subsequent execution speed could not be expected to be particularly fast. However this prototype algorithm gave results in exact agreement with the mainframe computer. This algorithm was employed in a band stop filtering application.

3.4 EXTENSION TO COMPLEX FILTERING

Vanwormhoudt (109) has shown that there exist two main classes of prime moduli. Since all moduli that are useful for number theoretic transforms are odd the two main classes are those for which M = 1 mod 4 (type A) and those for which M = 3 mod 4 (type B).

Let us consider a prime modulus (M_A) of type A. Then by Euler's theorem

$$\emptyset(M_A) \;=\; M_A - 1 \;=\; 0 \quad \text{mod } 4 \qquad (3.4.1)$$

and so $$4 \mid \emptyset(M_A)$$

This implies that an element of order 4 exists and such an element (j) will satisfy $j^2 = -1$ mod M_A.

For a modulus (M_B) of type B we see that

$$\emptyset(M_B) \;=\; M_A - 1 \;=\; 2 \quad \text{mod } 4 \qquad (3.4.2)$$

and so $$4 \nmid \emptyset(M_A)$$

and this implies that no such element of order 4 exists in Z_{M_B}.

It is this point which provides a basic structural difference between these two classes of moduli. It is well known that (109) the Chinese remainder theorem provides a ring isomorphism between a set of type A moduli and the ring Z_M where M is the product of these moduli, and so we may generalize these results for composite moduli. A composite modulus whose prime factors are all of type A will also possess an element j such that $j^2 = -1$ mod M, whereas if any one factor is of type B then no such element will exist.

In chapter 2 it was shown how the approach developed by Nussbaumer (57) allowed complex convolutions to be performed using Fermat number transforms using the element $j = 2^{b/2}$ for which

$j^2 = -1 \mod F_t$. Reddy and Reddy (83) have adapted this idea and shown that a similar procedure allows this technique to be used with other moduli.

It is proposed that Nussbaumer's algorithm may be employed with all moduli for which an element 'j' exists, and as has been shown, such an element will exist when the modulus is either prime and of type A, or when composite all the prime factors are of type A. the procedure by which complex convolutions may be performed using number theoretic transforms is presented:

Let $a_i = a'_i + j\,a''_i$ (3.4.3)

Where a is a complex sequence

a'_i = real part of a_i

a''_i = imaginary part of a_i

j is an element such that $j^2 = -1 \mod M$

Two real convolutions are required to compute y where $y_i = x_i * h_i$ and these are shown in equations (4) and (5).

$$y'_i + j\,y''_i = a_{xi} * a_{hi} \quad \mod M \qquad (3.4.4)$$

$$y'_i - j\,y''_i = b_{xi} * b_{hi} \quad \mod M \qquad (3.4.5)$$

Where the following are defined

$$a_{xi} = x'_i + j\,x''_i, \quad a_{hi} = h'_i + j\,h''_i \quad \mod M \qquad (3.4.6)$$

$$b_{xi} = x'_i - j\,x''_i, \quad b_{hi} = h'_i - j\,h''_i \quad \mod M \qquad (3.4.7)$$

Two methods are presented for finding an element j such that $j^2 = -1$ in the ring Z_M. If a primitive element α_p is known (an element of order $\emptyset(M)$) then an element of order 4 can easily be found by:

$$j = \alpha_p^{\emptyset(M)/4} \quad \text{mod } M \qquad (3.4.8)$$

It can be seen that this element of order 4 will satisfy $j^2 = -1$ mod M. An efficient technique has already been presented for finding a suitable primitive element.

Ore (68) presents an alternative procedure for finding such an element j for prime moduli. The solution he presents is

$$j = \pm \left[\frac{M-1}{2} \right] ! \quad \text{mod } M \qquad (3.4.9)$$

Much interest is shown in this chapter in the modulus M = 65521 for which j = 24297 satisfies $j^2 = -1$ mod M.

A discussion on the operation counts required by this technique for complex convolutions will be presented in section 5.4

It has been the purpose of this section to extend Nussbaumer's algorithm for performing complex convolutions by number theoretic tranform techniques to classes of moduli other than Fermat moduli.

3.5 EXTENSION TO OTHER MODULI

For a given modulus the output must be limited to avoid overflow; hence a compromise exists between the data amplitude and the filter impulse digitisation. The nature of this compromise is studied more closely in chapter 4. In general the digitisation will degrade the filter response, and so for a given choice of modulus there may be insufficient dynamic range for the filter design to achieve the limits set. In such cases, a larger modulus should be chosen; however, direct implementation of such a scheme would involve performing arithmetic with greater wordlength. This problem may be circumvented by use of the Chinese remainder theorem.

This theorem states that if an integer x is such that x = a_i mod m_i, where a set of moduli m_i is relatively prime, then x may be determined with respect to the modulus which is the product of these relatively prime moduli. An interpretation of this theorem shows that if calculations are performed with respect to two or more relatively prime moduli (m_1 and m_2) then by using the CRT the results may be determined mod (m_1m_2).

Therefore a search was made for other moduli that would combine with 65521 over specific transform lengths. The search was conducted for various transform lengths by scanning from 2^{16} downwards for moduli, other than 65521, for which constraints (a) to (d) would be satisfied. These results are shown in table 3.1

It can be seen from the last two entries in table 3.1 that the highest suitable modulus below 2^{16} that will directly support a transform length of 5040 is M = 55441. The choice of such a low modulus is undesirable, since a great loss occurs of the possible dynamic range of 2^{16}.

Agarwal and Cooley (7) have described how the Chinese remainder theorem may also be employed to convert a one dimensional cyclic convolution to a multidimensional convolution which is cyclic in all dimensions. This may be applied to cases where a given long convolution length is a product of shorter mutually prime convolution lengths. They cite an example whereby a number theoretic transform technique can be used for convolution of lengths N and by using the Chinese remainder theorem in this manner convolutions of length

TABLE 3.1

TABLE OF DUAL MODULI TO PAIR WITH M=65521 USING C.R.T.

CONVOLUTION LENGTH	DUAL MODULUS	PRIMARY WINOGRAD TRANSFORM LENGTH	MULTIDIMENSIONAL FACTOR REQUIRED
6	65497	6	-
10	65381	10	-
12	65497	12	-
14	65437	14	-
15	65101	15	-
18	65449	18	-
20	65381	20	-
21	65437	21	-
24	65497	24	-
28	65437	28	-
30	65101	30	-
35	65381	35	-
36	65449	36	-
40	64921	40	-
40	65497	8	5
42	65437	42	-
45	64621	45	-
45	65449	9	5
48	65281	48	-
56	65353	56	-
60	65101	60	-
63	65269	63	-
70	65381	70	-
72	65449	72	-
80	64081	80	-
80	65393	16	5
84	65437	84	-
90	64621	90	-
90	65449	18	5
105	65101	105	-
112	64849	112	-
112	65393	16	7
120	64921	120	-
120	65497	24	5
126	65269	126	-
140	65381	140	-

TABLE 3.1 (CONTINUED)

TABLE OF DUAL MODULI TO PAIR WITH M=65521 USING C.R.T.

CONVOLUTION LENGTH	DUAL MODULUS	PRIMARY WINOGRAD TRANSFORM LENGTH	MULTIDIMENSIONAL FACTOR REQUIRED
144	65089	144	-
168	65353	168	-
180	64621	180	-
180	65449	36	5
210	65101	210	-
240	64081	240	-
240	65281	48	5
252	65269	252	-
280	63841	280	-
280	65381	35	8
315	59221	315	-
315	65381	35	9
336	64849	336	-
336	65281	48	7
360	64081	360	-
360	65449	72	5
420	65101	420	-
504	64513	504	-
504	65449	72	7
560	63841	560	-
560	65281	16	5*7
630	59221	630	-
630	65381	70	9
720	64081	720	-
720	65089	144	5
840	63841	840	-
840	65353	168	5
1008	64513	1008	-
1008	65089	144	7
1260	59221	1260	-
1260	65381	140	9
1680	63841	1680	-
1680	65281	48	5*7
2520	55441	2520	-
2520	65449	72	5*7
5040	55441	5040	-
5040	65089	144	5*7

($N \circ p_1 \circ p_2 \cdot p_3 \ldots p_j$) may be computed provided N, $p_1, p_2, p_3,,p_j$ be mutually prime.

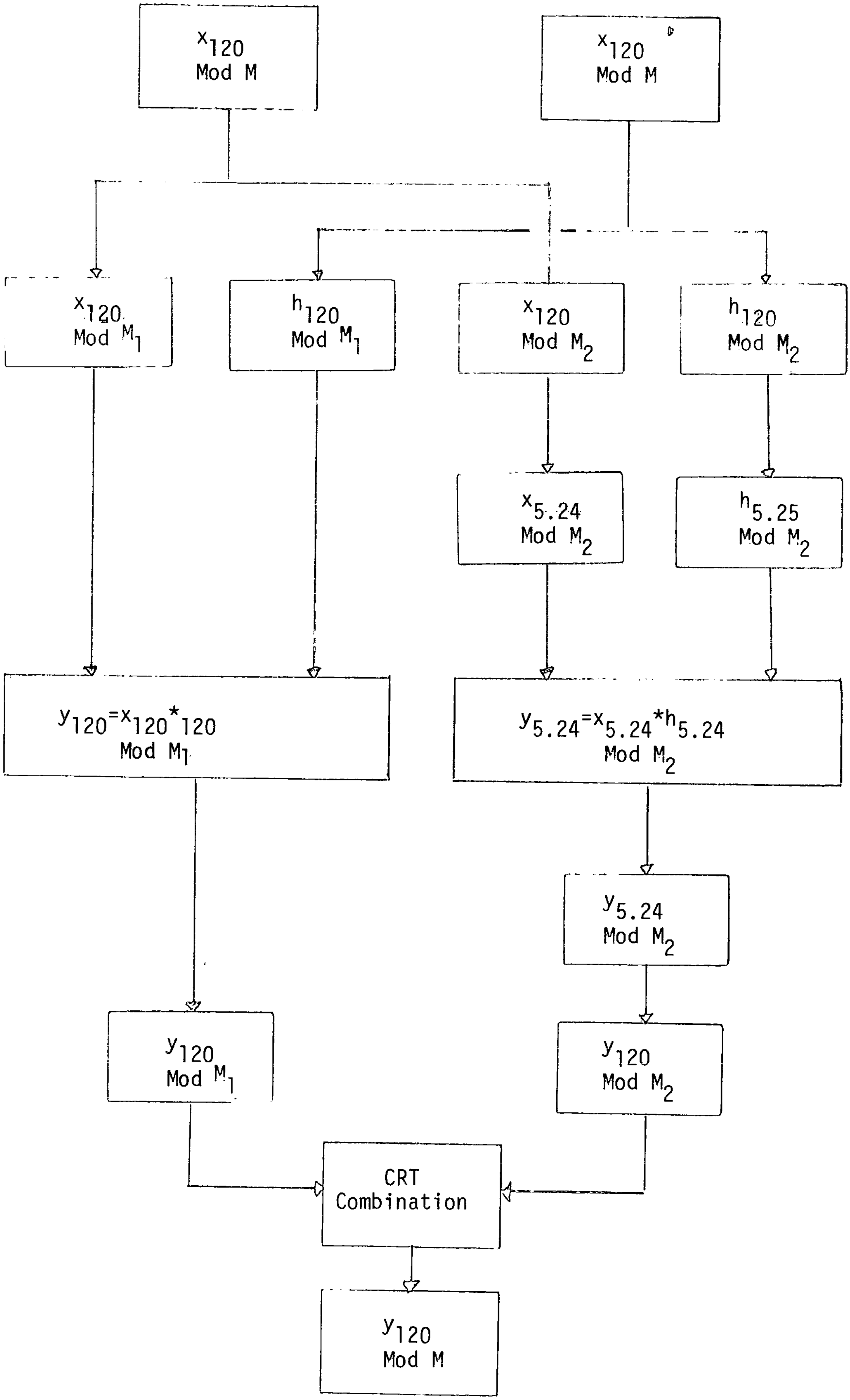
Efficient algorithms are described in (7) for convolutions of lengths 5, 7, (2, 4, 8) and (3, 9). Using such a scheme it is possible to perform NTT convolutions of length 144 and using the multidimensional mapping technique to derive convolutions of length 144 x 5 x 7 (=5040). Constraints upon the choice of modulus arise only from the NTT length used and not from the multidimensional factors employed. Therefore modulus M = 65089 (this is the choice for a length 144 transform) may be used for convolutions of length 5040, even though this modulus does not support such a transform length directly; by taking NTT convolutions of length 144 and using the CRT mapping technique to compute the 5040 length convolutions.

This technique can be used to factor transform lengths with inefficient moduli to lengths with more efficient moduli and the other entries in the table have been derived in similar manner. The Winograd transform algorithms and the Agarwal and Cooley convolution algorithms are designed for lengths which are a product of short factors. It is worthwhile pointing out that since only relatively prime algorithms can be combined with the Chinese remainder theorem the range of possible combinations is restricted to cases where the multidimensional factors are relatively prime to the transform lengths.

It is advantageous if the two moduli to be combined are used with the same transform lengths, since the same permutation sequences and essentially the same algorithms are used for both moduli, leading to economy in memory utilisation.

Figure 3.1 shows a scheme where 120 point convolutions may be

FIGURE 3.1 CONVOLUTION VIA SIMULTANEOUS MODULI



$M_1 = 65521$ $M_2 = 65497$

performed mod (65521 x 65497). The convolution mod 65521 can be performed directly using a 120 point Winograd transform. The sequences mod 65497 are mapped to (5 by 24) sequences and a two dimensional convolution is used. This requires ten 24-point convolutions which may be calculated by a direct Winograd transform method. The results of the two dimensional convolution are mapped back to a one dimensional sequence. Finally the results (mod m_1 and mod m_2) are interpreted mod M by the use of the Chinese remainder theorem.

The complex convolution procedure outlined in section 3.4 and the Chinese remainder combination procedure for the moduli shown in table 3.1 have both been compared with reference tecniques in the manner described previously in section 3.3.

CHAPTER 4

ASPECTS OF MODULAR ARITHMETIC

Number theoretic transforms provide an integer processing technique for signal processing. It is therefore prudent to examine the consequences of using such integer techniques. With fixed and floating point arithmetic round-off error is of concern, however integer techniques are entirely accurate provided overflow does not occur. The criteria for this condition are examined and reliable design techniques are presented to prevent integer overflow.

In certain cases there is insufficient dynamic range provided by a given modulus for the design criteria to be met without overflow. In such cases a larger modulus should be chosen. The Chinese remainder theorem provides a technique which achieves this without requiring arithmetic with increased wordlength. The consequences of this approach are examined in more detail.

Microprocessors handle most efficiently integer data and so it is relatively easy to program microprocessors to perform modular integer arithmetic. The general classes of arithmetic are studied in order to provide efficient programming procedures.

However it has been found that modular multiply by software even with advanced microprocessors is slow and so a hardware modular multiplier is described which will easily interface to most microprocessors.

4.1 FILTER DESIGN FOR MODULAR ARITHMETIC

In any practical situation when working with digital machines sampled data is available only with some finite precision. Therefore without any loss of generality input data can be considered to be integer with some upper bound and a scale factor applied. If the results of a digital filtering operation are conveyed back to the real world then the output is also presented in integer form, since it too is conveyed with a finite precision. It is therefore a logical step to adopt an integer technique for signal processing, and as discussed previously number theoretic transforms provide such a technique.

For a given maximum output of a filter there exists a compromise between the input data amplitude and the filter impulse digitisation. The purpose of this section is to examine more closely the criteria which govern this compromise.

NTTs produce exact results reduced mod M. However, numbers mod M do not necessarily have any physical significance since the mapping from Z_M to Z, the infinite set of integers, is a one to many mapping. For this reason we must apply bounds to the output of the convolution operator.

The maximum output of a convolution, y_{max}, is governed by:

$$|y_{max}| < x_{max} \cdot \sum_{i=0}^{N-1} |h_i| \qquad (4.1.1)$$

$$|y_{max}| < h_{max} \cdot \sum_{i=0}^{N-1} |x_i| \qquad (4.1.2)$$

If $|y_{max}|$ is bounded by

$$|y_{max}| < M/2 \qquad (4.1.3)$$

then the full dynamic range of M can be utilized without overflow.

Let

$$x_{max} \cdot \Sigma |h_i| < M/2 \qquad (4.1.4)$$

Thus given an input data amplitude, constraints limit the absolute summation of the impulse response.

Let f_i be a real filter impulse response derived by conventional techniques and let

$$F = \sum_{i=0}^{N-1} |f_i| \qquad (4.1.5)$$

We require to digitise the filter coefficients but with as fine a resolution as possible. If a scale factor (k,k>1) is applied to f_i before digitisation, then the resolution with which the filter coefficients are digitised has been increased. Let D denote the operation of digitisation then if we choose k_m such that

$$x_{max} D (k_m F) = M/2 \qquad (4.1.6)$$

we have optimised the resolution whilst guaranteeing that overflow should not occur. k must be less than k_m in practice since slight rounding up may occur in the digitisation operation.

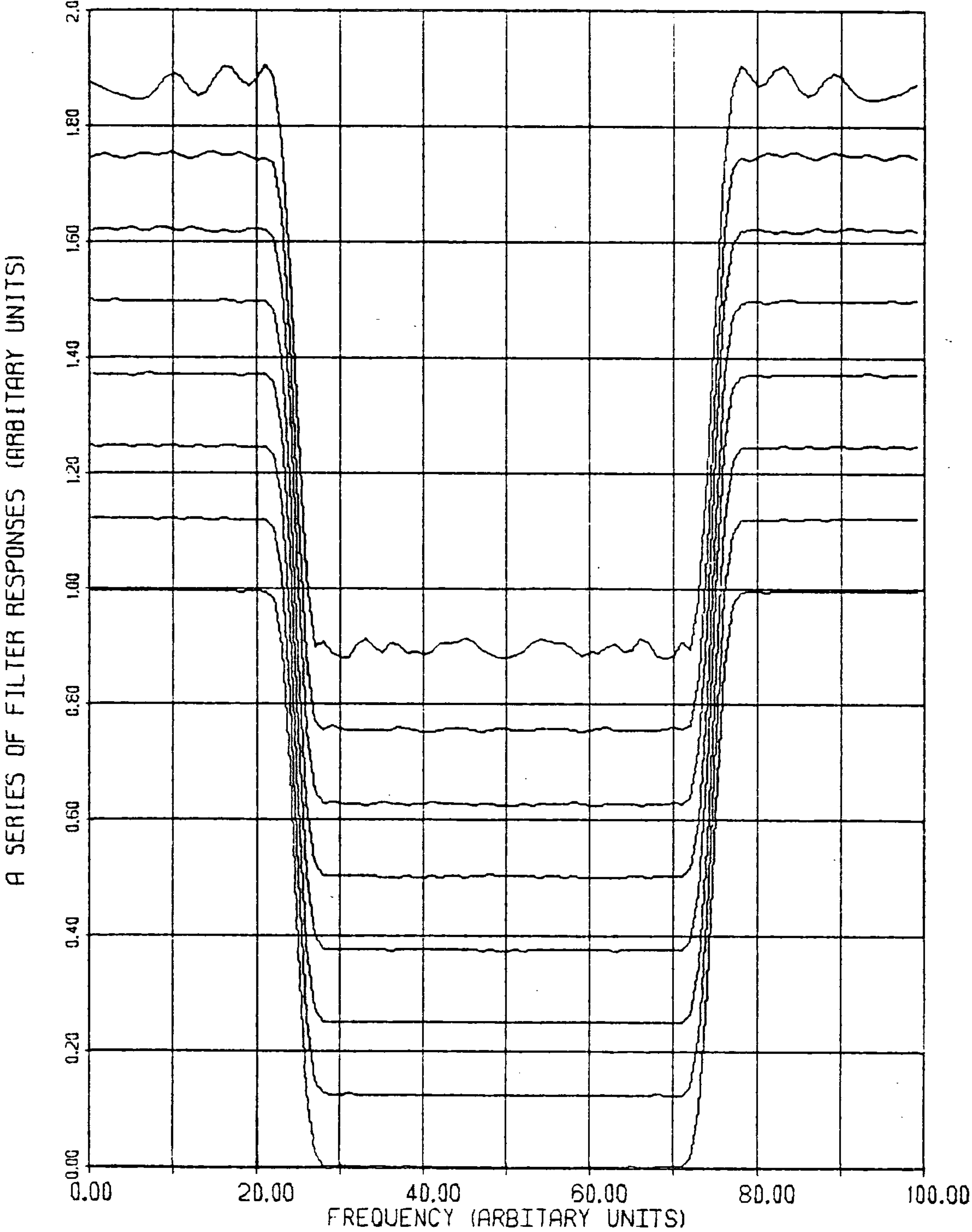
Therefore the best filter impulse response can be found by:

$$h_i = D(kf_i) \quad i \in (0,N-1) \text{ such that } k < \frac{M}{2 \; x_{max} \sum_{i=0}^{N-1} |f_i|} \qquad (4.1.7)$$

The input data amplitude, x_{max}, plays an important role in this relation and the filter digitisation can loosely be said to be inversely proportional to the input data amplitude.

By way of example a series of filter frequency responses are shown in figure 4.1. The curves are derived by taking the Fourier transform of various digitisations of a set of time domain

FIGURE 4·1 FILTER DEGRADATION WITH QUANTISATION


A SERIES OF FILTER RESPONSES (ARBITARY UNITS)
0.00
0.20
0.40
0.60
0.80
1.00
1.20
1.40
1.60
1.80
2.00
0.00
20.00
40.00
60.00
80.00
100.00
FREQUENCY (ARBITARY UNITS)

coefficients. The more deviant curves result from the coarser digitisation which would be asscocaited with larger input data amplitudes.

As can readily be seen from figure 4.1 the digitisation of the filter coefficients tends to degrade the filter response and so for a given input data amplitude and for a given modulus there may be insufficient dynamic range to represent the filter response within the limits set.

Conventially only the convolution length limits the accuracy with which the desired filter response can be achieved. However when using modular arithmetic there exists this additional compromise between the input data amplitude and the filter response. For filter designs these two effects must be considered together.

It may be that insufficient dynamic range is provided by a given modulus and in such cases a larger modulus should be chosen. However a direct implementation of such a scheme would involve performing arithmetic with greater wordlength. This would neccessarily slow the processor down. There are two techniques which are suitable for parallel processing and in such cases the overall speed may not be impaired if multiple processors are used.

Agarwal and Burrus (1) have described a technique which they have termed segmentation. This involves taking the input sequences and segmenting them into shorter blocks and convolving these separatly.

Let
$$x(i) = x_2(i) + x_1(i)2^k \qquad (4.1.8)$$

$$h(i) = h_2(i) + h_1(i)2^k \qquad (4.1.9)$$

where $0 < |x(i)|,\ |h(i)| < 2^{2k}$ and $0 < |x_2(i)|,\ |h_2(i)| < 2^k$

Then

$$y = x * h$$
$$= x_1 * h_1 2^{2k} + (x_1 * h_2 + x_2 * h_1)2^k + x_2 * h_2 \qquad (4.1.10)$$

Now since x_1, h_1, x_2 and h_2 have roughly half the number of bits it is possible to convolve them with aproximatly half the number of bits. In equation (10) the last term is small compared to the first and in most cases this term can be neglected. We need to take two forward transforms for x and two for h. The summation in brackets can be performed in the transform domain. Finally two inverse transforms are required one for $x_1 * h_1$ and one for $(x_1 * h_2 + x_2 * h_1)$.

An alternative solution to this problem is provided by the Chinese remainder theorem and it is the purpose of the next section to study this solution more closely.

4.2 CHINESE REMAINDER THEOREM

If the result of an arithmetic operation is known relative to a set of coprime moduli then by using the Chinese remainder theorem (CRT) this result may be determined with respect to the modulus which is the product of these coprime moduli. Therefore if convolution is simultaneously performed with respect to two or more such moduli then the Chinese remainder theorem can be employed to obtain the results with increased accuracy. It is this property which allows increased precision to be achieved using a processor structure which is itself well suited to a parallel technique.

The Chinese remainder theorem states that if an integer x is such that $x = a_i \bmod m_i$ where a set of moduli m_i is relatively prime (coprime) then

$$x = a_1 b_1 \frac{M}{m_1} + a_2 b_2 \frac{M}{m_2} + a_3 b_3 \frac{M}{m_3} + \ldots + a_l b_l \frac{M}{m_l} \bmod M \qquad (4.2.1)$$

where $M = \overset{l}{\underset{i=1}{\pi}} m_i$

The b_i are defined such that

$$b_i \frac{M}{m_i} = 1 \bmod m_i$$

Let us consider the case for two moduli

$$x = a_1 c_1 + a_2 c_2 \bmod M \qquad (4.2.2)$$

Where

$$c_1 = b_1 \frac{M}{m_1} \quad \text{and} \quad c_2 = b_2 \frac{M}{m_2}$$

Let $x = 1$ and so $a_1 = 1 = a_2$, therefore

$$1.\, c_1 + 1.\, c_2 = 1 \bmod M$$

or

$$c_1 + c_2 = 1 \bmod M \qquad (4.2.3)$$

If the two moduli m_1 and m_2 are both b-bit then in general c_1 and c_2 would be 2b-bit constants and therfore a direct implementation of equation (2) would require two b x 2b bit multiplications. By using the result in equation (3) this requirement can be reduced.

From equations (2) and (3)

$$x = a_1 c_1 + a_2 (1 - c_1) \qquad (4.2.4)$$

$$= c_1(a_1 - a_2) + a_2 \bmod M \qquad (4.2.5)$$

However c_1 is defined by

$$c_1 = b_1 \frac{M}{m_1} \qquad \bmod M \qquad (4.2.6)$$

and therefore

$$c_1 m_1 = b_1 M$$

or

$$c_1 m_1 = 0 \qquad \text{mod } M \qquad (4.2.7)$$

Thus any integer multiple of $c_1 m_1$ can be added into equation (5) without altering the result, hence

$$x = c_1 (km_1 + a_1 - a_2) + a_2 \quad \text{mod } M \quad k \in \qquad (4.2.8)$$

If k is chosen such that $(km_1 + a_1 - a_2)$ lies in the range zero to $2^b - 1$ then only one b x 2b bit multiplication is required to evaluate equation (2). This operation is all that is required for the recombination of results with respect to two moduli using the Chinese remainder theorem.

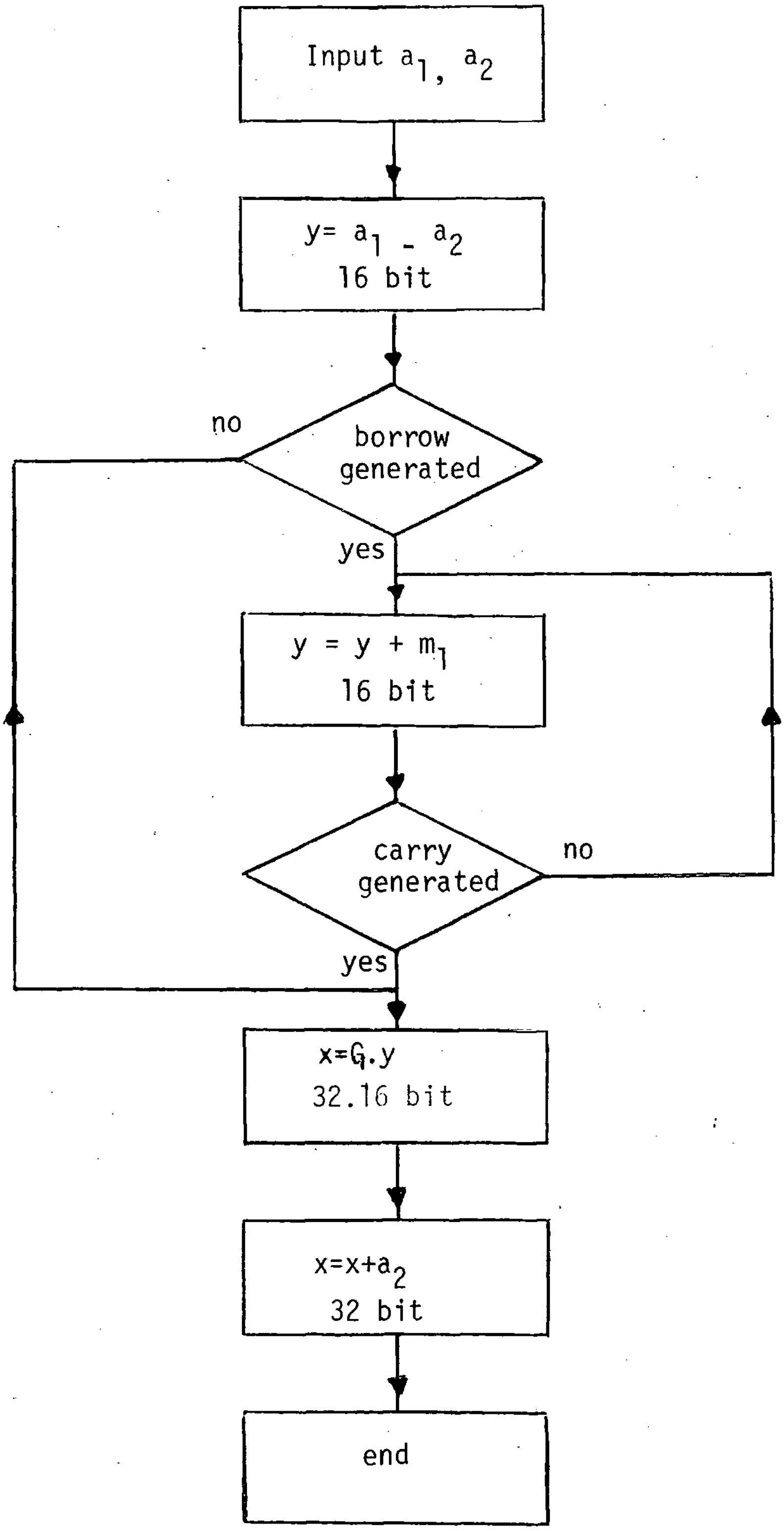
A flow chart to achieve this algorithm for two 16 bit moduli is shown in figure 4.2

Heindel and Horowitz (32) have introduced a concept which they have termed the parallel Chinese remainder algorithm (PCRA). This is best illustrated by example. If there are four coprime b bit moduli for which results are to be combined then using the analagous form of equation (2) it can be seen that four b x 4b bit multiplications are required by the conventional iterative Chinese remainder algorithm [Garner's algorithm, (32), (27)]. By using the PCRA the four moduli are paired as follows:

$$(P_1, P_2) \quad , \quad (P_3, P_4)$$

and these pairs are individually combined using the Chinese remainder theorem. There now exist two moduli P_1P_2 and P_3P_4 for which results can subsequently be combined. The factorisation by two of the combination operator achieves computational savings if there exist efficient techniques for the two modulus combinations. The technique described in equation (8) is therefore well suited to the PCRA. If the

FIGURE 4.2 FLOWCHART TO PERFORM CRT COMBINATION.



four b bit moduli are combined using the PCRA and the two modulus combination technique, it can be seen that only two b x 2b bit and one 2b x 4b bit multiplications are required and this technique is therefore more efficient than Garner's algorithm. This composite algorithm promises to be an efficient technique when the number of moduli to be employed is chosen to be a power of 2. Such a technique is well suited for parallel processing.

It has been the purpose of this section to introduce the Chinese remainder theorem and to show how it may be used to derive results with greater precision. This promises to be the technique best suited to a multiprocessor system. Processors can be dedicated to determining results for one of the chosen moduli and a further processor can be used for the PCRA combination for the final presentation of the results.

It is interesting to note that for filter design with such systems the filter coefficients need only be determined with respect to the final composite modulus M. For each sub modulus processor the filter coefficients are reduced mod m_i and so do not have any physical significance, though they still have to be determined. It is only at the final Chinese remainder combination that physical significance is restored to the processed data.

4.3 MICROPROCESSORS AND MODULAR ARITHMETIC

The modluli previously described are optimally close to 2^{16} and therefore only a small duplicity arises if all 16 bit binary patterns are allowed on input and output with an arithmetic procedure. In all examples in this section the choice of M = 65521 will be taken.

a: Additiion mod M

When two numbers are added together they may or may not generate an overflow. If there is no overflow then the result of the addition is returned.

If
$$x = a + b = c + \text{carry} \qquad (4.3.1)$$
$$= c + 2^{16} \qquad (4.3.2)$$
$$= c + (2^{16} - M) \quad \text{mod } M \qquad (4.3.3)$$
$$= c + 15 \quad \text{mod } 65521 \qquad (4.3.4)$$

Therefore if a carry is detected, 15 must be added into the partial sum. This may generate a futher carry, but will not generate more than a total of two carries. An example of suitable coding for such an operation is given for an Intel 8080 microprocessor.

```
PLUS    DAD    D
        RNC
        LXI    D, 15
        JMP    PLUS
```

This subroutine will add mod 65521 the two 16 bit numbers in register pairs (DE) and (HL) returning the answer in (HL).

b: Subtraction mod M.

For microprocessors with 16 bit subtraction instructions the 16 bit addition instruction in the previous example may be directly replaced by such an instruction. However few 8 bit microprocessors have such instructions, and so for the majority a byte orientated subtraction should be used.

c: Multiplication mod M

This operation can be classified into subdivisions.

(i) Multiplication of a 16 bit number (x) by an 8 or 16 bit constant (k).

This can be considered to be a mapping from a 16 bit number to another 16 bit number. Since multiplication is a distributive operation then the mapping may be achieved by treating separatly the low and high bytes of x and finally using a modular add to combine their respective outputs together. Figure 4.3 shows how such a mapping may be achieved. The obvious design requires two 2^8 x 16 bit patterns which may conviently be read only memory (ROM) and the read operations can be controlled by the microprocessor.

For the multiplication by an 8 bit constant the memory can be restructured to be byte orientated and in so doing a saving is obtained since part of the memory R_1 is duplicated in R_2. The minimized structure is shown in figure 4.4. ROMs R are read once only and ROM R' is read twice. This memory reduction scheme seems well suited to 8 bit processors. However for 16 bit processors exta software effort would be required to implement the byte orientated scheme. The possible memory saving is offset by necessarily slowing down the execution speed of such processors and since memory is dropping in price this does not seem worthwhile.

(ii) Multiplication of a 16 bit number (x) by a 32 bit constant.

This operation is required when combining the results of arithmetic procedures with two 16 bit moduli by using the Chinese

FIGURE 4.3. MULTIPLICATION BY 8 BIT FIXED CONSTANT.

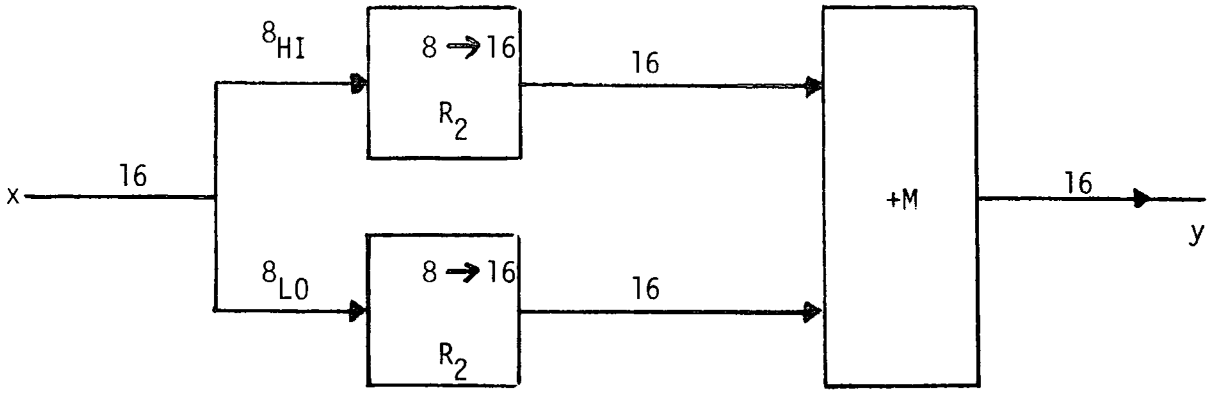
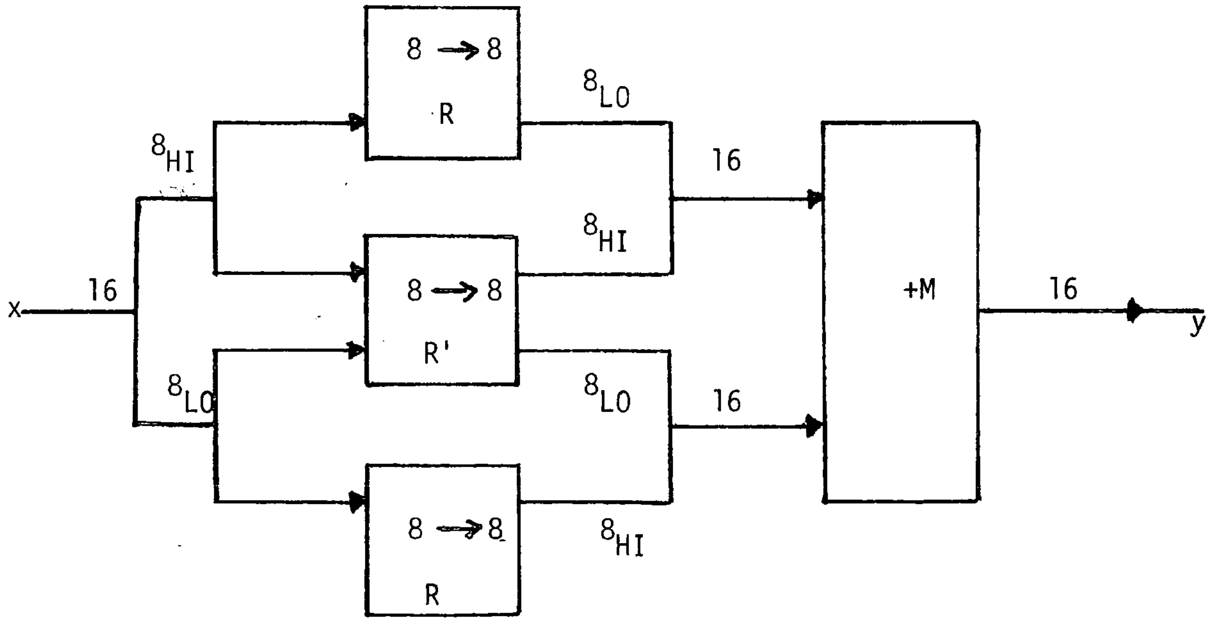


FIGURE 4.4 MEMORY MINIMISATION FOR 8 BIT FIXED CONSTANT MULTIPLICATION



remainder theorem. This class of operation can be considered to be a mapping from a 16 bit number to a 32 bit number. By adopting a similar scheme and treating the low and high bytes of x separatly the mapping may be achieved with two 2^8 x 32 bit patterns which can also be ROM. A final 32 bit modluar add is required to derive the final output. This operation can be performed in an analogous manner to that described in section 4.3.a.

(iii) Multiplication of two 16 bit variables mod M

The multiplication of two 16 numbers generates a 32 bit answer.

$$\text{Let } y = ab = 2^{16}.y_h + y_l \qquad 0 < a,b,y_h,y_l < 2^{16} \qquad (4.3.5)$$

$$= (2^{16} - M).y_h + y_l$$

$$= 15 \,.\, y_h + y_l \qquad \text{mod } 65521 \qquad (4.3.6)$$

Therefore a general 32 bit intermediate answer may be partially reduced mod M by a multiplication of y_h by the fixed constant $(2^{16} - M)$. This can be achieved by the scheme described in 4.3.c part (i).

(iv) Multiplication by 2^{-1}

This is the analogous operation to division by 2.

$$\text{Let} \qquad y = 2^{-1}.\, x \qquad (4.3.7)$$

By applying a shift right to x we may determine from the carry flag if x were even or odd. If x was even then we have already determined y. If x was odd then y may be derived by adding in (M + 1)/2.

The procedures described cover the general classes of arithmetic required for convolution performed within the ring Z_M where M is a

TABLE 4.1

SUMMARY OF MEMORY REQUIREMENTS FOR MULTIPLICATION LOOK-UP TABLES

		Memory required pages	
		For each modulus	For 2 moduli
Real convolutions	Generalized Multiplications	3	6
	C.R.T combination	-	8
	Total	3	14
Complex convolutions	Generalized Multiplications	3	6
	Multiplication by j	4	8
	C.R.T. combination	-	8
	Total	7	22

product of 65521 and one of the moduli shown in table 3.1.

Table 4.1 summarizes the memory requirements for the multiplication look up tables. The unit of one page is used to denote a quantity of memory of 256 bytes. In the case of multiplications of two 16 bit variables 3 pages are required for the table to derive the term corresponding to the multiplication of the higher 16 bit intermediate answer with the fixed constant $(2^{16} - M)$. The multiplication by 'j' can be most efficiently performed by the technique described in section 4.3.c part (i) and this requires an additional 4 pages.

4.4 A HARDWARE MODULAR MULTIPLIER

The algorithms described in chapter 3 have been designed for an implementation on a microprocessor with fast multiplication. As has been described in this chapter, number theoretic transforms require modular integer arithmetic, and so when two 16 bit variables are multiplied the intermediate 32 bit product must be reduced mod M. This requires a multiplication of the high 16 bit part of the product by the fixed constant $(2^{16} - M)$ and this is followed by a modular addition. It has been shown how the fixed multiplication by $(2^{16} - M)$ can be achieved by a memory look up technique.

As has been outlined the procedure for general modular multiplication is involved. Preliminary timings show that for the TEXAS 9900 microprocessor such operations take in the order of 90 μs. This compares poorly with the 19 μs required for an 16 x 16 bit unsigned integer multiplication on the same machine.

Davies and Fung (117) have described how a hardware multiplier may be attached to a microprocessor system in order to increase its performance. It therefore seems a logical step to append a fast modular integer multiplier to such systems and this should greatly increase the throughput of such systems.

If the modular operations of add and multiply are compared with their integer equivalents, it is apparent that the modular addition is relatively easy to accomplish in software and there seems little point in designing specific hardware to achieve this function. However as has been described this operation is required during the computation of a modular multiply and so hardware will be described to perform the modular additions required.

When a modular add is performed between two 16 bit quantities a carry may be generated. If this is so then $(2^{16} - M)$ must be added into the sum. This addition may generate a further carry. However only a total of two such carries may be generated by this action. This situation can be achieved by the addition of for example FFFF and FFFF (in hexadecimal) with M = 65521 or $2^{16} - M = F$.

Then:

$$\begin{array}{r l}
\text{F F F F} & (= 14) \\
\underline{+\ \text{F F F F}} & \\
1 : \text{F F F E} & \\
\underline{\qquad\qquad +\ \text{F}} & \\
1 : \text{0 0 0 D} & \\
\underline{\qquad\qquad +\ \text{F}} & \\
\underline{\qquad\qquad \text{1 C}} & (= 28)
\end{array}$$

The hardware described in figure 4.6 performs the modular add operation. The value of $(2^{16} - M)$ is stored in latch 1 and this is fed

FIGURE 4.5 MODULAR ADDER.

C = a + b mod M.

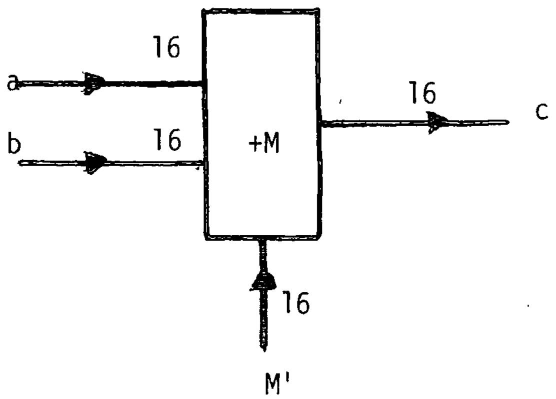
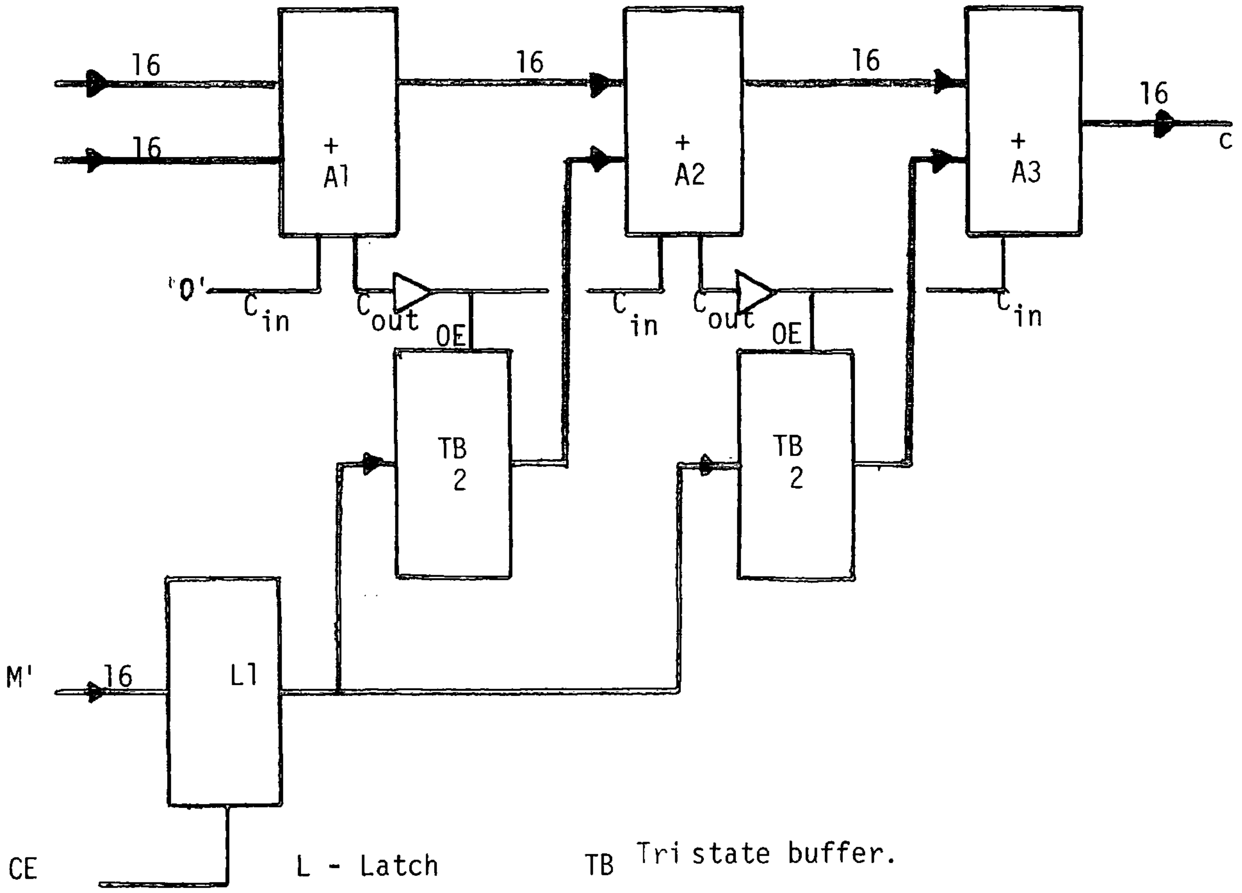


FIGURE 4.6 MODULAR ADDER HARDWARE.



L - Latch TB Tri state buffer.
A - Adder

FIGURE 4.7 MODULAR MULTIPLIER

c = a * b mod M

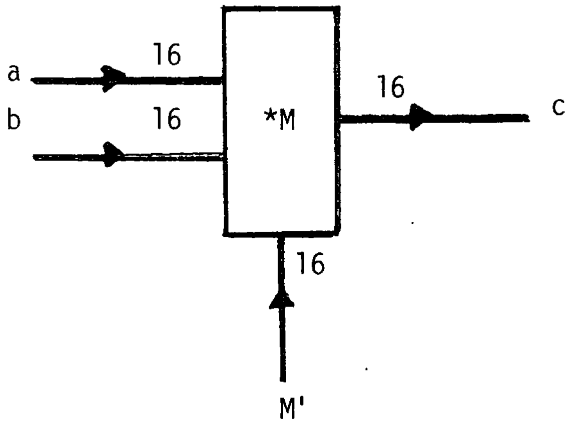
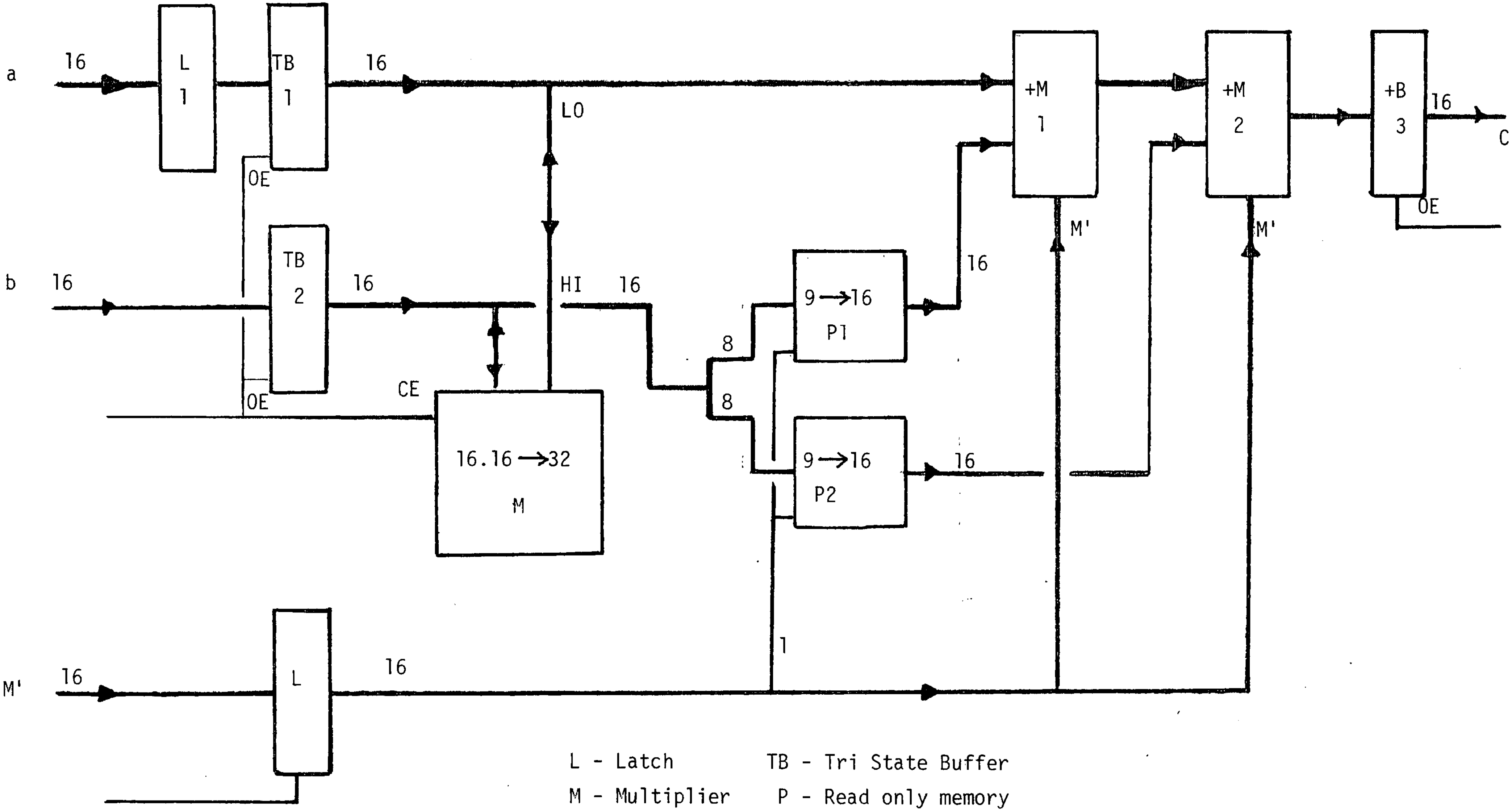


Figure 4.8 Modular Multiplier



L - Latch TB - Tri State Buffer
M - Multiplier P - Read only memory

to tri-state buffers 1 and 2. These buffers provide a switchable 'on carry add in (2^{16}- M)' operation. If a carry out is generated by the first adder then tri-state buffer 1 is enabled and no carry is passed to the second adder. If no carry is generated from A1 then tri-state buffer 1 is disabled and so the inputs to A2 will float to logic 1 and this corresponds to adding in '-1'. However a carry-in is also present and so effectively '0' is added in. This procedure is repeated for the second adder stage. This hardware provides a fast asynchronous modular adder.

Figure 4.8 describes the hardware for the modular multiply operation. Hardware multiplier chips are becoming available to provide a fast multiplication of two 16 bit variables to produce a 32 bit product. However of those that are available, the outputs and inputs tend to be multiplexed together. When tri-state buffers 1 and 2 are enabled from the microprocessor the multiplier chip is also enabled. After the input phase the multiplier chip multiplies the two 16 bit variables and outputs the 32 bit product towards the adders +M1 and +M2.

The high 16 bit part of the product is separated into two 8 bit address lines for read only memories P1 and P2. These two memories perform the multiplication by the fixed constant (2^{16} - M). However to allow for the use of two moduli for Chinese remainder combination each of these two memories can effectively be split into two parts, one for each modulus. These can be considered to be 9⟶16 bit memories. The 9th address line of the input is a 'modulus select' signal. Since the two moduli must differ by at least one binary bit this address line can be taken from a selected data line of the modulus latch (L1).

The read only memories generate two 16 bit words which must be added with the low 16 bit part of the 32 bit product. This requires a total of two modular adders described earlier. The two modular adders can both be driven from the same modulus latch. Tri-state buffer 3 can be enabled by the microprocessor when it requires to read back in the result of the modular multiply.

After the initial input phase the multiplier, which has been described, is completely asynchronous and the final output would be derived within about 5 μs; during which time the processor will be able to store the last result and be accessing new data for the multiplier. If the microprocessor is faster than the multiplier then a small degree of pipelining may be achieved by the use of latch 1. The value for the next multiply operation can be loaded into latch 1 before the result of the previous multiply has been calculated. As soon as this has occured the result may be read and the other input, b, may be loaded and this reinitiates the multiply cycle. If the multiply is slower than the microprocessor then this will increase the throughput.

The hardware which has been described in figure 4.8 will perform fast modular multiplies and this will greatly increase the processor throughput. The cost of this hardware is in the order of £300 at 1980 prices.

CHAPTER 5

COMPLEX TRANSFORMS

The major part of this work has been to consider arithmetic over a finite integer field Z_M with a view to deriving efficient algorithms for signal processing using a microprocessor. Reed (86) has shown that the finite Galois field $GF(q^2)$ may also be used for efficient signal processing. This field has intersting similarities with the familiar complex field and using $GF(q^2)$ it is possible to define number theoretic transforms which are structurally identical to those defined over Z_M. The number theory of complex quadratic rings will be unfamiliar to non-mathematicians, and inorder to introduce the relevant nomenclature a brief introduction to the topic of number theoretic transforms defined over such rings is presented in section 5.1.

In chapter 3 it was shown that microprocessors are suited to moduli which are just less than 2^{16}, and so the same techniques are applied in this chapter to find finite fields $GF(q^2)$, which are suited to a microprocessor implementation of complex number theoretic transforms.

In order to perform transforms of length N it is neccessary to find elements (α) of order N. Since it is in principle difficult to find such elements over $GF(q^2)$ the search procedure and results are presented.

A novel convolution procedure is proposed in which complex convolution is performed simultaneously over $GF(q^2)$ and over Z_M using related M and q.

The chapter concludes with a brief discussion on real and complex transform techniques and it is shown that number theoretic transforms have a structural advantage over the Fourier transform for convolution since both real and complex number theoretic transforms can be defined.

5.1 INTRODUCTION

Vanwormhoudt (109) has shown that there are two main classes of prime moduli useful for number theoretic transforms. There are moduli for which M = 1 mod 4 (type A) and those for which M = 3 mod 4 (type B). It is easily shown that for type A moduli there will exist an element 'j' such that $j^2 = -1 \bmod M$, and that no such element will exist for type B moduli. This observation indicates a key structural difference between these two classes of moduli.

This structural difference can be formulated by stating that the polynomial

$$P(x) = x^2 + 1 = 0 \tag{5.1.1}$$

has a solution in Z_M [or GF(M)] for type A moduli and it does not have a solution in Z_M for type B moduli. Reed (86) has shown that a root (i') of this polynomial will exist in an extension field $GF(M_B^2)$ for type B moduli. The Galois field $GF(q^2)$ is composed of the set

$$GF(q^2) = [\ a + i'b\ ] \qquad a,b \in GF(q)$$

where i' is a root of equation (1) satisfying $(0 + i')^2 = (-1 + 0i')$ in $GF(q^2)$.

If $x^2 + 1 = 0$ is not solvable in GF(q) then $i' \in GF(q^2)$ plays a similar role over the finite field GF(q) that -1 plays over the field of rationals. A mapping from a set of complex integers

$[\ (a + ib)\ a,b$ are integers, such that $-(q - 1)/2 < a,b < (q - 1)/2\ ]$

to GF(q^2) may be defined by:

$$\emptyset : \ (a + i\,b) \longrightarrow (a + i'b)$$

This mapping in 'one to one and onto'. Under this mapping a complex integer number and an element of GF(q^2) can be used interchangebly. For example suppose that a + i'b and c + i'd are elements of GF(q^2) then

$$(a + i'b) \pm (c + i'd) = a \pm c + i'(b \pm c) \qquad (5.1.2)$$

$$(a + i'b) \cdot (c + i'd) = ac + i'^2 bd + i'bc + i'ad$$

$$= ac - bd + i'(bc + ad) \qquad (5.1.3)$$

These are these exact analogues one might expect if (a + i'b) and (c + i'd) were complex numbers. However it is of interest to demonstrate an important property of the Galois field GF(q^2) which the fields of complex rational numbers does not have. If $x = a + i'b \in GF(q^2)$ and $x \neq 0$ then

$$x^{q^2-1} = (a + i'b)^{q^2-1} = 1 \qquad \text{in } GF(q^2) \qquad (5.1.4)$$

which is a special case of the result $x^{q^n-1} = 1$ for all non zero x in GF(q^n)

Pollard (72) suggested that the Fourier transform may be defined in finite fields of the form GF(p^n) of which the number theoretic transforms amplified in chapters 2 and 3 are a special case. It is apparent that transforms over GF(q^2) are also a special case. However, as has been shown, the arithmetic over GF(q^2) does bear interesting parallels with the normal complex domain arithmetic.

Agarwal and Burrus (3) postulated criteria which must be satisfied for a length N number theoretic transform, which supports the cyclic convolution property, in GF(p) to exist. Such transforms are of the form.

$$X(k) = \sum_{j=0}^{N-1} x(j)\ \alpha^{jk} \tag{5.1.5}$$

$$x(j) = N^{-1} \sum_{k=0}^{N-1} X(k)\ \alpha^{-jk} \tag{5.1.6}$$

These criteria are that

(a) α be an element of order N (in each $Z_{p_i}^{r_i}$ of the factors of M, for composite M)

(b) N^{-1} must exist.

Let a composite modulus M be reduced to its prime factored form

$$M = p_1^{r1}.\ p_2^{r2}.\ p_3^{r3}\ldots.\ p_l^{rl} \tag{5.1.7}$$

Since the Chinese remainder theorem provides a ring isomorphism the study of Z_M can be limited to the case where $m = p_i^{ri}$. In $Z_{p_i}^{r_i}$ the 'maximal order' of elements is $p_i^{ri-1}(p_i - 1)$. Since a transform over Z_M can be considered to be simultaneous over each $Z_{p_i}^{r_i}$, the greatest order for which the first criterion will be satisfied, will be given by the greatest commom divisor of such terms. Using this approach Agarwal and Burrus have shown that criteria (a) and (b) are equivalent to the following:

(c) $N \mid O(M)$ where $O(M)$ = g.c.d. [p_1-1, p_2-1,, p_l-1]

By adopting a similar approach for the quadratic field $GF(q^2)$ the corresponding criterion is that

(d) $N \mid O_c(M)$where $O_c(M)$ = g.c.d.[p_1^2-1, p_2^2-1,, p_l^2-1]

provided that all the primes p_i be of type B.

For the special case of $GF(q^2)$ where q is a prime of type B then

$$N \mid (q^2 - 1)$$

This result follows since the 'maximal order' of elements in $GF(q^2)$ is $q^2 - 1$ and this in turn explains the result given in equation (4). The properties of these structures have been thoroughly examined by Vanwormhoudt (109).

Reed (86) has proposed the use of quadratic Mersenne transforms over $GF(q^2)$ where q is a Mersenne prime of the form

$$q = 2^p - 1 \qquad q,p \text{ prime}$$

He has shown that unlike their simple counterpart these transforms support radix 2 FFT algorithms, and so seem attractive for efficient computation. By way of example it can be seen that the Mersenne prime $M_7 = 127$ supports a quadratic transform over $GF(127^2)$ of length 256. However the corresponding α are not simple and so the ease of computation is lost. None the less these provide a promising class of complex transform.

5.2 MICROPROCESSOR IMPLEMENTATION OF COMPLEX TRANSFORMS

It has been shown how fast transform algorithms may be performed requiring only bit shifts and additions. However constraints limit the choice of a particular number theoretic transform. For some microprocessors no penalty is incurred by allowing more general multiplications and this relaxes the basic constraints. It is apparent that short wordlength processors are best suited to moduli which are just less than the convenient word sizes. By employing Winograd transform algorithms, new classes of transform are possible and it has been found that the choice of modulus M = 65521 is optimal. This modulus supports a transform length of 5040 and hence it supports any Winograd transform length. In so doing it incurs little redundancy in the use of 16 bit arithmetic.

It is therefore a logical step to apply the same search procedures for complex transforms. The corresponding constraints for these transforms are as follows:

(a) an element (α) of order N must be found.

(b) N^{-1} must exist

(c) N must divide $O_c(M)$ = g.c.d.[p_1^2-1, p_2^2-1,, p_l^2-1]

(d) M, and its factors if composite, must all be of type B

i.e. p_i = 3 mod 4.

Agarwal and Burrus (3) have shown that constraint (c) is equivalent to constraints (a) and (b). A search was conducted initially for a prime modulus (p) which would support a transform length of 5040 according to the following restrictions.

(e) p = 3 mod 4

(f) $5040 \mid (p^2 - 1)$

It was found that the choice of the prime modulus M = 65519 satisfied these constraints. Since this modulus is also very close to 2^{16} no search was conducted over composite moduli to satisfy constraint (c) instead of constraint (f). It is apparent that M = 65519 is close to M = 65521 which is the choice for simple transforms, and the relation between these two moduli is examined in more detail in section 5.4.

Since it has been found that the choice of M = 65519 is optimal for complex transforms a complex integer subroutine library was written to enable complex transforms to be developed using the mainframe facilities. A selection of these procedures is described in Appendix D.

Complex integer arithmetic is not supported in FORTRAN and so basic operations had to be written. FORTRAN is obviously not suited to this arithmetic, while other languages (for example PASCAL) can adapt much more easily. However the complex algorithm development was conducted in FORTRAN because using this language no problems would be encountered in interfacing to the existing modular arithmetic subroutine libraries (see Appendix A). A complex integer has twice the wordlength of an ordinary integer and so a passing convention was adopted of treating such variables as double precision (REAL*8).

In the familiar complex field there is a choice of techniques for performing complex multiplications. It is possible to perform such operations requiring 4 real multiplications and 2 real additions. It is also possible to achieve this requiring only 3 multiplications and 5 additions. In finite complex integer fields these techniques are directly applicable. The method which should be chosen for a microprocessor implementation is that which takes the least time to perform. For a machine with software multiplication it seems likely that the (3 + 5) method will be faster, but for a machine with the external hardware modular multiplier described in section 4.5 it seems likely that the (4 + 2) method would be faster. This choice is obviously machine dependant. For the algorithm development on the mainframe the (4 + 2) technique was adopted.

It has been described in chapter 3 how the general-N program was employed to derive specific algorithms for the transforms of lengths 60 and 240. It is apparent that any transform algorithm can be considered to be an efficient decomposition of the transform matrix, while the arithmetic of the algorithm is exactly that employed in the definition of the matrix. Therefore the algorithms developed for transform lengths of 60 and 240 presented in Appendix B may be directly transcribed to derive corresponding algorithms for the complex transforms of lengths 60 and 240.

The differences between these algorithm classes are that all the arithmetic operations are reinterpreted in the complex sense. Therefore normal modular addition is replaced by complex modular addition and likewise for other operations. The multiplication coefficients must also be reinterpreted as trigonometrical expressions of powers of an element of order N. The method by which an element of order N may be determined is described in the next section. The multiplication coefficients presented by Silverman (101) require only purely real or purely imaginary multiplications, and so when these coefficients were interpreted in the complex number theoretic sense it was found that they too were either purely real or purely imaginary. These complex multiplications require therefore only two real modular multiplications and no additions, instead of the 4 and 2 additions or (3 and 5 additions) which would be required for more general complex multiplications. This also implies that the memory size required for the storage of these coefficients need be no larger than that required for the corresponding real number theoretic transform algorithms provided flags be kept to denote whether a particular coefficient is either real or imaginary.

The algorithms for the complex transforms for the lengths of 60 and 240 are presented in Appendix E. These algorithms have been tested in simulation with reference techniques in the manner prescribed in section 3.3. They provided accurate results.

Reed and Truong (87) have suggested how the Chinese remainder theorem may be applied to complex transform algorithms to increase the possible dynamic range. They proposed convolutions over a direct sum of quadratic Mersenne number fields. This technique may be applied for any sets of quadratic fields in the manner in which the Chinese remainder theorem was applied in section 4.2 to perform simultaneous convolutions over pairs of simple rings.

A computer search was therefore performed to find other moduli which would combine with M = 65519 over specific Winograd transform lengths. The search was conducted initially for prime type B moduli below 2^{16}. The results of the search are shown in table 5.1

It can be seen from table 5.1 that five different prime moduli satisfy the entire range of Winograd transform lengths. The lowest of these moduli is 65071 and even this is efficient, utilizing 99.3% of the dynamic range possible with a 16 bit wordlength. It is therefore not necessary to have to employ multidimensional convolution techniques using shorter transform lengths as was the case with simple transform structures. In comparison with the selection of dual moduli for simple transforms few moduli are required. This is because the terms $q^2 - 1$ tend to be more easily divisible by transform factors than terms of the form p - 1. Since the search for prime moduli revealed a set of efficient moduli the search was not extended for composite moduli.

The combination algorithms presented in section 4.2 may be applied to complex transforms using these moduli.

TABLE 5.1

TABLE OF MODULI TO PAIR WITH M = 65519 USING THE CHINESE REMAINDER THEOREM

TRANSFORM	DUAL	TRANSFORM	DUAL
LENGTH	MODULUS	LENGTH	MODULUS
6	65479	112	65479
10	65479	120	65479
12	65479	126	65323
14	65479	140	65479
15	65479	144	65447
18	65447	168	65479
20	65479	180	65179
21	65479	210	65479
24	65479	240	65479
28	65479	252	65323
30	65479	280	65479
35	65479	315	65071
36	65447	336	65479
40	65479	360	65179
42	65479	420	65479
45	65179	504	65323
48	65479	560	65479
56	65479	630	65071
60	65479	720	65071
63	65323	840	65479
70	65479	1008	65071
72	65447	1260	65071
80	65479	1680	65479
84	65479	2520	65071
90	65179	5040	65071
105	65479		

5.3 A SEARCH FOR PRIMITIVE ELEMENTS

The previous section has described how various prime moduli in the range of 2^{16} may be used to define complex number theoretic transforms. In order to find a modulus suitable for a transform of length N it was necessary to scan over prime moduli until the following conditions were met:

(e) $p = 3 \bmod 4$

(f) $N \mid (p^2 - 1)$

and having found moduli to satisfy these criteria it is now necessary to find elements (α) of order N in the corresponding quadratic fields.

This may be accomplished by several techniques. If an element β of order kN is already known then an element α of order N can easily be derived from

$$\alpha = \beta^k \qquad (5.3.1)$$

However in general no such elements were known for the derived moduli. It was shown in chapter 2 that the number of elements of order N in given field is $\emptyset(N)$, whereas the number of elements in $GF(p^2)$ is p^2. Therefore if N is small compared with p^2 then a direct sequential search procedure for an element of a given order N will not be efficient.

The number of primitive elements is given by $\emptyset(p^2 - 1)$ which is the maximum possible number of elements of a given order in $GF(p^2)$ and so it seems worthwhile to perform initially a search for a primitive element. Once a primitive element has been found then elements of arbitary orders which divide $p^2 - 1$ can easily be found by applying equation (1).

In chapter 3 an efficient procedure for finding primitive elements was outlined. For simple rings where the 'maximal order' in the order of 2^{16} it is possible to adopt crude searching techniques. However for these quadratic fields where the 'maximal order' is in the order of 2^{32} efficient search procedures must be employed since crude techniques were found to be too slow. Let $p^2 - 1$ be reduced to its prime factored form.

$$p^2 - 1 = p_1^{r1}.\ p_2^{r2}.\ p_3^{r3}\ldots\ p_l^{rl} \qquad (5.3.2)$$

then Euler's theorem shows that for non zero α, if

$$\alpha^{(p^2-1)/p_i} \neq (\ 1 + 0i'\ ) \text{ in } \quad GF(p^2) \quad i \in (1,l)$$

then must be a primitive root in $GF(p^2)$.

A function CPRIM was written to scan over the elements $(1 + ki')$ in $GF(p^2)$ until these conditions are met. This routine employs an efficient exponentiation function (CEXPM) over $GF(p^2)$. These routines are described in Appendix D. Once a primitive element has been found then an element (α) of order N may be determined. Table 5.2 shows these primitive elements and corresponding α for the six moduli presented in section 5.2.

It can be seen that the primitive elements which have been found are fairly close to $(1 + 0i')$ and it was found that the procedure which has been described was fast in execution.

For Winograd transform lengths other than those presented in table 5.2 the corresponding α can again be derived using equation (1).

TABLE 5.2

TABLE OF PRIMITIVE ELEMENTS

MODULUS	PRIMITIVE ELEMENT	ALPHA	N
65519	1 + 28i'	16768 - 22901i'	5040
65479	1 + 4i'	18289 + 25118i'	1680
65447	1 + 2i'	8080 + 31341i'	144
65323	1 + 2i'	30782 + 23361i'	504
65179	1 + 5i'	-4900 - 5483i'	360
65071	1 + 15i'	-10665 - 27192i'	5040

5.4 PRIME PAIR MODULI

It has been found that the choice of modulus optimal for simple number theoretic transforms is $M = 65521$ and the chioce for complex number theoretic transforms has been found to be $M = 65519$. It is the purpose of this section to derive a relationship between these two choices of moduli and to propose a new scheme for complex convolution using this relation.

Much interest has been shown in the use of the Chinese remainder theorem to derive results with respect to the modulus $(m_1 m_2)$ given results mod m_1 and mod m_2, provided that m_1 and m_2 be relatively prime. It is necessary that both moduli support the same transform length for a simple application of the Chinese remainder theorem for convolution.

A scheme for convolution is proposed whereby the results are determined with respect to two moduli p and q such that p is a type A modulus and q is a type B modulus. The restrictions upon the allowable transform lengths can be seen to be:

Transforms mod p : The transform length N must divide p - 1
i.e. $N \mid (p - 1)$

Transforms in $GF(q^2)$: The transform length N must divide $q^2 - 1$
i.e. $N \mid (q^2 - 1)$

It can be seen that $N \mid (p - 1)$ and $N \mid (q^2 - 1)$ must both be satisfied for the same transform length to be directly supported for both moduli.

The proposed scheme is to utilize two prime pair moduli (2 primes p, q such that ($|p - q| = 2$). It can be seen that one of the pair (p) must be of type A and the other (q) must be of type B.

For a transform length N, N must divide p - 1 and if p = q + 2 then it is shown that N divides $q^2 - 1$.

$$\text{If} \quad N \mid (p - 1) \longrightarrow N \mid (q + 1) \quad \text{since} \quad p = q + 2$$
$$\text{If} \quad N \mid (q + 1) \longrightarrow N \mid (q^2 - 1) \quad \text{since} \quad q^2 - 1 = (q + 1)(q - 1)$$

Therefore if modulus p supports transforms of length N then modulus $GF(q^2)$ also supports the same transform length provided $p = q + 2$ and $p = 1 \bmod 4$.

Since both 65521 and 65519 are prime this is obviously the criterion which implies that if modulus 65521 supports a transform of length 5040 then modulus $GF(65519^2)$ also supports the same transform length and so both moduli can be seen to support any Winograd transform length.

It is shown in this chapter how complex transforms may be defined over $GF(q^2)$ to perform complex convolutions, and it is shown in chapter 3 how complex convolutions may be performed over Z_p using an extension of Nussbaumer's technique. Therefore the proposed scheme can be used directly for complex convolutions.

The technique can be extended for other prime pairs. For efficient use of b-bit processors moduli must be below and close to 2^b. A search was made for prime pair moduli to support general Winograd transform lengths and additionally which would make efficient use of b-bit processors. Such processors can conviently be constructed using bit slice technology. A figure of efficiency (ρ) is derived, given by $\rho = (pq)^{1/2} \cdot 2^{-b}$, which denotes the usable dynamic range of a b bit register with the prime pair moduli p and q. For certain b values the most suitable moduli were relatively inefficient and so

TABLE 5.3

PRIME PAIR MODULI FOR b-BIT PROCESSORS

b	p	WNmax	$1 - \rho$
12	4021	60	1.9. 10^{-2}
14	16141	60	1.5. 10^{-2}
16	65521	5040	2.4. 10^{-4}
18	241921	5040	7.7. 10^{-2}
18	259381	180	1.1. 10^{-2}
20	957601	5040	8.7. 10^{-2}
20	1048573	252	3.8. 10^{-6}
22	4163041	5040	7.5. 10^{-3}
24	16763041	5040	8.4. 10^{-4}
26	66900961	5040	1.1. 10^{-3}
28	268390081	5040	1.7. 10^{-4}
30	1073656081	5040	8.0. 10^{-5}
32	4294841041	5040	2.9. 10^{-5}

alternative pairs were found which would support restricted Winograd transform lengths, but with a higher ρ. These results are shown in table 5.3 where the prime q is given by $q = p - 2$ and $W_N$max denotes the maximum Winograd transform length allowed for that choice of modulus.

It is worth while to compare the operation counts incurred in performing convolutions by the two different techniques. Let M_N and A_N denote the number of real multiplications and the number of real additions/ subtractions required to perform a real N point Winograd transform algorithm.

For complex convolutions to be performed using Nussbaumer's technique with type A moduli it has been seen in section 3.4 that 2 N-point convolutions, 2N multiplications by 'j', N multiplications by 2^{-1}, N by $(2j)^{-1}$ and 4N additions are required. Let T_{MN} and T_{AN} denote the total number of real multiplications and real additions required to perform a complex convolution by Nussbaumer's technique then

$$T_{MN} = 4M_N + 6N \qquad (5.4.1)$$

$$T_{AN} = 4A_N + 6N \qquad (5.4.2)$$

For arithmetic over $GF(q^2)$ it can be seen that a complex addition requires two real additions. For complex Winograd transforms over $GF(q^2)$ it has been observed that the multiplication coefficients are either purely real or purely imaginary and so these complex muliplications require two real multiplications and no additions. The general multiplications in the transform domain require either (4 + 2) or (3 + 5) real multiplications and additions respectively.

Let T_{MC} and T_{AC} denote the total number of real multiplications and real additions required to compute a complex convolution by a direct transform method over $GF(q^2)$:

$$T_{MC} = 4M_N + 4N \qquad (5.4.3)$$

$$T_{AC} = 4A_N + 2N \qquad (5.4.4)$$

for (4 + 2) multiplication or

$$T_{MC} = 4M_N + 3N \qquad (5.4.5)$$

$$T_{AC} = 4A_N + 5N \qquad (5.4.6)$$

for (3 + 5) multiplication.

In equations (1) to (6) the terms M_N and A_N dominate and so it can be seen that the direct complex transform technique is marginally more efficient. Therefore savings may be made in utilizing a dual complex transform technique with one of the moduli described in table 5.1. However this will result in a slight loss of the dynamic range possible with the choice of M = 65521 and so there appears to be a small trade off acting between these two parameters.

A novel technique has been proposed for complex convolution by number theoretic transform techniques. This technique has been extended to cover other classes of prime pair moduli for use with b bit, bit slice processors. It has also been shown that the computational load required for complex convolution by Nussbaumer's technique compares well with the direct transform techniques proposed in this chapter.

5.5 COMPLEX TRANSFORMS AND REAL TRANSFORMS

It has already been pointed out that in signal processing applications data is sampled using analogue to digital convertors, and so input signals are essentially integer in character. If the results of the processing are to be conveyed back to the real world then digital to analogue convertors are often employed and so output signals can also be integer in character.

There are many examples of such systems, and when the Fourier transform is used for convolution it can be seen that a complex transform technique is being employed to process integer information. There has been much interest in the literature in designing Fourier transform algorithms which are suited to integer (or purely real) signals. For example Silverman (101) describes such a technique for the Winograd Fourier transform algorithm. As has previously been observed the Winograd multiplication coefficients are either purely real or purely imaginary and so these multiplications with real data give answers which are also either purely real or purely imaginary. Hence certain computational savings are possible by keeping flags to denote the data types. It seems that when a complex transform technique is used for real signal processing then the algorithm complexity is increased in order to reduce the processing load.

Brigham (14) has outlined how the complex Fourier transform may be used to perform the transform of two real transfoms simultaneously. If it is desired to compute the discrete Fourier transform of the real time functions h(k) and g(k) then first form.

$$y(k) = h(k) + i\, g(k) \qquad (5.5.1)$$

Compute the Fourier transforms of y(k) and split into real and imaginary parts

$$R(n) + i\ I(n) = \sum_{k=0}^{N-1} y(k)e^{-j2\pi nk/N} \qquad n \in (0, N-1) \qquad (5.5.2)$$

then the transforms of h(k) and g(k) are given by H(n) and G(n) respectively where

$$H(n) = 1/2\ (\ [\ R(n) + R(N-n)\ ]\ +\ i\ [\ I(n) - I(N-n)\ ]\ )\ (5.5.3)$$

$$G(n) = 1/2\ (\ [\ I(n) + I(N-n)\ ]\ -\ i\ [\ R(n) - R(N-n)\ ]\ )\ (5.5.4)$$

This procedure has been interpreted in the number theoretic sense and been shown to work correctly for the simultaneous transform of two real sequences using a complex number theoretic transform. However as has been seen this procedure requires N multiplications by 2^{-1} and 2N additions per N point sequence in addition to the transform and so it cannot be as efficient as a purely real transform technique.

It is logical to attempt to employ a purely real (or integer) transform technique for the processing of real signals. However the maximum transform length available for a Fourier type transform defined in the real domain is limited to 2 with the choice $\alpha = -1$.

In conclusion it can be said that when data is purely real (or integer) then a real transform technique should be employed and when data is complex then a corresponding complex transform technique should be employed. As has been described it is possible to define both real and complex number theoretic transforms. In this respect number theoretic transforms have an advantage over the Fourier transform for convolution. This implies that it is not neccessary for special techniques to be employed for processing of one signal type with a transform defined over a different type of domain and this leads to algorithm simplification for number theeoretic transforms.

CHAPTER 6

ERROR CORRECTING CODES AND NUMBER THEORETIC TRANSFORMS

If information is transmitted over a noisy channel then errors will occur in the received signal. By adding redundant symbols to the transmitted information and by using appropiate error-correcting codes it may be possible to recover the original information without error. It can therefore be seen that such codes are useful when it is inconvenient for a retransmission of information to be performed.

The theory of error correcting codes is beyond the scope of this work. However there are many excellent text books on the subject (10),(70),(104),(121), and a brief introduction to this topic is presented in Appendix F inorder to define the relevant nomenclature and relate this subject to the theory of number theoretic transforms. The purpose of this chapter is to show how a certain class of error correcting code may be encoded and decoded using number theoretic transforms. Several techniques are presented and each is illustrated by a numeric example.

Reed, Truong and Welch (93),(98) have shown that number theoretic transforms may be used for both encoding and decoding a particular class of error correcting code known as a Reed-Soloman (RS) code. A variant on this procedure was used to decode information from the Mariner and Viking space probes (93). Reed et al have proposed RS codes over fields modulo Fermat numbers using radix 2 FFT algorithms. The earlier work in this project has shown that the choice of modulus M = 65521 with Winograd fast transform algorithms is suited to a microprocessor implementation. Therefore Reed's technique has been adapted to using this modulus with appropiate block lengths.

Reed has also shown that non-systematic RS codes are computationally more efficient and these are expliote particularly for longer block lengths with M = 65521. He has also defined codes over complex fields and this approach has also been adapted for a microprocessor implementation.

6.1 THE FAST DECODING OF REED-SOLOMAN CODES USING NUMBER THEORETIC TRANSFORMS

Justensen (38), Reed, Truong and Welch (93),(98) have shown that number theoretic transforms over $GF(F_N)$ of integers modulo a Fermat prime can be used to encode and decode Reed-Soloman codes. In reference (98) a general procedure for this technique is presented.

Let the code length for the RS code be N, and let a codeword be represented by f(x) a polynomial of degree N - 1 over $GF(F_N)$. The generator polynomial of f(x) is defined by g(x):

$$g(x) = \prod_{i=1}^{d-1} (x - \alpha^i) \qquad (6.1.1)$$

where $d - 1 = n - k$ and α is an element of order N. The resultant RS code with N symbols, f(x), is a multiple of the generator polynomial and is composed of $d - 1$ check symbols and the $N - (d - 1)$ information symbols. If t is the number of errors the code will correct, then for an RS code it has been shown that

$$d > 2t + 1 \qquad (6.1.2)$$

Suppose that the code $f(x) = f_0 + f_1 x + .. + f_{N-1} x^{N-1}$ is transmitted over a noisy channel. The received code

$r(x) = r_0 + r_1x + .. + r_{N-1}x^{N-1}$ is composed of the original code with the addition of possible errors. i.e.

$$r(x) = f(x) + e(x) \qquad (6.1.3)$$

where $e(x) = e_0 + e_1x + .. + e_{N-1}x^{N-1}$ is the error locator polynomial. Upon receiving the message r(x) one may start to decode the message symbols by forming the syndrome by taking the transform over $GF(F_N)$ of r(x).

$$S(k) = \sum_{i=0}^{N-1} r(i)\ \alpha^{ik} \qquad (6.1.4)$$

$$= \sum_{i=0}^{N-1} f(i)\ \alpha^{ik} + \sum_{i=0}^{N-1} e(i)\ \alpha^{ik}$$

$$= \quad F(k) \quad + \quad E(k) \qquad (6.1.5)$$

Since F(x) is a multiple of g(x) then $f(a^i) = 0$ for i e (1, d-1) and hence S(k) = E(k) for k e (1, d-1). It is this result which gives an easy technique for determining the error locator polynomial from the syndrome S(k).

It is shown in reference (98) that the transform of the error locator polynomial E(k) must satisfy the following relation.

$$S_{j+t} - \lambda_1.S_{j+t-1} + \ldots + (-1)^t\lambda_t.S_j = 0 \quad \text{for } j < t \qquad (6.1.6)$$

and $$E_{j+t} - \lambda_1.E_{j+t-1} + \ldots + (-1)^t\lambda_t.E_j = 0 \quad \text{for } j < t \qquad (6.1.7)$$

Berlekamps iterative algorithm (99), (112), (10) may be employed to determine the λ_i from equation (6) and these may be subsituted in equation (7) to derive the transform of the error polynomial.

The error locator polynomial may then be determined by taking the inverse transform of E_k and then finally the corrected codeword r_c may be obtained using

$$r_c(x) = r(x) - e(x) \tag{6.1.8}$$

An example of the procedures required is presented for a two error correcting (16, 12) Reed-Soloman code defined over GF(17).

In GF(17) the element $\alpha = 6$ is of order 16 and hence the corresponding generator polynomial can be seen to be

$$g(x) = -2 + 2x + x^2 - 7x^3 + x^4 \tag{6.1.9}$$

Given the information polynomial I(x):

$$\begin{aligned} I(x) = {} & 5x^4 + 6x^5 + 7x^6 + 8x^7 - 8x^8 - 7x^9 \\ & - 6x^{10} - 5x^{11} - 4x^{12} - 3x^{13} - 2x^{14} - x^{15} \end{aligned} \tag{6.1.10}$$

Then the corresponding codeword f(x) may be determined by dividing g(x) into I(x) and subtracting the remainder from I(x) to give f(x).

Hence

$$\begin{aligned} f(x) = {} & -3 + x - 5x^2 - 6x^3 + 5x^4 + 6x^5 + 7x^6 + 8x^7 \\ & - 8x^8 - 7x^9 - 6x^{10} - 5x^{11} - 4x^{12} - 3x^{13} - 2x^{14} - x^{15} \end{aligned}$$

or

$$f(x) = I(x) + (-3 + x - 5x^2 - 6x^3) \tag{6.1.11}$$

This procedure guarantees that f(x) is a multiple of g(x). Let two errors occur during transmission:

$$\begin{aligned} r(x) = {} & -3 + x - 2x^2 - 6x^3 + 10x^4 + 6x^5 + 7x^6 + 8x^7 \\ & - 8x^8 - 7x^9 - 6x^{10} - 5x^{11} - 4x^{12} - 3x^{13} - 2x^{14} - x^{15} \end{aligned}$$

or

$$r(x) = f(x) + (3x^2 + 5x^4) \qquad (6.1.12)$$

At the receiver the transform of r(x) is taken to give R(x):

$$R(x) = 2 - 8x + 7x^2 + 4x^3 + 2x^4 + 3x^5 - 5x^6 - 4x^7 - x^8 + x^9 + 8x^{10} - 6x^{11} - 6x^{12} - 7x^{13} + 7x^{14} + 6x^{15} \qquad (6.1.13)$$

If r(x) were a valid codeword then $R(x^i)$ i ϵ (1,4) would have been zero. As this is not the case the λ_i in equation (6) must be determined. From (98), if two errors have occured then

$$\lambda_1 = (S(1) . S(4) - S(3) . S(2)) . D^{-1} \qquad (6.1.14)$$

and $$\lambda_2 = (S(2) . S(4) - S(3)^2) . D^{-1} \qquad (6.1.15)$$

where $$D = S(1) . S(3) - S(2)^2 \qquad (6.1.16)$$

If only one error has occured then D is zero and λ_1 is given by

$$\lambda_1 = S(4) . (S(3))^{-1} \qquad (6.1.17)$$

Therefore the calculation of D given in equation (16) shows whether one or two errors have occured and the appropiate λ_i may be determined. In this particular example $\lambda_1 = 6$ and $\lambda_2 = 8$ and so the transform of the error sequence E(x) may be derived.

$$E(x) = 8 - 8x + 7x^2 + 4x^3 + 2x^4 - 3x^5 + 7x^7 + 8x^8 - 8x^9 + 7x^{10} + 4x^{11} + 2x^{12} - 3x^{13} + 7x^{15} \qquad (6.1.18)$$

By taking the inverse transform the error locator polynomial is obtained.

$$e(x) = 3x^2 + 5x^4$$

and so both errors have been successfully corrected.

Reed, Truong and Welch (93), (98), (99) have observed that two problems arise when using Fermat number transforms in this type of application.

In order to represent a number in GF(F_N) $2^N + 1$ bits are required, and as has been shown previously, 2^N bits are not sufficient for a unique representation of the elements of GF(F_N). Reed and Truong suggest that the information symbols be restricted to the range 0 to 2^{2^N}-1. After encoding the check symbols occur in the range 0 to 2^{2^N}. If the symbol 2^{2^N} is required then it should be deliberatly corrupted, for example to zero. The transform decoder will correct such an error automatically. These errors deliberatly decrease the codes error correcting capability, but as they point out the probability of such occasions is low.

While retaining simple arithmetic for the computation of the transform it has been shown that the transform lengths available with Fermat number moduli are restricted and this constrains the permissible block lengths for this technique.

It should be observed that for a microprocessor implementation of Fermat number transforms truncation errors can still occur during the computation of the transform itself, and this would lead to serious errors in the decoder algorithm.

The technique of using number theoretic transforms for decoding Reed-Soloman codes over GF(q) can be applied for moduli other than Fermat numbers. However the example has shown that during the computation of the λ_i in equation (6) multiplicative inverses are required. For composite moduli not all non-zero elements have multiplicative inverses, and for this reason Reed and Truong suggest that only prime moduli be considered.

In the light of the preceeding work for developing algorithms suitable for a microprocessor implementation the technique was adapted to using the prime modulus M = 65521 with Winograd fast transform algorithms. (16, 12) and (60, 56) two error correcting RS codes were developed in FORTRAN on an IBM 370 using the fast transform algorithms previously derived. These were tested and found to correct errors successfully.

The use of the modulus M = 65521 does represent a slight loss of dynamic range possible with 16 bit arithmetic. This limitation would not appear to affect drastically the transmission of character information where it is unlikely that all 256 8 bit patterns would be required, however this may cause a problem for the transmission of pure binary data. It is apparent that since M = 65521 supports any Winograd transform length then the block length restriction of Fermat number transforms is removed.

This application of number theoretic transforms to decoding RS codes relies upon valid codewords generating appropiate zeros in the transform domain. When this is not the case then the transform domain information gives an efficient technique for determining the error locator polynomial. It is apparent that in this context, number theoretic transforms are not being used for the convolution of signals with physical significance. Therefore the discussions presented in section 4.1 on the techniques to avoid modular overflow do not apply in this context, and this allows the free use of the entire dynamic range of the modulus.

These codes are designed to correct errors in the codeword symbols. Since each symbol contains 16 bits it can be seen that these codes have good burst error correcting ability. Provided that the error bursts are not too prolonged then this technique is superior to a bit orientated code where bit matrix transpositions have to be employed to give good burst error correcting ability.

Reed, Truong, Welch (99), and Scholtz (112) have shown how the determination of the λ_i in equation (6) by Berlekamp's algorithm may be performed by a continued fractions technique and this applies particularly for codes designed to correct a large number of errors. The programmed examples with modulus M = 65521 were designed only for a maximum of two error corrections as the determination of the λ_i becomes more difficult when a larger number of errors are required to be corrected.

The encoding procedure requires a division of a polynomial of order k by a polynomial of order n - k and this is not a trivial operation. Number theoretic transforms can be used for polynomial multiplication and division. However these general procedures cannot be applied in this case since the generator polynomial can be considered to be a 'zero divisor', as it contains zeros in the transform domain. Hence a full polynomial division algorithm has to be employed.

It has been the purpose of this section to show how number theoretic transforms can be used for decoding Reed-Soloman error correcting codes. The technique has been adapted to make it more suitable for a microprocessor implementation using the modulus M = 65521 with fast Winograd transform algorithms.

6.2 NON SYSTEMATIC REED-SOLOMAN CODES

Reed, Truong and Welch (98) have shown that non-systematic Reed-Soloman (NSRS) codes can also be defined over GF(q) where q is prime. This class of code is defined in reference (70), however it is hoped that the example which is presented is sufficient as explanation.

Let us consider the information polynomial i(x) with coefficients

$$i(x) = i_0 + i_1.x + ... + i_k x^{k-1} + 0.x^k + ... + 0.x^{n-1} \qquad (6.2.1)$$

As can be seen this polynomial has k information symbols and the coefficients for x^k to x^{n-1} are zero. Let us take the transform of i(x)

$$I(k) = \sum_{j=0}^{N-1} i(j)\ \alpha^{jk} \qquad (6.2.2)$$

It was observed in the previous section that the generator polynomial can loosely be termed a 'zero-divisor' because for systematic Reed-Soloman (SRS) codes valid codewords have zeros in the transform domain. This result arises because the generator polynomial defined by g(x):

$$g(x) = \prod_{i=0}^{d-1} (x - \alpha^i) \qquad (6.2.3)$$

is composed of factors which are zero for powers of α. It is therefore not surprising that these factors interact with the transform operator to produce zeros in the transform domain. Although it is not proved here, this result does imply that the transform function I(x) is also a multiple of the generator polynomial and so I(x) is also a valid Reed-Soloman codeword. This type of code is called a non systematic Reed-Soloman code.

The systematic Reed-Soloman codes presented in the previous section have the message symbols of valid codewords identical with the corresponding symbols in the information polynomial. However non systematic Reed-Soloman codes do not normally have any message symbols identical with the corresponding information polynomial symbols. This makes the NSRS codes unfamiliar, but as will become apparent, computational saving may be made by this approach.

The polynomial I(x) is transmitted over the channel. Let the received polynomial be R(x) where

$$R(x) = I(x) + E(x) \qquad (6.2.4)$$

where E(x) will be seen to be the transform of the error polynomial e(x). The syndrome is formed by taking the inverse transform of R(x) giving

$$s_j = r_j = i_j + e_j \qquad (6.2.5)$$

However $\quad i_j = 0 \quad$ for $j \in (k, N-1)$ and so

$$s_j = e_j \qquad j \in (k,k\ N-1) \qquad (6.2.6)$$

The same procedure may be now adopted as for SRS codes. Determine the λ_i from

$$s_{N-j} - \lambda_1 \cdot s_{N-j-1} + \dots + (-1)^t \lambda_t \cdot s_{N-j-t} = 0 \qquad \text{for } j < t \qquad (6.2.7)$$

and substitute in

$$e_{N-j} - \lambda_1 \cdot e_{N-j-1} + \dots + (-1)^t \lambda_t \cdot e_{N-j-t} = 0 \qquad \text{for } j > t \qquad (6.2.8)$$

to obtain the error pattern. This may be substituted in equation (5) to give

$$i_x = r_x - e_x \qquad (6.2.9)$$

An example is presented for a two error correcting (16, 12) NSRS code defined over GF(17). Let the information polynomial i(x) be

$$\begin{aligned} i(x) = {} & 1 + 2x + 3x^2 + 4x^3 + 5x^4 + 6x^5 \\ & + 7x^6 + 8x^7 - 8x^8 - 7x^9 - 6x^{10} - 5x^{11} \qquad (6.2.10) \end{aligned}$$

The encoded sequence I(x) is formed by taking the transform of i(x)

$$\begin{aligned} I(x) = {} & - 7 - 6x + 7x^2 - x^3 + x^4 + 3x^5 + 7x^6 + 8x^7 \\ & - 6x^8 + 8x^9 + 8x^{10} - 2x^{11} + x^{12} - 8x^{13} - 2x^{14} + 2x^{15} \\ & \qquad (6.2.11) \end{aligned}$$

Let two errors occur during transmission and so R(x):

$$\begin{aligned} R(x) = {} & - 7 - 4x + 7x^2 + 3x^3 + 4x^4 + 3x^5 + 7x^6 + 8x^7 \\ & - 6x^8 + 8x^9 + 8x^{10} - 2x^{11} + x^{12} - 8x^{13} - 2x^{14} + 2x^{15} \end{aligned}$$

or

$$R(x) = I(x) + (2x + 4x^3) \qquad (6.2.12)$$

The syndrome is formed by taking the inverse transform of the received sequence

$$\begin{aligned} s(x) = &- 5 + 7x - 7x^2 - 4x^3 - 3x^4 + 6x^5 - 8x^6 + 0x^7 \\ &- 2x^8 + 5x^9 + 4x^{10} + 3x^{11} + 8x^{12} + 0x^{13} - 2x^{14} + 8x^{15} \end{aligned} \qquad (6.2.13)$$

The λ_i in equation (7) may be determined from

$$\lambda_1 = (s(z) \circ s(z + 3) - s(z + 2) \circ s(z + 1)) \circ D^{-1} \qquad (6.2.14)$$

$$\lambda_2 = (s(z + 1) \circ s(z + 3) - s(z + 2)^2) \cdot D^{-1} \qquad (6.2.15)$$

where $D = s(z) \cdot s(z + 2) - s(z + 1)^2$ and $z = N - 4$

If only one error had occured then D would be zero and λ_1 is given by

$$\lambda_1 = s(z + 3) \cdot (s(z + 2))^{-1} \qquad (6.2.16)$$

It should be observed that equations (14) to (16) are of the same form as the corresponding equations (6.1.14) to (6.1.17) for systematic Reed-Soloman codes. In both cases the λ_i are determined from the elements where four zeros are expected (elements 1 to 4 in the transform domain for SRS and elements N - 4 to N - 1 in the 'time' domain for NSRS codes).

In the example two errors have occured giving $\lambda_1 = -4 = \lambda_2$. From this the error sequence may be determined.

$$\begin{aligned} e(x) = &- 6 + 5x + 7x^2 - 8x^3 - 8x^4 + 0x^5 + 2x^6 - 8x^7 \\ &+ 6x^8 - 5x^9 - 7x^{10} + 8x^{11} + 8x^{12} + 0x^{13} - 2x^{14} + 8x^{15} \end{aligned} \qquad (6.2.17)$$

The information polynomial may now be recovered using

$$i(x) = s(x) - e(x) \tag{6.2.18}$$

It can be seen that NSRS codes only require a number theoretic transform for encoding and the decoding requires an inverse transform and a determination of the error polynomial using Berlekamp's algorithm. In comparison SRS codes require a polynomial division for encoding and both a forward and an inverse transform for decoding, in addition to a similar determination of the error polynomial. Therefore the NSRS codes require less computational effort and are thus more attractive.

This class of error correcting code has been implemented with the modulus M = 65521 in FORTRAN on an IBM 370. The following two error correcting codes have been developed and tested with accurate results (16, 12), (60, 56), and (5040, 5036). The transform algorithms have used the Winograd general N transform program described in chapter 3.

6.3 REED-SOLOMAN CODES OVER COMPLEX FIELDS

Reed, Truong and Welch (98) have proposed both systematic and non-systematic Reed-Soloman codes over quadratic Mersenne fields. These fields support radix 2 FFT algorithms (86) and hence they can be used to provide an efficient error correcting code technique. Since Mersenne numbers do not suffer the truncation problems of Fermat number moduli, this class of transform seems promising.

A numeric example is presented for a two error correcting (16, 12) NSRS code defined over GF(7^2). The general procedure followed is identical to that presented for NSRS codes over GF(q).

Let the information polynomial be i(x):

$$\begin{aligned} i(x) = {} & (1 - 1i') + (2 - 2i')x + (3 - 3i')x^2 + (-3 + 3i')x^3 \\ & + (-2 + 2i')x^4 + (-1 + 0i')x^5 + (0 + 0i')x^6 + (1 - 0i')x^7 \\ & + (2 - 2i')x^8 + (3 - 3i')x^9 + (-3 + 3i')x^{10} + (-2 + 2i')x^{11} \end{aligned} \tag{6.3.1}$$

Take the transform of i(x) over GF(7^2) with $\alpha = (2 + 3i')$ which is an element of order 16.

$$\begin{aligned} I(x) = {} & (1 - 1i') + (3 + 0i')x + (1 - 1i')x^2 + (1 - 2i')x^3 \\ & + (0 - 2i')x^4 + (2 - 0i')x^5 + (2 - 2i')x^6 + (0 - 3i')x^7 \\ & + (1 - 1i')x^8 + (-1 + 0i')x^9 + (2 - 2i')x^{10} + (1 + 0i')x^{11} \\ & + (2 + 0i')x^{12} + (0 - 0i')x^{13} + (1 - 0i')x^{14} + (0 + 0i')x^{15} \end{aligned} \tag{6.3.2}$$

Let two symbol errors occur during transmission and so

$$R(x) = I(x) + (2 + 0i')x + (4 + 0i')x^3 \tag{6.3.3}$$

The syndrome is formed by taking the inverse transform of the received signal

$$\begin{aligned} s(x) = {} & (-3 + 0i') + (-1 + 2i')x + (-1 + 0i')x^2 + (1 + 0i')x^3 \\ & + (-1 + 0i')x^4 + (-3 + 0i')x^5 + (0 + 0i')x^6 + (2 + 2i')x^7 \\ & + (-1 - 3i')x^8 + (-1 + 0i')x^9 + (1 - 0i')x^{10} + (1 - 3i')x^{11} \\ & + (-1 + 0i')x^{12} + (2 + 0i')x^{13} + (0 - 0i')x^{14} + (-1 + 3i')x^{15} \end{aligned} \tag{6.3.4}$$

The λ_i may be determined using equation (6.2.7) with arithmetic defined over GF(7^2). This gives $\lambda_1 = (2 - 2i')$ and $\lambda_2 = (0 + i')$. The error pattern may now be calculated in the manner of equation (6.2.8) giving

$$\begin{aligned} e(x) = {} & (3 + 0i') + (-3 - 3i')x + (3 - 3i')x^2 + (-3 - 2i')x^3 \\ & + (1 - 0i')x^4 + (-2 - 0i')x^5 + (0 + 0i')x6 + (1 - 3i')x^7 \\ & + (-3 - 0i')x^8 + (3 + 3i')x^9 + (-3 + 3i')x^{10} + (3 + 2i')x^{11} \\ & + (-1 + 0i')x^{12} + (2 + 0i')x^{13} + (0 - 0i')x^{14} + (-1 + 3i')x^{15} \end{aligned} \tag{6.3.5}$$

and finally the information polynomial may be recovered using

$$i(x) = s(x) - e(x) \tag{6.3.6}$$

In chapter 5 it was shown that transforms over $GF(65519^2)$ are particularly suited to a microprocessor implementation of complex number theoretic transforms using Winograd fast transform algorithms. It has been shown that NSRS codes require less computation than SRS codes and so algorithms were developed using the mainframe facilities for the following two error correcting NSRS codes (16, 12) and (60, 56) defined over $GF(65519^2)$. The algorithms were tested and found to correct errors successfully.

If complex RS codes (CRS) defined over $GF(65519^2)$ are compared with those defined over GF(65521) (RRS) it can be seen that the symbol bit length for the complex transform is twice that of the normal transform. Therefore provided errors occur in bursts, the error correcting ability of a t error correcting length N CRS code has comparable correcting ability with a 2t error correcting length 2N RRS code. Since the determination of the λ_i for t errors is much simpler for than for 2t errors it can be seen that the complex codes are suited to correcting longer error bursts than the normal codes. It may also be deduced that the normal code is more suited to cases where the errors occur in more frequent shorter bursts.

CHAPTER 7

CONCLUSIONS AND SUGGESTIONS FOR FURTHER WORK

Previous work has shown that the Fermat number transform has certain optimum properties. These transforms can be computed by radix 2 FFT algorithms requiring only bit shifts and additions and can effectively be implemented with subtract in carry hardware.

However two limitations have become apparent. Fermat moduli contain $2^b + 1$ elements and since only 2^b distinct elements can be retained in b bit memories there exists a representation problem. If one wishes to store the element 2^b then a truncation error will occur. However elements of the finite rings over which number theoretic transforms are defined do not have physical significance, and so a truncation error could generate serious errors in an entire data block. The probability of error for 16 bit processors was found to be unnacceptably high and so alternative number theoretic transforms were considered for a microprocessor implementation. Fermat number moduli also suffer from the limitation that the optimum transform length is proportional to the wordlength and for certain applications this transform length is too short. For a purely hardware realization of a Fermat number transform the hardware should be of the subtract in carry type. It was quickly realized that if one tried to imitate this mode of arithmetic with a microprocessor implementation then the resultant computation would not be efficient.

Microprocessors are now becoming available with fast multiply instructions, and for those that do not have this facility multiply peripheral chips may be attached. This trend leads toward a reappraisal and relaxing of the basic constraints which govern the choice of a particular number theoretic transform. Therefore broader classes of number theoretic transform were considered allowing non simple arithmetic.

While work was being conducted in this area a new class of efficient Fourier transform algorithms were published. These Winograd algorithms provide efficient algorithms for a selection of transform lengths between 2 and 5040. Since the Fourier transform and number theoretic transforms are alike in structure, then in principle any Fourier transform algorithm may be applied to number theoretic transforms. Therefore it was considered feasible to apply the Winograd algorithms to number theoretic transforms. Bailey was working concurrently using this idea.

A search was conducted for moduli which would support Winograd transform lengths. The search revealed that M = 65521 supports all the transforms lengths achievable by combinations of the published Winograd short length transform algorithms. This modulus additionally incurs little redundancy in the use of 16 bit arithmetic.

A general N Winograd number theoretic transform program was written and this was found to be a very useful tool for the development of algorithms for specific transform lengths. Transform algorithms were developed for lengths 60 and 240. The multiplication coefficients required in the computation of these transform algorithms are not as simple as those required for Fermat number transforms. However for microprocessors with fast multiply instructions this is no longer a limitation. Therefore such microprocessor systems would appear to be able to compute such transforms without serious penalty.

A length 60 transform algorithm was implemented on an Intel 8080 microprocessor. Since this microprocessor does not have the fast multiply instruction, nor was a fast multiplier chip available, the transform computation was slow.

Preliminary investigations have shown that even on a 16 bit processor with hardware multiply such as a Texas 9900 the added computational load required for performing modular arithmetic can be a serious penalty when compared with simple integer arithmetic (i.e. modulo 2^{16}). Therefore a special hardware modular multiplier was designed to interface to such processors.

The main application of number theoretic transforms is for convolution of signals with physical significance. Provided modular arithmetic be employed throughout then the final answers will be exact mod M. However it is neccessary that the final answer also has physical significance and so the dynamic range of the input signals is constrained. This limitation tends to degrade the filter response obtainable with a given modulus, and it is possible that the filter response may not achieve the design limits set. In such cases a larger modulus should be used.

A direct implementation of a larger modulus requires arithmetic with greater precision. An alternative solution is provided by using the Chinese remainder theorem to combine the simultaneous results with respect to two or more relatively prime moduli. This is a practicable solution which lends itself to a parallel processor architecture.

Therefore a computer search was conducted for moduli which would pair with M = 65521 over specific Winograd transform lengths. In cases where the dual modulus incurs great redundancy in the use of 16 bit arithmetic, Agarwal and Cooley's multidimensional convolution may be employed using shorter transform lengths with more efficient moduli.

Reed and Truong have described how number theoretic transforms may be defined over complex fields and in particular over complex or quadratic Mersenne fields. These fields support radix 2 FFT algorithms and so, unlike their simple counterpart, they are efficient. In the light of the previous work a search was conducted for complex moduli which would be suitable for a microprocessor implementation using Winograd fast transform algorithms. The search reveled that GF(65519^2) is the most suitable modulus. A search was made for dual complex moduli which can be combined with 65519 using the Chinese remainder theorem.

The optimum moduli for these two classes of number theoretic transform are 65521 and 65519. It was noted that these constitute a prime pair and this result was generalised for other bit lengths.

This thesis concludes with a chapter describing how number theoretic transforms may be used to encode and decode a class or error correcting code known as Reed-Soloman codes. It is shown that this provides an efficient technique.

In summary microprocessors have been shown to be suitable processors for the computation of number theoretic transforms. Since the price of microprocessor systems is dropping it can be seen that they can provide cheap and efficient signal processors using number theoretic transforms for convolution. Since it has not been possible to implement the algorithms using a processor for which they have been designed no details of execution time are available.

SUGGESTIONS FOR FURTHER WORK

During the course of this project various signal processing algorithms have been designed and tested for an eventual implementation on a microprocessor with either a fast multiply instruction or fast multiplier peripheral chips. However there was insufficient time for these algorithms to be fully implemented. It is therefore suggested that this be accomplished. Preliminary trials indicate that modular integer arithmetic is substantially slower than normal integer arithmetic particularly for the multiply operation. Therefore a fast modular multiplier has been desgined to alleviate this problem and if it proves necessary this should be used.

The transform algorithms have been designed with a single host processor in mind. However in certain circumstances it has been shown that use can be made of the Chinese remainder theorem in order to effect a modulus with greater dynamic range without the neccesity of extended precision arithmetic. This technique is particularly suited to a parallel processor architecture. Let us consider a two modulus procedure. Two fast processors may conviently conduct the signal processing, one for each modulus, and a slower processor would be sufficient to perform the Chinese remainder combination of the results using the memory look up technique which has been described.

The transform algorithms can also be reconfigured to suit a multiprocessor architecture. The basic Winograd algorithm can be divided into the following stages. After an initial permutation, input additions and subtractions are performed and this is followed by a series of multiplications. Output additions and subtractions are then performed and finally another permutation is required. The bulk of the processing occurs in the three inner stages. The entire process is well suited to a pipeline architecture where single processors can conviently perform the tasks at each stage. It is therefore suggested that this avenue be explored.

Agarwal and Cooley's multidimensional convolutional algorithms may be used to compute convolution directly. One may therefore either compute convolution by transform techniques or by using these algorithms. In a comparison between these 2 techniques it is apparent that in operation count terms the direct method would be more efficient for short convolution lengths than the transform technique. Similarly for greater convolution lengths the transform technique is more efficient. There therfore must exist a crossover point beyond which the transform technique would become more effecient. It is interesting to note that the modulus for the transform technique is constrained to guarantee that the transform support the cyclic convolution property. With number theoretic transforms moduli which are a power of 2 cannot be used and so full modular arithmetic has to be employed. For rectangular convolution this restriction does not apply and so ordinary b bit integer arithmetic may be used. It has been seen that a penalty is incurred when using modular integer arithmetic and this would therefore affect the comparison between the two techniques. It is suggested that this relation be examined more closely.

Attention is drawn to the fact that normal b bit integer arithmetic is in fact arithmetic modulo 2^b. Some of the rectangular algorithms require division by 3, 5, 7 and 9. These operations may be replaced by multiplication by the elements 3^{-1}, 5^{-1}, 7^{-1} and 9^{-1} modulo 2^b. This does not result in any truncation error. However divisions by 2, 4 and 8 must be accomplished by normal shifting techniques since these elements do not have multiplicative inverses modulo 2^b. This will result in a truncation error but Agarwal and Cooley reason that this error would be less than for a corresponding fixed point arithmetic. This whole area needs to be examined more deeply particularly with a view to a short convolution length implementation using microprocessors.

It has also been shown that number theoretic transforms may be used to encode and decode Reed-Soloman error correcting codes. Reed, Truong and Welch claim that such a technique provides the fastest procedure for such coding. In the light of this it seems worthwhile that this whole topic be explored perhaps in a more mathematical and rigorous manner than was adpoted in this work.

The major part of this work is concerned with developing a suitable microprocessor implementation of number theoretic transforms with moduli in the order of 2^{16}. Microprocessors will soon become available with the ability to perform efficient 32 bit integer arithmetic. It is therefore suggested that this work be reviewed for an implementation with such processors.

It can be deduced from table 5.1 that the 32 bit moduli would be more efficient than the 16 bit moduli derived earlier. However the main advantage will be that unlike the 16 bit moduli such moduli will not suffer from the limitation of insufficient dynamic range for typical signal processing tasks. The techniques of Chinese remainder combination with dual moduli would therefore appear to be redundant. However the applications of the Chinese remainder theorem are certainly ubiquitous, and it is felt that the Chinese remainder theorem would still make an effective contribution to signal processing, perhaps more in the realm of algorithm development. In particular in the field of error correcting codes there would still be distinct advantages to employing the Chinese remainder theorem to achieve better correcting ability for comparable decoder complexity. This is an area which would benifit further investigation.

Eastwood and Jesshope (25) have shown that number theoretic transforms may be applied to solving partial differential equations; and Derome (21) has shown that number theoretic transforms are suited to the convolving of long arrays as required in picture processing and electron microscopy in particular. It is suggested that both these topics be explored more thoroughly.

APPENDIX A

```
C***********************************************************
C
C MODULAR ARITHMETIC SUBROUTINE LIBRARY
C
C***********************************************************
C
C THIS SUBROUTINE SETS UP THE CURRENT MODULUS
C IN COMMON BLOCK /MD/
C
      SUBROUTINE MSET(M)
      IMPLICIT INTEGER(A-Z)
      COMMON/MD/MD,MD2
      MD=M
      MD2=M/2
      RETURN
      END
C***********************************************************
      FUNCTION MODO(F)
C
C THIS IS THE BASIC MODULO ARITHMETIC FUNCTION
C
      INTEGER F
      COMMON /MD/MD,MD2
      J=0
      IF(F.LT.0)J=-1
      J=J+F/MD
      MODO=F-J*MD
      IF(MODO.GT.MD2)MODO=MODO-MD
      RETURN
      END
C***********************************************************
      INTEGER FUNCTION INV(X)
C THIS FUNCTION RETURNS THE MODULAR INVERSE OF AN INTEGER
C USING INV(X)=X**E-1, WHERE E IS THE VALUE OF EULER'S
C FUNCTION FOR THE MODULUS MD
C
      IMPLICIT INTEGER(A-Z)
      COMMON/MD/MD,MD2
      INV=EXPM(X,EULER(MD)-1)
      IF(MODO(INV*X) .EQ.1)RETURN
      PAUSE ' ERROR IN INV'
      INV=0
      RETURN
      END
C***********************************************************
C THIS FUNCTION RETURNS ALPHA**X MOD M
C IT USES A QUICK TECHNIQUE FOR DETERMINING THE POWERS
C
      INTEGER FUNCTION EXPM(ALPHA,X)
      IMPLICIT INTEGER(A-Z)
      Y=X
      SUM=1
      PROD=MODO(ALPHA)
```

APPENDIX A

```
  100 IF(MOD(Y,2).NE.0)SUM=MODO(SUM*MOD(Y,2)*PROD)
      Y=Y/2
      PROD=MODO(PROD*PROD)
      IF(Y.GE.1)GOTO 100
      EXPM=SUM
      RETURN
      END
C*****************************************************************
C
C THIS FUNCTION RETURNS THE VALUE OF O(M) (THE BIG O FUNCTION)
C AS DEFINED BY AGARWAL &  BURRUS
C
      INTEGER FUNCTION OH(M)
      IMPLICIT INTEGER(A-Z)
      INTEGER FAC(2,255),GFAC(2,255),HFAC(2,255)
      LOGICAL FLAG(255)
      COMMON/MFACT/FAC
      COMMON/UPS/HFAC,FLAG,H
      CALL FACTOR(M,FAC,F)
      IF(F.EQ.1)GOTO 900
      CALL FACTOR(FAC(1,1)-1,HFAC,H)
      DO 100 I=2,F
      CALL FACTOR(FAC(1,I)-1,GFAC,G)
      DO 200 J=1,H
      FLAG(J)=.TRUE.
      DO 300 K=1,G
      IF(HFAC(1,J).EQ.GFAC(1,K))GOTO 350
  300 CONTINUE
      FLAG(J)=.FALSE.
      GOTO 360
  350 IF(HFAC(2,J).GT.GFAC(2,K))HFAC(2,J)=GFAC(2,K)
  360 CONTINUE
  200 CONTINUE
      CALL UPSET
      IF(H.EQ.0)GOTO 910
  100 CONTINUE
      OH=FPWR(HFAC,H)
      RETURN
  900 OH=FAC(1,1)-1
      RETURN
  910 OH=1
      RETURN
      END
C*****************************************************************
C
C THIS SUBROUTINE IS USED IN OH(M)
C
      SUBROUTINE UPSET
      IMPLICIT INTEGER(A-Z)
      INTEGER HFAC(2,255)
      LOGICAL FLAG(255)
      COMMON/UPS/HFAC,FLAG,H
      J=0
      DO 100 I=1,H
```

APPENDIX A

```
  100 IF(FLAG(I))J=J+1
      IF(J.EQ.O)GOTO 500
      H2=H-1
      DO 200 I=1,H2
      IF(FLAG(I))GOTO 190
      DO 300 K=I,H2
      FLAG(K)=FLAG(K+1)
      HFAC(1,K)=HFAC(1,K+1)
      HFAC(2,K)=HFAC(2,K+1)
  300 CONTINUE
  190 CONTINUE
  200 CONTINUE
  500 H=J
      RETURN
      END
C*************************************************************
C THIS FUNCTION IS USED IN OH(M)
      INTEGER FUNCTION FPWR(FAC,F)
      INTEGER FAC(2,255),F
      FPWR=1
      DO 100 I=1,F
      FPWR=FPWR*FAC(1,I)**FAC(2,I)
  100 CONTINUE
      RETURN
      END
C*************************************************************
C THIS FUNCTION INTERPRETS IN THE NUMBER THEORETIC SENSE
C THE TRIGONOMETRICAL FUNCTIONS COSINE AND SINE
C
      INTEGER FUNCTION MCOS(U,P)
      IMPLICIT INTEGER(A-Z)
      COMMON/MCOSSN/LASTU,VAL,RES
      COMMON/J/J
      INTEGER VAL(4,2),POWER(4),RES(9)
      ROW=1
      GOTO 100
      ENTRY MSIN(U,P)
      ROW=2
  100 IF(U.EQ.LASTU)GOTO 300
      POWER(1)=U
      DO 150 I=2,4
  150 POWER(I)=MODO(POWER(I-1)*U)
      INV2=INV(2)
      INV2J=INV(2*J)
      DO 200 I=1,4
      VAL(I,1)=MODO(INV2*(POWER(I)+INV(POWER(I))))
      VAL(I,2)=MODO(INV2J*(POWER(I)-INV(POWER(I))))
  200 CONTINUE
      LASTU=U
  300 MCOS=0
      IF(P.LE.4)MCOS=VAL(P,ROW)
      MSIN=MCOS
      RETURN
      END
```

APPENDIX A

```
C*****************************************************************
C
C THIS SUBROUTINE PROVIDES A QUICK FACTORING TECHNIQUE
C
      SUBROUTINE FACTOR(Y,FAC,F)
      IMPLICIT INTEGER(A-Z)
      INTEGER FAC(2,255)
      INTEGER*2 PRIMES(168)
      DATA PRIMES/     2,     3,     5,     7,    11,    13,    17,
     1    19,    23,    29,    31,    37,    41,    43,    47,    53,
     1    59,    61,    67,    71,    73,    79,    83,    89,    97,
     1   101,   103,   107,   109,   113,   127,   131,   137,   139,
     1   149,   151,   157,   163,   167,   173,   179,   181,   191,
     1   193,   197,   199,   211,   223,   227,   229,   233,   239,
     1   241,   251,   257,   263,   269,   271,   277,   281,   283,
     1   293,   307,   311,   313,   317,   331,   337,   347,   349,
     1   353,   359,   367,   373,   379,   383,   389,   397,   401,
     1   409,   419,   421,   431,   433,   439,   443,   449,   457,
     1   461,   463,   467,   479,   487,   491,   499,   503,   509,
     1   521,   523,   541,   547,   557,   563,   569,   571,   577,
     1   587,   593,   599,   601,   607,   613,   617,   619,   631,
     1   641,   643,   647,   653,   659,   661,   673,   677,   683,
     1   691,   701,   709,   719,   727,   733,   739,   743,   751,
     1   757,   761,   769,   773,   787,   797,   809,   811,   821,
     1   823,   827,   829,   839,   853,   857,   859,   863,   877,
     1   881,   883,   887,   907,   911,   919,   929,   937,   941,
     1   947,   953,   967,   971,   977,   983,   991,   997/
      IF(Y.GE.1000000)PAUSE 'INPUT TO FACTOR GREATER THAN 1000,000'
      X=Y
      SCAN=0
      F=1
      IF(X.EQ.1)GOTO 400
      FAC(2,1)=0
  100 IF(X.EQ.1)GOTO 500
      LIM=IFIX(SQRT(FLOAT(X)))
  200 SCAN=SCAN+1
      FACT=PRIMES(SCAN)
      IF(FACT.GT.LIM)GOTO 400
      IF(MOD(X,FACT).NE.0)GOTO 200
      FAC(1,F)=FACT
  300 FAC(2,F)=FAC(2,F)+1
      X=X/FACT
      IF(MOD(X,FACT).EQ.0)GOTO 300
      F=F+1
      FAC(2,F)=0
      GOTO 100
  400 FAC(1,F)=X
      FAC(2,F)=1
      RETURN
  500 F=F-1
      RETURN
      END
```

APPENDIX B

```
C*************************************************************
C
C SUBROUTINE LIBRARY FOR WINOGRAD REAL ALGORITHMS
C           FOR LENGTHS 60,240  MOD 65521
C
C*************************************************************
C
C LENGTH 60 ALGORITHM
C
      SUBROUTINE W60(Y,FWD)
      IMPLICIT INTEGER(A-Z)
      INTEGER Y(60),X(60),S(18),TS(5)
      INTEGER RF(60),RFI(60),ARROW(12,2),COEFS(72)
      DATA RF/              0,      36,      12,      48,      24,      45,
     1     21,      57,     33,       9,      30,       6,      42,      18,
     1     54,      15,     51,      27,       3,      39,      40,      16,
     1     52,      23,      4,      25,       1,      37,      13,      49,
     1     10,      46,     22,      58,      34,      55,      31,       7,
     1     43,      19,     20,      56,      32,       8,      44,       5,
     1     41,      17,     53,      29,      50,      26,       2,      38,
     1     14,      35,     11,      47,      23,      59/
      DATA RFI/               0,      12,      24,      36,      48,      15,
     1     27,      39,     51,       3,      30,      42,      54,       6,
     1     18,      45,     57,       9,      21,      33,      20,      32,
     1     44,      56,      8,      35,      47,      59,      11,      23,
     1     50,       2,     14,      26,      38,       5,      17,      29,
     1     41,      53,     40,      52,       4,      16,      28,      55,
     1      7,      19,     31,      43,      10,      22,      34,      46,
     1     58,      25,     37,      49,       1,      13/
      DATA ARROW/           0,       0,       0,       6,      12,      12,
     1     12,      18,     24,      24,      24,      30,      36,      36,
     1     36,      42,     48,      48,      48,      54,      60,      60,
     1     60,      66/
      DATA COEFS /     1,16379,13376,64390,46385,48647,41224,
     113991,53009,26603,10376,22681,32759, 8192,45457,34457,
     128704,25311, 3685,11774,18768,25609,49957,64260,49434,
     136489,56773,45080,23174,64056,33074,56939,   32, 5797,
     128796,17202,64429, 1365, 4591, 9847, 4687,50514, 3681,
     111779,30785,35388, 4541,64807, 1638,30713,25874,17990,
     125730,55271,27239,15092,52104,12439,25949, 1071,58145,
     1 9220,13250,44432,50619,27276,50734, 2041,30577,40308,
     160673,45578/
      POINT=1
      DO 50 I=1,60
   50 X(I)=Y(RF(I)+1)
      DO 100 I=1,20
      T=MODO(X(20+I)+X(40+I))
      X(I)=MODO(X(I)+T)
      X(40+I)=MODO(X(20+I)-X(40+I))
      X(20+I)=T
  100 CONTINUE
      DO 200 TIMES=1,3
      INDEX=20*(TIMES-1)
```

APPENDIX B

```
      DO 2ØØ I=1,5
      T=MODO(X(INDEX+I)+X(INDEX+1Ø+I))
      T2=MODO(X(INDEX+5+I)+X(INDEX+15+I))
      X(INDEX+15+I)=MODO(X(INDEX+5+I)-X(INDEX+15+I))
      X(INDEX+1Ø+I)=MODO(X(INDEX+I)-X(INDEX+1Ø+I))
      X(INDEX+5+I)=MODO(T-T2)
      X(INDEX+I)=MODO(T+T2)
  2ØØ CONTINUE
C GOT TO REPEAT T4, 3 TIMES
      DO 3ØØ TIMES=1,12
      INDEX=5*(TIMES-1)
      TS(1)=MODO(X(INDEX+2)+X(INDEX+5))
      TS(2)=MODO(X(INDEX+3)+X(INDEX+4))
      S(6)=MODO(X(INDEX+2)-X(INDEX+5))
      S(5)=MODO(X(INDEX+4)-X(INDEX+3))
      S(4)=MODO(S(5)+S(6))
      S(3)=MODO(TS(1)-TS(2))
      S(2)=MODO(TS(1)+TS(2))
      S(1)=MODO(X(INDEX+1)+S(2))
      DO 25Ø I=1,6
  25Ø S(I)=MODO(S(I)*COEFS(ARROW(POINT,FWD)+I))
      POINT=POINT+1
      TS(1)=MODO(S(1)+S(2))
      TS(2)=MODO(TS(1)+S(3))
      TS(3)=MODO(S(4)-S(5))
      TS(4)=MODO(TS(1)-S(3))
      TS(5)=MODO(S(4)+S(6))
      X(INDEX+5)=MODO(TS(2)-TS(3))
      X(INDEX+4)=MODO(TS(4)-TS(5))
      X(INDEX+3)=MODO(TS(4)+TS(5))
      X(INDEX+2)=MODO(TS(2)+TS(3))
      X(INDEX+1)=S(1)
  3ØØ CONTINUE
      DO 4ØØ TIMES=1,3
      INDEX=2Ø*(TIMES-1)
      DO 4ØØ I=1,5
      T=MODO(X(INDEX+1Ø+I)+X(INDEX+15+I))
      X(INDEX+15+I)=MODO(X(INDEX+1Ø+I)-X(INDEX+15+I))
      X(INDEX+1Ø+I)=X(INDEX+5+I)
      X(INDEX+5+I)=T
  4ØØ CONTINUE
      DO 5ØØ I=1,2Ø
      T=MODO(X(I)+X(2Ø+I))
      T2=MODO(T+X(4Ø+I))
      X(4Ø+I)=MODO(T-X(4Ø+I))
      X(2Ø+I)=T2
  5ØØ CONTINUE
      DO 6ØØ I=1,6Ø
  6ØØ Y(RFI(I)+1)=X(I)
      RETURN
      END
```

APPENDIX B

```
C**********************************************************************
C
C LENGTH 240 ALGORITHM
C
      SUBROUTINE W240(Y,FWD)
      IMPLICIT INTEGER(A-Z)
      INTEGER Y(240),X(250),T(26),S(6),TS(5)
      COMMON/XARRAY/COUNT,X
C           240= 3,16, 5,
      INTEGER RF( 240),RFI( 240)
      DATA RF     /     0,    96,   192,    48,   144,   225,    81,
     1  177,    33,   129,   210,    66,   162,    13,   114,   195,
     1   51,   147,     3,    99,   180,    36,   132,   228,    84,
     1  165,    21,   117,   213,    69,   150,     6,   102,   198,
     1   54,   135,   231,    87,   183,    39,   120,   216,    72,
     1  168,    24,   105,   201,    57,   153,     9,    90,   186,
     1   42,   138,   234,    75,   171,    27,   123,   219,    60,
     1  156,    12,   108,   204,    45,   141,   237,    93,   139,
     1   30,   126,   222,    78,   174,    15,   111,   207,    63,
     1  159,   160,    16,   112,   208,    64,   145,     1,    97,
     1  193,    49,   130,   226,    82,   178,    34,   115,   211,
     1   67,   163,    19,   100,   196,    52,   148,     4,    85,
     1  181,    37,   133,   229,    70,   166,    22,   118,   214,
     1   55,   151,     7,   103,   199,    40,   136,   232,    88,
     1  184,    25,   121,   217,    73,   169,    10,   106,   202,
     1   58,   154,   235,    91,   187,    43,   139,   220,    76,
     1  172,    28,   124,   205,    61,   157,    13,   109,   190,
     1   46,   142,   238,    94,   175,    31,   127,   223,    79,
     1   80,   176,    32,   128,   224,    65,   161,    17,   113,
     1  209,    50,   146,     2,    98,   194,    35,   131,   227,
     1   83,   179,    20,   116,   212,    68,   164,     5,   101,
     1  197,    53,   149,   230,    86,   182,    38,   134,   215,
     1   71,   167,    23,   119,   200,    56,   152,     8,   104,
     1  185,    41,   137,   233,    89,   170,    26,   122,   218,
     1   74,   155,    11,   107,   203,    59,   140,   236,    92,
     1  188,    44,   125,   221,    77,   173,    29,   110,   206,
     1   62,   158,    14,    95,   191,    47,   143,   239/
      DATA RFI    /     0,    48,    96,   144,   192,    15,    63,
     1  111,   159,   207,    30,    78,   126,   174,   222,    45,
     1   93,   141,   189,   237,    60,   108,   156,   204,    12,
     1   75,   123,   171,   219,    27,    90,   138,   186,   234,
     1   42,   105,   153,   201,     9,    57,   120,   168,   216,
     1   24,    72,   135,   183,   231,    39,    87,   150,   198,
     1    6,    54,   102,   165,   213,    21,    69,   117,   180,
     1  228,    36,    84,   132,   195,     3,    51,    99,   147,
     1  210,    18,    66,   114,   162,   225,    33,    81,   129,
     1  177,    80,   128,   176,   224,    32,    95,   143,   191,
     1  239,    47,   110,   158,   206,    14,    62,   125,   173,
     1  221,    29,    77,   140,   188,   236,    44,    92,   155,
     1  203,    11,    59,   107,   170,   218,    26,    74,   122,
     1  135,   233,    41,    89,   137,   200,     8,    56,   104,
     1  152,   215,    23,    71,   119,   167,   230,    38,    86,
     1  134,   182,     5,    53,   101,   149,   197,    20,    68,
```

<u>APPENDIX B</u>

```
     1  116,  164,  212,   35,   83,  131,  179,  227,   50,
     1   98,  146,  194,    2,   65,  113,  161,  209,   17,
     1  160,  208,   16,   64,  112,  175,  223,   31,   79,
     1  127,  190,  238,   46,   94,  142,  205,   13,   61,
     1  109,  157,  220,   28,   76,  124,  172,  235,   43,
     1   91,  139,  187,   10,   58,  106,  154,  202,   25,
     1   73,  121,  169,  217,   40,   88,  136,  184,  232,
     1   55,  103,  151,  199,    7,   70,  118,  166,  214,
     1   22,   85,  133,  181,  229,   37,  100,  148,  196,
     1    4,   52,  115,  163,  211,   19,   67,  130,  178,
     1  226,   34,   82,  145,  193,    1,   49,   97/
      DO 50 I=1,250
50    X(I)=0
      IND=1
      COUNT=1
      IF(FWD.NE.1)COUNT=55
      DO 100 I=1,240
100   X(I)=Y(RF(I)+1)
      DO 200 INDEX=1,80
      TEMP=MODO(X(80+INDEX)+X(160+INDEX))
      X(160+INDEX)=MODO(X(80+INDEX)-X(160+INDEX))
      X(80+INDEX)=TEMP
      X(INDEX)=MODO(X(INDEX)+TEMP)
200   CONTINUE
      DO 300 ROUND=1,3
      DISP=80*(ROUND-1)
      DO 400 T3CNT=1,5
      SLIDE=DISP+T3CNT
      T(1)=MODO(X(SLIDE)+X(SLIDE+40))
      T(2)=MODO(X(SLIDE+20)+X(SLIDE+60))
      T(3)=MODO(X(SLIDE+10)+X(SLIDE+50))
      T(4)=MODO(X(SLIDE+10)-X(SLIDE+50))
      T(5)=MODO(X(SLIDE+30)+X(SLIDE+70))
      T(6)=MODO(X(SLIDE+30)-X(SLIDE+70))
      T(7)=MODO(X(SLIDE+5)+X(SLIDE+45))
      T(8)=MODO(X(SLIDE+5)-X(SLIDE+45))
      T(9)=MODO(X(SLIDE+15)+X(SLIDE+55))
      T(10)=MODO(X(SLIDE+15)-X(SLIDE+55))
      T(11)=MODO(X(SLIDE+25)+X(SLIDE+65))
      T(12)=MODO(X(SLIDE+25)-X(SLIDE+65))
      T(13)=MODO(X(SLIDE+35)+X(SLIDE+75))
      T(14)=MODO(X(SLIDE+35)-X(SLIDE+75))
      T(15)=MODO(T(1)+T(2))
      T(16)=MODO(T(3)+T(5))
      T(17)=MODO(T(15)+T(16))
      T(18)=MODO(T(7)+T(11))
      T(19)=MODO(T(7)-T(11))
      T(20)=MODO(T(9)+T(13))
      T(21)=MODO(T(9)-T(13))
```

APPENDIX B

```
    T(22)=MODO(T(18)+T(20))
    T(23)=MODO(T(8)+T(14))
    T(24)=MODO(T(3)-T(14))
    T(25)=MODO(T(10)+T(12))
    T(26)=MODO(T(12)-T(10))
    X(SLIDE+60)=MODO(X(SLIDE+20)-X(SLIDE+60))
    X(SLIDE+20)=MODO(X(SLIDE)-X(SLIDE+40))
    X(SLIDE)=MODO(T(17)+T(22))
    X(SLIDE+5)=MODO(T(17)-T(22))
    X(SLIDE+10)=MODO(T(15)-T(16))
    X(SLIDE+15)=MODO(T(1)-T(2))
    X(SLIDE+25)=MODO(T(19)-T(21))
    X(SLIDE+30)=MODO(T(4)-T(6))
    X(SLIDE+35)=MODO(T(24)+T(26))
    X(SLIDE+40)=MODO(T(24))
    X(SLIDE+45)=MODO(T(26))
    X(SLIDE+50)=MODO(T(18)-T(20))
    X(SLIDE+55)=MODO(T(3)-T(5))
    X(SLIDE+65)=MODO(T(19)+T(21))
    X(SLIDE+70)=MODO(T(4)+T(6))
    X(SLIDE+75)=MODO(T(23)+T(25))
    X(T3CNT+240)=MODO(T(23))
    X(T3CNT+245)=MODO(T(25))
400 CONTINUE
    DO 500 TIMES=1,16
500 CALL W240U5(DISP+5*(TIMES-1))
    CALL W240U5(240)
    CALL W240U5(245)
    DO 600 T3CNT=1,5
    SLIDE=DISP+T3CNT
    T(1)=MODO(X(SLIDE+15)+X(SLIDE+25))
    T(2)=MODO(X(SLIDE+15)-X(SLIDE+25))
    T(3)=MODO(X(SLIDE+55)+X(SLIDE+65))
    T(4)=MODO(X(SLIDE+65)-X(SLIDE+55))
    T(5)=MODO(X(SLIDE+20)+X(SLIDE+30))
    T(6)=MODO(X(SLIDE+20)-X(SLIDE+30))
    T(7)=MODO(X(SLIDE+40)-X(SLIDE+35))
    T(8)=MODO(X(SLIDE+45)-X(SLIDE+35))
    T(9)=MODO(T(5)+T(7))
    T(10)=MODO(T(5)-T(7))
    T(11)=MODO(T(6)+T(8))
    T(12)=MODO(T(6)-T(8))
    T(13)=MODO(X(SLIDE+60)+X(SLIDE+70))
    T(14)=MODO(X(SLIDE+60)-X(SLIDE+70))
    T(15)=MODO(X(SLIDE+75)+X(T3CNT+240))
    T(16)=MODO(X(SLIDE+75)-X(T3CNT+245))
    T(17)=MODO(T(13)+T(15))
    T(18)=MODO(T(13)-T(15))
    T(19)=MODO(T(14)+T(16))
```

APPENDIX B

```
      T(20)=MODO(T(14)-T(16))
      X(SLIDE)=X(SLIDE)
      X(SLIDE+60)=MODO(X(SLIDE+10)-X(SLIDE+50))
      X(SLIDE+40)=MODO(X(SLIDE+5))
      X(SLIDE+20)=MODO(X(SLIDE+10)+X(SLIDE+50))
      X(SLIDE+5)=MODO(T(9)+T(17))
      X(SLIDE+10)=MODO(T(1)+T(3))
      X(SLIDE+15)=MODO(T(12)-T(20))
      X(SLIDE+25)=MODO(T(11)+T(19))
      X(SLIDE+30)=MODO(T(2)+T(4))
      X(SLIDE+35)=MODO(T(10)-T(18))
      X(SLIDE+45)=MODO(T(10)+T(18))
      X(SLIDE+50)=MODO(T(2)-T(4))
      X(SLIDE+55)=MODO(T(11)-T(19))
      X(SLIDE+65)=MODO(T(12)+T(20))
      X(SLIDE+70)=MODO(T(1)-T(3))
      X(SLIDE+75)=MODO(T(9)-T(17))
  600 CONTINUE
  300 CONTINUE
      DO 900 INDEX=1,80
      TEMP=MODO(X(INDEX)+X(80+INDEX))
      TEMP2=MODO(TEMP-X(160+INDEX))
      X(80+INDEX)=MODO(TEMP+X(160+INDEX))
      X(160+INDEX)=MODO(TEMP2)
  900 CONTINUE
      DO 950 I=1,240
  950 Y(RFI(I)+1)=X(I)
      RETURN
      END
      SUBROUTINE W240U5(INDEX)
      IMPLICIT INTEGER(A-Z)
      INTEGER X(250),S(6),TS(5)
      COMMON/XARRAY/COUNT,X
      INTEGER POINT( 108)
      DATA POINT /     0,     0,     0,     0,     0,     6,     6,
     1   12,    18,    24,    30,    30,    30,    36,    36,    42,
     1   48,    54,    60,    60,    60,    60,    60,    66,    66,
     1   72,    78,    84,    90,    90,    90,    96,    96,   102,
     1  108,   114,   120,   120,   120,   120,   120,   126,   126,
     1  132,   138,   144,   150,   150,   150,   156,   156,   162,
     1  168,   174,   180,   180,   180,   180,   180,   186,   186,
     1  192,   198,   204,   210,   210,   210,   216,   216,   222,
     1  228,   234,   240,   240,   240,   240,   240,   246,   246,
     1  252,   258,   264,   270,   270,   270,   276,   276,   282,
     1  288,   294,   300,   300,   300,   300,   300,   306,   306,
     1  312,   318,   324,   330,   330,   330,   336,   336,   342,
     1  348,   354,     2,     3,     5,     7,    11,    13,    17,
     1   19,    23,    29,    31,    37,    41,    43,    47,    53,
     1   59,    61,    67,    71,    73,    79,    83,    89,    97,
```

APPENDIX B

```
1  101,  103,  107,  109,  113,  127,  131,  137,  139,
1  149,  151,  157,  163,  167,  173,  179,  181,  191,
1  193,  197,  199,  211,  223,  227,  229,  233,  239,
1  241,  251,  257,  263,  269,  271,  277,  281,  283,
1  293,  307,  311,  313,  317,  331,  337,  347,  349,
1  353,  359,  367,  373,  379,  383,  389,  397,  401,
1  409,  419,  421,  431,  433,  439,  443,  449,  457,
1  461,  463,  467,  479,  487,  491,  499,  503,  509,
1  521,  523,  541,  547,  557,  563,  569,  571,  577,
1  587,  593,  599,  601,  607,  613,  617,  619,  631,
1  641,  643,  647,  653,  659,  661,  673,  677,  683,
1  691,  701,  709,  719,  727,  733,  739,  743/
 DATA WCOEF /     1,16379,13376,64390,46385,48647,  181,
115154,62300,57373, 8997,25293,35219, 5117,58875, 4079,
163743,55585,43121,28000, 5133,43294, 3018,52472,27317,
147755,47096,30335,53947,58698,41224,13991,53009,26608,
110376,22681,57671,42573,28563,33015,43468,42959,47157,
122955, 1365,65043,24331,25327, 6581, 8154,32753,26283,
162867,10301,35374,54064,35483,25337,46108,60955,32759,
1 8192,45457,34457,23704,25311,32489,41290,37592,12222,
119265,60342,45453,25085, 9969,26642, 2667,14904,33600,
123521,25061,  580,53494,52334,57306,26649,60398,52704,
117361,42995, 3685,11774,18763,25609,49957,64260,11775,
134422,55437,48759,  319,33843,27546,63849,30713,33470,
161710,60291,22889,53290,49152,58357, 3981,17309,12460,
149946,45057,60276,61880, 6849,49434,36489,56773,45080,
123174,64056,36698,52409,54637,34876, 1150,62440,57555,
142718,49451,33169,35530,34713,43821,20875,47410,17652,
125283,55500,  768,64561,51492,48686,45777,13926,33074,
156939,   32, 5797,28796,17202,23983,19162, 5792,  921,
135917,34075,53000,64792,56301, 8715,57080,39650,13389,
165165,22371,57913,40727,55943,53863,63707, 3931, 9822,
123845, 4201,65248,49482,17528,18842,17552,45389,16108,
145386,27560, 3310,31904,25284,16800,44521,45291,65231,
138774,39354,21747,21957,40153,25482,26721,41278,11853,
1 1564,50429,39459,50827,37430,50061,19325,56837, 8847,
150276,32582,19143,25212,  700,28803,58058,  452,31745,
142220,45040,63608,27129,30955,27546,63849,30713,32051,
1 3811, 5230,25515,17247,55272,28225,58069, 1619,33170,
156819,39229,37258,39193,30198,41359,62963,24181,60556,
117665,27595,40321,31500,30345,  435, 7360, 6490,  140,
165346,38052,27298,58200, 3604,14981,63175,22638,39093,
122041, 9376,23190, 3773,13026,19490,55628,16648, 4046,
127703,64471,55077,43955,64843,50664, 2191,63482,35630,
157588,51849,24202, 2508,52212,50205,27044,57676,60009,
1 6890,48134,55944,11178,30332,63677, 2305,36073,11108,
129035, 6819,59362,24079,42634,44918,13655,54861,52996,
164797, 2797,52282,62939,23896,27370,64069,35293,29558,
142967,49372,13101,    4,35822, 9435,17390,63941,12696,
```

APPENDIX B

```
149651,56785,1ØØ77,643Ø9,44155, 4741,1Ø454,568Ø9,5487Ø,
142712,64Ø14,11141, 2454,27262,2Ø439,54363,13485,13979,
131667,51691,19688,45422, 6Ø46,36261,35575,4Ø694,6Ø556,
1231Ø6,33Ø16/
 TS(1)=MODO(X(INDEX+2)+X(INDEX+5))
 TS(2)=MODO(X(INDEX+3)+X(INDEX+4))
 S(6)=MODO(X(INDEX+2)-X(INDEX+5))
 S(5)=MODO(X(INDEX+4)-X(INDEX+3))
 S(4)=MODO(S(5)+S(6))
 S(3)=MODO(TS(1)-TS(2))
 S(2)=MODO(TS(1)+TS(2))
 S(1)=MODO(X(INDEX+1)+S(2))
 DO 25Ø I=1,6
25Ø S(I)=MODO(S(I)*WCOEF(POINT(COUNT)+I))
 COUNT=COUNT+1
 TS(1)=MODO(S(1)+S(2))
 TS(2)=MODO(TS(1)+S(3))
 TS(3)=MODO(S(4)-S(5))
 TS(4)=MODO(TS(1)-S(3))
 TS(5)=MODO(S(4)+S(6))
 X(INDEX+5)=MODO(TS(2)-TS(3))
 X(INDEX+4)=MODO(TS(4)-TS(5))
 X(INDEX+3)=MODO(TS(4)+TS(5))
 X(INDEX+2)=MODO(TS(2)+TS(3))
 X(INDEX+1)=S(1)
 RETURN
 END
```

APPENDIX C

```
( LENGTH 6Ø TRANSFORM ALGORITM FOR INTEL 8Ø8Ø
  USING PROM LOOK UP MULTIPLICATIONS
  WRITTEN IN FORTH                             )
6Ø ARRAY RF 6Ø ARRAY RFI 24 ARRAY INDEX 12 ARRAY S
12Ø ARRAY FILTER 12Ø ARRAY WKSP 144 ARRAY COEFS
 12Ø ARRAY X 12Ø ARRAY Y  12Ø  ARRAY Z
 Ø INTEGER .X Ø INTEGER .Y Ø INTEGER .INDEX   Ø INTEGER .H
  RF FILLB
   Ø  72  24  96  48  9Ø  42 114  66  18  6Ø  12  84  36 1Ø8
  3Ø 1Ø2  54   6  78  8Ø  32 1Ø4  56   8  5Ø   2  74  26  98
  2Ø  92  44 116  68 11Ø  62  14  86  38  4Ø 112  64  16  88
  1Ø  82  34 1Ø6  58 1ØØ  52   4  76  28  7Ø  22  94  46 118
  RFI FILLB
   Ø  24  48  72  96  3Ø  54  78 1Ø2   6  6Ø  84 1Ø8  12  36
  9Ø 114  18  42  66  4Ø  64  88 112  16  7Ø  94 113  22  46
 1ØØ   4  28  52  76  1Ø  34  58  82 1Ø6  8Ø 1Ø4   8  32  56
 11Ø  14  38  62  86  2Ø  44  68  92 116  5Ø  74  98   2  26
  INDEX FILLB
  Ø  Ø  Ø  6 12 12 12 18 24 24 24 3Ø 36 36 36 42 48 48 48 54
 6Ø 6Ø 6Ø 66
COEFS FILL      1 16379 13376 6439Ø 46385 48647 41224 13991
53ØØ9 266Ø8 1Ø376 22681 32759  8192 45457 34457 287Ø4 25311
 3685 11774 18768 256Ø9 49957 6426Ø 49434 36489 56773 45Ø8Ø
23174 64Ø56 33Ø74 56939    32  5797 28796 172Ø2 64429  1365
 4591  9847  4687 59514  3681 11779 3Ø785 35388  4541 648Ø7
 1638 3Ø713 25874 1799Ø 2573Ø 55271 27239 15Ø92 521Ø4 12439
25949  1Ø71 58145  922Ø 1325Ø 44432 5Ø619 27276 5Ø734  2Ø41
3Ø577 4Ø3Ø8 6Ø673 45578
: FILL 6Ø Ø DO DUP I DUP + + I SWAP ! LOOP DROP ;
: INNER 6 Ø DO DUP I DUP + + @ , LOOP DROP ;
: DUMP 6Ø Ø DO DUP I DUP + + INNER CRLF 6 +LOOP DROP ;
CODE RFSWAP 118 B LXI 59 A MVI BEGIN RF H LXI E A MOV Ø D MVI
D DAD E M MOV .X LHLD D DAD E M MOV H INX D M MOV WKSP H LXI
 B DAD M E MOV H INX M D MOV C DCR C DCR A DCR FM END RET
CODE RFISWAP 118 B LXI 59 A MVI BEGIN WKSP H LXI B DAD E M MOV
H INX D M MOV D PUSH E A MOV Ø D MVI RFI H LXI D DAD E M MOV
.X LHLD
 D DAD D POP M E MOV H INX M D MOV C DCR C DCR A DCR FM END RET
CODE +M BEGIN D DAD RNC 15 D LXI UC END
CODE -M 241 A MVI E SUB E A MOV 255 A MVI D SBB D A MOV
$ +M JNC A E MOV 15 SUI E A MOV A D MOV Ø SBI D A MOV
$ +M JMP
( +M INPUT IN HL+DE OUTPUT IN HL, -M SAME OUTPUT=HL-DE )
: :GETI H LXI B DAD E M MOV H INX D M MOV ;
: :STOREI H LXI B DAD M E MOV H INX M D MOV ;
: -MACRO :GETI D PUSH :GETI H POP [ $ -M STK ] CALL XCHG
 :STOREI ; ( 3 2 1 -MACRO == X(3+I)=X(1+I)-X(2+I)   )
: ;STORE H LXI M E MOV H INX M D MOV ;
```

APPENDIX C

```
: ;GET   H LXI E M MOV H INX D M MOV ;
CODE T3 38 B LXI BEGIN WKSP 4Ø + :GETI D PUSH
WKSP 8Ø + :GETI H POP $ +M CALL  H PUSH H PUSH WKSP
:GETI H POP $ +M CALL XCHG WKSP :STOREI WKSP 4Ø +
:GETI D PUSH WKSP 8Ø + :GETI H POP $ -M CALL XCHG
WKSP 8Ø + :STOREI D POP WKSP 4Ø +
:STOREI C DCR C DCR FM END RET
CODE IT4 BEGIN PSW PUSH WKSP :GETI D PUSH WKSP 2Ø + :GETI H POP
 $ +M CALL H PUSH WKSP 1Ø + :GETI D PUSH WKSP 3Ø + :GETI H POP
$ +M CALL H PUSH WKSP 3Ø + WKSP 3Ø + WKSP 1Ø + -MACRO
WKSP 2Ø + WKSP 2Ø + WKSP -MACRO
D POP H POP H PUSH D PUSH $ +M CALL XCHG WKSP :STOREI
D POP H POP $ -M CALL XCHG WKSP 1Ø + :STOREI
C DCR C DCR PSW POP A DCR FZ END RET
CODE T4  5  A MVI  8 B LXI $ IT4 CALL
         5  A MVI 48 B LXI $ IT4 CALL
         5  A MVI 88 B LXI $ IT4 JMP
CODE *MG B D MOV C E MOV Ø D LXI
16 A MVI BEGIN $ HLDE*2 CALL FC IF XCHG B DAD XCHG FC IF
H INX THEN THEN A DCR FZ END D PUSH B H MOV
14 H MVI D M MOV H INR E M MOV D PUSH L B MOV
D M MOV 13 H MVI E M MOV H POP $ +M CALL D POP
$ +M CALL RET : *MGB PPH PPD *MG PSH ;
( MULTS REQUIRES INDEX FOR FWD OR INDEX+12 FOR REVERSE TRAN
STORED IN .INDEX BEFORE STARTING )
CODE MULTS S H LXI XCHG .INDEX LHLD A M MOV RLC H INX
.INDEX SHLD COEFS H LXI C A MOV Ø B MVI B DAD
6 A MVI BEGIN PSW PUSH
C M MOV H INX B M MOV H INX H PUSH XCHG E M MOV H INX D M MOV
H PUSH L C MOV H B MOV $ *MG CALL XCHG H POP M D MOV H DCX
M E MOV H INX H INX XCHG H POP PSW POP A DCR FZ END RET
CODE U5 Ø B LXI 12 A MVI BEGIN PSW PUSH WKSP 2 + :GETI D PUSH
WKSP 8  + :GETI H POP $ +M CALL H PUSH WKSP 4 + :GETI D PUSH
WKSP 6 + :GETI H POP $ +M CALL D POP D PUSH H PUSH $ +M CALL
XCHG S 2 + ;STORE D PUSH WKSP :GETI H POP $ +M CALL XCHG S
;STORE D POP H POP $ -M CALL XCHG S 4 + ;STORE WKSP 2 + :GETI
D PUSH WKSP 8 + :GETI H POP $ -M CALL XCHG S 1Ø + ;STORE D PUSH
WKSP 6 + :GETI D PUSH WKSP 4 + :GETI H POP $ -M CALL XCHG S 8 +
;STORE H POP $ +M CALL XCHG S 6 + ;STORE B PUSH $ MULTS CALL
B POP S ;GET WKSP :STOREI D PUSH S 2 + ;GET H POP $ +M CALL H
PUSH S 4 + ;GET H POP H PUSH D PUSH $ +M CALL H PUSH S 6 + ;GET
D PUSH S 8 + ;GET H POP $ -M CALL D POP D PUSH H PUSH $ +M CALL
XCHG WKSP 2 + :STOREI D POP  H POP $ -M CALL XCHG  WKSP 8 +
:STOREI D POP H POP $ -M CALL H PUSH S 6 + ;GET D PUSH S 1Ø +
;GET H POP $ +M CALL D POP D PUSH H PUSH $ +M CALL XCHG WKSP 4 +
 :STOREI D POP  H POP $ -M CALL XCHG WKSP 6 + :STOREI
A C MOV 1Ø ADI C A MOV PSW POP A DCR FZ END RET
CODE IS4 BEGIN PSW PUSH WKSP 2Ø + :GETI D PUSH
WKSP 3Ø + :GETI H POP $ +M CALL H PUSH
```

APPENDIX C

```
WKSP 3Ø + WKSP 3Ø + WKSP 2Ø + -MACRO
WKSP 1Ø + :GETI WKSP 2Ø + :STOREI D POP
WKSP 1Ø +  :STOREI
C DCR C DCR PSW POP A DCR FZ END RET
CODE S4 5 A MVI  8 B LXI $ IS4 CALL
        5 A MVI 48 B LXI $ IS4 CALL
        5 A MVI 88 B LXI $ IS4 JMP
CODE S3 38 B LXI BEGIN WKSP :GETI D PUSH WKSP 4Ø + :GETI
H POP $ +M CALL H PUSH WKSP 8Ø + :GETI H POP H PUSH D PUSH
$ -M CALL XCHG WKSP 8Ø + :STOREI D POP H POP $ +M CALL XCHG
      WKSP 4Ø + :STOREI C DCR C DCR FM END RET
: TRAN .X ! IF 12 ELSE Ø THEN INDEX + .INDEX !
 RFSWAP T3 T4 U5 S4 S3 RFISWAP ;
CODE XHMULT 113 B LXI BEGIN .H LHLD B DAD E M MOV H INX D M MOV
D PUSH .X LHLD B DAD E M MOV H INX D M MOV XTHL B PUSH $ *MG
CALL B POP XCHG H POP M D MOV H DCX M E MOV C DCR C DCR FM END
RET
: CONVOLUTION .X ! .H ! INDEX .INDEX ! RFSWAP T3 T4 U5 S4 S3
RFISWAP XHMULT RFSWAP T3 T4 U5 S4 S3 RFISWAP ;
```

APPENDIX D

```
C************************************************************
C
C COMPLEX INTEGER LIBRARY
C
C COMPLEX INTEGER STORED AS TWO I*4 IN ADJACENT FULL WORDS
C APPEARS TO FORTRAN AS REAL*8.
C
      REAL FUNCTION CM*8(R1,R2)
C COMPLEX INTEGER MODULAR MULTIPLY
C
      REAL*8 R1,R2,R3,R4,R5
      INTEGER IR3(2),IR4(2),IR5(2)
      EQUIVALENCE (R3,IR3(1)),(R4,IR4(1)),(R5,IR5(1))
      R3=R1
      R4=R2
      IR5(1)=MODO(IR3(1)*IR4(1)-IR3(2)*IR4(2))
      IR5(2)=MODO(IR3(1)*IR4(2)+IR3(2)*IR4(1))
      CM=R5
      RETURN
      END
C************************************************************
      REAL FUNCTION CA*8(R1,R2)
C
C COMPLEX INTEGER MODULAR ADD
C
      REAL*8 R1,R2,R3,R4,R5
      INTEGER IR3(2),IR4(2),IR5(2)
      EQUIVALENCE (R3,IR3(1)),(R4,IR4(1)),(R5,IR5(1))
      R3=R1
      R4=R2
      IR5(1)=MODO(IR3(1)+IR4(1))
      IR5(2)=MODO(IR3(2)+IR4(2))
      CA=R5
      RETURN
      END
C************************************************************
      REAL FUNCTION CS*8(R1,R2)
C
C COMPLEX INTEGER MODULAR SUBTRACT
C
      REAL*8 R1,R2,R3,R4,R5
      INTEGER IR3(2),IR4(2),IR5(2)
      EQUIVALENCE (R3,IR3(1)),(R4,IR4(1)),(R5,IR5(1))
      R3=R1
      R4=R2
      IR5(1)=MODO(IR3(1)-IR4(1))
      IR5(2)=MODO(IR3(2)-IR4(2))
      CS=R5
      RETURN
      END
```

APPENDIX D

```
C*************************************************************
      REAL FUNCTION CSET*8(I,J)
C
C COMPLEX INTEGER C=CMPLX(I,J)
C
      REAL*8 A
      INTEGER AI(2)
      EQUIVALENCE(A,AI(1))
      AI(1)=I
      AI(2)=J
      CSET=A
      RETURN
      END
C*************************************************************
C
C CEXPM=ALPHA**X
C
      REAL FUNCTION CEXPM*8(ALPHA,X)
      IMPLICIT REAL*8 (A-H,O-Z)
      Y=X
      CEXPM=CSET(1,0)
      PROD=ALPHA
  100 Y=Y-DMOD(Y,1.0D0)
      IF(DMOD(Y,2.0D0).NE.0.0D0)CEXPM=CM(CEXPM,PROD)
      Y=Y/2.0D0
      IF(Y.LT.1.0D0)RETURN
       PROD=CM(PROD,PROD)
      GOTO 100
      END
C*************************************************************
C
C CINV RETURNS A TO THE POWER M**2-2
C
      REAL FUNCTION CINV*8(A)
      IMPLICIT REAL*8 (A-D)
      COMMON/MD/MD,MD2
      AM=MD
      AM=AM**2 -2.0D0
      CINV=CEXPM(A,AM)
      IF(CM(CINV,A).NE.CSET(1,0))PAUSE 'CINV ERROR'
      RETURN
      END
C*************************************************************
C
C CPRIM RETURNS A 'PRIMITIVE' ROOT OF ORDER M**2-1
C
      REAL FUNCTION CPRIM*8(WASTE)
      IMPLICIT REAL*8 (A-D),INTEGER (E-Z)
      INTEGER AI(2),FAC(2,255)
      EQUIVALENCE (A,AI(1))
      COMMON/MD/MD,MD2
      AM=MD
      AM=AM*AM -1.0D0
```

APPENDIX D

```
      CALL BIGFAC(AM,FAC,F)
      DO 5Ø I=1,F
      AB=AM/FAC(1,I)
   5Ø FAC(1,I)=AB
      CONE=CSET(1,Ø)
      A=CSET(1,Ø)
  1ØØ AI(2)=MODO(AI(2)+1)
      IF(AI(2).EQ.Ø)PAUSE 'CPRIM NO ROOT'
      TEST=1
  2ØØ CTEST=CEXPM(A,DFLOAT(FAC(1,TEST)))
      IF(CTEST.EQ.CONE)GOTO 1ØØ
      TEST=TEST+1
      IF(TEST.LE.F)GOTO 2ØØ
      CPRIM=A
      RETURN
      END
C*************************************************************
C
C COMPLEX COS AND SIN FUNCTION
C
      REAL FUNCTION CMCOS*8(U,P)
      IMPLICIT REAL*8 (A-D),INTEGER (E-Z)
      COMMON/MCOSSN/LASTU,VAL
      COMMON/JJ/J
      REAL*8 VAL(4,2),POWER(4),U,LASTU,INV2,INV2J,J
      ROW=1
      GOTO 1ØØ
      ENTRY CMSIN(U,P)
      ROW=2
  1ØØ IF(U.EQ.LASTU)GOTO 3ØØ
      POWER(1)=U
      DO 15Ø I=2,4
  15Ø POWER(I)=CM(POWER(I-1),U)
      INV2=CINV(CSET(2,Ø))
      INV2J=CINV(CM(CSET(2,Ø),J))
      DO 2ØØ I=1,4
      VAL(I,1)=CM(INV2,CA(POWER(I),CINV(POWER(I))))
      VAL(I,2)=CM(INV2J,CS(POWER(I),CINV(POWER(I))))
  2ØØ CONTINUE
      LASTU=U
  3ØØ CMCOS=Ø
      IF(P.LE.4)CMCOS=VAL(P,ROW)
      CMSIN=CMCOS
      RETURN
      END
```

APPENDIX E

```
C*************************************************************
C
C SUBROUTINE LIBRARY FOR WINOGRAD COMPLEX ALGORITHMS
C          FOR LENGTHS 6Ø,24Ø,  MOD 65519
C
C*************************************************************
C
C LENGTH 6Ø ALGORITHM
C
      SUBROUTINE C6Ø(Y,FWD)
      IMPLICIT REAL*8 (A-D),INTEGER(E-Z)
      REAL*8 Y(6Ø),X(6Ø),S(18),TS(5),COEFS(72),T,T2
      INTEGER RF(6Ø),RFI(6Ø),ARROW(12,2),ICOEF(144)
      EQUIVALENCE(COEFS(1),ICOEF(1))
      DATA RF/              Ø,    36,     12,     48,     24,     45,
     1    21,     57,     33,      9,     3Ø,      6,     42,     18,
     1    54,     15,     51,     27,      3,     39,     4Ø,     16,
     1    52,     28,      4,     25,      1,     37,     13,     49,
     1    1Ø,     46,     22,     58,     34,     55,     31,      7,
     1    43,     19,     2Ø,     56,     32,      8,     44,      5,
     1    41,     17,     53,     29,     5Ø,     26,      2,     38,
     1    14,     35,     11,     47,     23,     59/
      DATA RFI/              Ø,     12,     24,     36,     48,     15,
     1    27,     39,     51,      3,     3Ø,     42,     54,      6,
     1    18,     45,     57,      9,     21,     33,     2Ø,     32,
     1    44,     56,      8,     35,     47,     59,     11,     23,
     1    5Ø,      2,     14,     26,     38,      5,     17,     29,
     1    41,     53,     4Ø,     52,      4,     16,     28,     55,
     1     7,     19,     31,     43,     1Ø,     22,     34,     46,
     1    58,     25,     37,     49,      1,     13/
      DATA ARROW/           Ø,      Ø,      Ø,      6,     12,     12,
     1    12,     18,     24,     24,     24,     3Ø,     36,     36,
     1    36,     42,     48,     48,     48,     54,     6Ø,     6Ø,
     1    6Ø,     66/
      DATA ICOEF /     1,     Ø,49138,     Ø,64206,     Ø,     Ø,
     157700,     Ø,54Ø97,     Ø, 4216,     Ø,     1,     Ø,49138,
     1     Ø,642Ø6, 7819,     Ø,11422,     Ø,613Ø3,     Ø,32758,
     1     Ø,57331,     Ø,34729,     Ø,     Ø,44488,     Ø,17133,
     1     Ø,59195,     Ø,32758,     Ø,57331,     Ø,34729,21Ø31,
     1     Ø,48386,     Ø, 6324,     Ø,     Ø,376Ø6,     Ø,51271,
     1     Ø,24648,57561,     Ø,58687,     Ø, 9Ø84,     Ø,27913,
     1     Ø,14248,     Ø,4Ø871,     Ø,     Ø,57561,     Ø,58687,
     1     Ø, 9Ø84, 1Ø92,     Ø,64154,     Ø, 7622,     Ø,     Ø,
     12Ø878,     Ø,24214,     Ø,47977,     Ø,64427,     Ø, 1365,
     1     Ø,57897,2Ø878,     Ø,24214,     Ø,47977,     Ø,63881,
     1     Ø,348Ø7,     Ø,54Ø36,     Ø,     Ø,342Ø2,     Ø,29198,
     1     Ø,26313,     Ø, 1638,     Ø,3Ø712,     Ø,11433,342Ø2,
     1     Ø,29198,     Ø,26313,     Ø,     Ø,14661,     Ø,3Ø313,
     1     Ø,12693,23891,     Ø, 8622,     Ø,26359,     Ø,14661,
     1     Ø,3Ø313,     Ø,12693,     Ø,     Ø,41623,     Ø,56897,
     1     Ø,3916Ø/
      POINT=1
      DO 5Ø I=1,6Ø
```

APPENDIX E

```
   50 X(I)=Y(RF(I)+1)
      DO 100 I=1,20
      T=CA(X(20+I),X(40+I))
      X(I)=CA(X(I),T)
      X(40+I)=CS(X(20+I),X(40+I))
      X(20+I)=T
  100 CONTINUE
      DO 200 TIMES=1,3
      INDEX=20*(TIMES-1)
      DO 200 I=1,5
      T=CA(X(INDEX+I),X(INDEX+10+I))
      T2=CA(X(INDEX+5+I),X(INDEX+15+I))
      X(INDEX+15+I)=CS(X(INDEX+5+I),X(INDEX+15+I))
      X(INDEX+10+I)=CS(X(INDEX+I),X(INDEX+10+I))
      X(INDEX+5+I)=CS(T,T2)
      X(INDEX+I)=CA(T,T2)
C GOT TO REPEAT T4, 3 TIMES
  200 CONTINUE
      DO 300 TIMES=1,12
      INDEX=5*(TIMES-1)
      TS(1)=CA(X(INDEX+2),X(INDEX+5))
      TS(2)=CA(X(INDEX+3),X(INDEX+4))
      S(6)=CS(X(INDEX+2),X(INDEX+5))
      S(5)=CS(X(INDEX+4),X(INDEX+3))
      S(4)=CA(S(5),S(6))
      S(3)=CS(TS(1),TS(2))
      S(2)=CA(TS(1),TS(2))
      S(1)=CA(X(INDEX+1),S(2))
      DO 250 I=1,6
  250 S(I)=CM(S(I),COEFS(ARROW(POINT,FWD)+I))
      POINT=POINT+1
      TS(1)=CA(S(1),S(2))
      TS(2)=CA(TS(1),S(3))
      TS(3)=CS(S(4),S(5))
      TS(4)=CS(TS(1),S(3))
      TS(5)=CA(S(4),S(6))
      X(INDEX+5)=CS(TS(2),TS(3))
      X(INDEX+4)=CS(TS(4),TS(5))
      X(INDEX+3)=CA(TS(4),TS(5))
      X(INDEX+2)=CA(TS(2),TS(3))
      X(INDEX+1)=S(1)
  300 CONTINUE
      DO 400 TIMES=1,3
      INDEX=20*(TIMES-1)
      DO 400 I=1,5
      T=CA(X(INDEX+10+I),X(INDEX+15+I))
      X(INDEX+15+I)=CS(X(INDEX+10+I),X(INDEX+15+I))
      X(INDEX+10+I)=X(INDEX+5+I)
      X(INDEX+5+I)=T
  400 CONTINUE
      DO 500 I=1,20
      T=CA(X(I),X(20+I))
      T2=CA(T,X(40+I))
      X(40+I)=CS(T,X(40+I))
```

APPENDIX E

```
      X(20+I)=T2
  500 CONTINUE
      DO 600 I=1,60
  600 Y(RFI(I)+1)=X(I)
      RETURN
      END
C*************************************************************
C
C LENGTH 240 ALGORITHM
C
      SUBROUTINE C240(Y,FWD)
      IMPLICIT REAL*8 (A-D),INTEGER(E-Z)
      REAL*8 Y(240),X(250),T(26),S(6),TS(5),TEMP,TEMP2
      INTEGER COUNT,DISP
C           240= 3,16, 5,
      COMMON/CARRAY/X,COUNT
      INTEGER RF( 240),RFI( 240)
      DATA RF     /     0,    96,   192,    48,   144,   225,    81,
     1  177,    33,   129,   210,    66,   162,    18,   114,   195,
     1   51,   147,     3,    99,   180,    36,   132,   228,    84,
     1  165,    21,   117,   213,    69,   150,     6,   102,   198,
     1   54,   135,   231,    87,   183,    39,   120,   216,    72,
     1  168,    24,   105,   201,    57,   153,     9,    90,   186,
     1   42,   138,   234,    75,   171,    27,   123,   219,    60,
     1  156,    12,   108,   204,    45,   141,   237,    93,   189,
     1   30,   126,   222,    78,   174,    15,   111,   207,    63,
     1  159,   160,    16,   112,   208,    64,   145,     1,    97,
     1  193,    49,   130,   226,    82,   178,    34,   115,   211,
     1   67,   163,    19,   100,   196,    52,   148,     4,    85,
     1  181,    37,   133,   229,    70,   166,    22,   118,   214,
     1   55,   151,     7,   103,   199,    40,   136,   232,    88,
     1  184,    25,   121,   217,    73,   169,    10,   106,   202,
     1   58,   154,   235,    91,   187,    43,   139,   220,    76,
     1  172,    28,   124,   205,    61,   157,    13,   109,   190,
     1   46,   142,   238,    94,   175,    31,   127,   223,    79,
     1   80,   176,    32,   128,   224,    65,   161,    17,   113,
     1  209,    50,   146,     2,    98,   194,    35,   131,   227,
     1   83,   179,    20,   116,   212,    68,   164,     5,   101,
     1  197,    53,   149,   230,    86,   182,    38,   134,   215,
     1   71,   167,    23,   119,   200,    56,   152,     8,   104,
     1  185,    41,   137,   233,    89,   170,    26,   122,   218,
     1   74,   155,    11,   107,   203,    59,   140,   236,    92,
     1  188,    44,   125,   221,    77,   173,    29,   110,   206,
     1   62,   158,    14,    95,   191,    47,   143,   239/
      DATA RFI    /     0,    48,    96,   144,   192,    15,    63,
     1  111,   159,   207,    30,    78,   126,   174,   222,    45,
     1   93,   141,   189,   237,    60,   108,   156,   204,    12,
     1   75,   123,   171,   219,    27,    90,   138,   186,   234,
     1   42,   105,   153,   201,     9,    57,   120,   168,   216,
     1   24,    72,   135,   183,   231,    39,    87,   150,   198,
     1    6,    54,   102,   165,   213,    21,    69,   117,   180,
     1  228,    36,    84,   132,   195,     3,    51,    99,   147,
     1  210,    18,    66,   114,   162,   225,    33,    81,   129,
     1  177,    80,   128,   176,   224,    32,    95,   143,   191,
```

APPENDIX E

```
     1  239,   47,  110,  158,  206,   14,   62,  125,  173,
     1  221,   29,   77,  140,  188,  236,   44,   92,  155,
     1  203,   11,   59,  107,  170,  218,   26,   74,  122,
     1  185,  233,   41,   89,  137,  200,    8,   56,  104,
     1  152,  215,   23,   71,  119,  167,  230,   38,   86,
     1  134,  182,    5,   53,  101,  149,  197,   20,   68,
     1  116,  164,  212,   35,   83,  131,  179,  227,   50,
     1   98,  146,  194,    2,   65,  113,  161,  209,   17,
     1  160,  208,   16,   64,  112,  175,  223,   31,   79,
     1  127,  190,  238,   46,   94,  142,  205,   13,   61,
     1  109,  157,  220,   28,   76,  124,  172,  235,   43,
     1   91,  139,  187,   10,   58,  106,  154,  202,   25,
     1   73,  121,  169,  217,   40,   88,  136,  184,  232,
     1   55,  103,  151,  199,    7,   70,  118,  166,  214,
     1   22,   85,  133,  181,  229,   37,  100,  148,  196,
     1    4,   52,  115,  163,  211,   19,   67,  130,  178,
     1  226,   34,   82,  145,  193,    1,   49,   97/
      DO 50 I=1,250
50    X(I)=0
      IND=1
      COUNT=1
      IF(FWD.NE.1)COUNT=55
      DO 100 I=1,240
100   X(I)=Y(RF(I)+1)
      DO 200 INDEX=1,80
      TEMP=CA(X(30+INDEX),X(160+INDEX))
      X(160+INDEX)=CS(X(30+INDEX),X(160+INDEX))
      X(30+INDEX)=TEMP
      X(INDEX)=CA(X(INDEX),TEMP)
200   CONTINUE
      DO 300 ROUND=1,3
      DISP=30*(ROUND-1)
      DO 400 T3CNT=1,5
      SLIDE=DISP+T3CNT
      T(1)=CA(X(SLIDE),X(SLIDE+40))
      T(2)=CA(X(SLIDE+20),X(SLIDE+60))
      T(3)=CA(X(SLIDE+10),X(SLIDE+50))
      T(4)=CS(X(SLIDE+10),X(SLIDE+50))
      T(5)=CA(X(SLIDE+30),X(SLIDE+70))
      T(6)=CS(X(SLIDE+30),X(SLIDE+70))
      T(7)=CA(X(SLIDE+5),X(SLIDE+45))
      T(8)=CS(X(SLIDE+5),X(SLIDE+45))
      T(9)=CA(X(SLIDE+15),X(SLIDE+55))
      T(10)=CS(X(SLIDE+15),X(SLIDE+55))
      T(11)=CA(X(SLIDE+25),X(SLIDE+65))
      T(12)=CS(X(SLIDE+25),X(SLIDE+65))
      T(13)=CA(X(SLIDE+35),X(SLIDE+75))
      T(14)=CS(X(SLIDE+35),X(SLIDE+75))
      T(15)=CA(T(1),T(2))
      T(16)=CA(T(3),T(5))
      T(17)=CA(T(15),T(16))
      T(18)=CA(T(7),T(11))
      T(19)=CS(T(7),T(11))
      T(20)=CA(T(9),T(13))
```

APPENDIX E

```
      T(21)=CS(T(9),T(13))
      T(22)=CA(T(18),T(20))
      T(23)=CA(T(8),T(14))
      T(24)=CS(T(8),T(14))
      T(25)=CA(T(10),T(12))
      T(26)=CS(T(12),T(10))
      X(SLIDE+60)=CS(X(SLIDE+20),X(SLIDE+60))
      X(SLIDE+20)=CS(X(SLIDE),X(SLIDE+40))
      X(SLIDE)=CA(T(17),T(22))
      X(SLIDE+5)=CS(T(17),T(22))
      X(SLIDE+10)=CS(T(15),T(16))
      X(SLIDE+15)=CS(T(1),T(2))
      X(SLIDE+25)=CS(T(19),T(21))
      X(SLIDE+30)=CS(T(4),T(6))
      X(SLIDE+35)=CA(T(24),T(26))
      X(SLIDE+40)=CMOD(T(24))
      X(SLIDE+45)=CMOD(T(26))
      X(SLIDE+50)=CS(T(18),T(20))
      X(SLIDE+55)=CS(T(3),T(5))
      X(SLIDE+65)=CA(T(19),T(21))
      X(SLIDE+70)=CA(T(4),T(6))
      X(SLIDE+75)=CA(T(23),T(25))
      X(T3CNT+240)=CMOD(T(23))
      X(T3CNT+245)=CMOD(T(25))
  400 CONTINUE
      DO 500 TIMES=1,16
  500 CALL C240U5(DISP+5*(TIMES-1))
      CALL C240U5(240)
      CALL C240U5(245)
      DO 600 T3CNT=1,5
      SLIDE=DISP+T3CNT
      T(1)=CA(X(SLIDE+15),X(SLIDE+25))
      T(2)=CS(X(SLIDE+15),X(SLIDE+25))
      T(3)=CA(X(SLIDE+55),X(SLIDE+65))
      T(4)=CS(X(SLIDE+65),X(SLIDE+55))
      T(5)=CA(X(SLIDE+20),X(SLIDE+30))
      T(6)=CS(X(SLIDE+20),X(SLIDE+30))
      T(7)=CS(X(SLIDE+40),X(SLIDE+35))
      T(8)=CS(X(SLIDE+45),X(SLIDE+35))
      T(9)=CA(T(5),T(7))
      T(10)=CS(T(5),T(7))
      T(11)=CA(T(6),T(8))
      T(12)=CS(T(6),T(8))
      T(13)=CA(X(SLIDE+60),X(SLIDE+70))
      T(14)=CS(X(SLIDE+60),X(SLIDE+70))
      T(15)=CA(X(SLIDE+75),X(T3CNT+240))
      T(16)=CS(X(SLIDE+75),X(T3CNT+245))
      T(17)=CA(T(13),T(15))
      T(18)=CS(T(13),T(15))
      T(19)=CA(T(14),T(16))
      T(20)=CS(T(14),T(16))
      X(SLIDE)=X(SLIDE)
      X(SLIDE+60)=CS(X(SLIDE+10),X(SLIDE+50))
      X(SLIDE+40)=CMOD(X(SLIDE+5))
```

APPENDIX E

```
      X(SLIDE+20)=CA(X(SLIDE+10),X(SLIDE+50))
      X(SLIDE+5)=CA(T(9),T(17))
      X(SLIDE+10)=CA(T(1),T(3))
      X(SLIDE+15)=CS(T(12),T(20))
      X(SLIDE+25)=CA(T(11),T(19))
      X(SLIDE+30)=CA(T(2),T(4))
      X(SLIDE+35)=CS(T(10),T(18))
      X(SLIDE+45)=CA(T(10),T(18))
      X(SLIDE+50)=CS(T(2),T(4))
      X(SLIDE+55)=CS(T(11),T(19))
      X(SLIDE+65)=CA(T(12),T(20))
      X(SLIDE+70)=CS(T(1),T(3))
      X(SLIDE+75)=CS(T(9),T(17))
600   CONTINUE
300   CONTINUE
      DO 900 INDEX=1,30
      TEMP=CA(X(INDEX),X(80+INDEX))
      TEMP2=CS(TEMP,X(160+INDEX))
      X(80+INDEX)=CA(TEMP,X(160+INDEX))
      X(160+INDEX)=CMOD(TEMP2)
900   CONTINUE
      DO 950 I=1,240
950   Y(RFI(I)+1)=X(I)
      RETURN
      END
      SUBROUTINE C240U5(INDEX)
      IMPLICIT REAL*8 (A-D),INTEGER(E-Z)
      REAL*8 X(250),S(6),TS(5)
      INTEGER COUNT
      COMMON/CARRAY/X,COUNT
      INTEGER POINT( 108)
      DATA POINT /    0,     0,     0,     0,     0,     6,     6,
     1   12,    18,    24,    30,    30,    30,    36,    36,    42,
     1   48,    54,    60,    60,    60,    60,    60,    66,    66,
     1   72,    78,    84,    90,    90,    90,    96,    96,   102,
     1  108,   114,   120,   120,   120,   120,   120,   126,   126,
     1  132,   138,   144,   150,   150,   150,   156,   156,   162,
     1  168,   174,   180,   180,   180,   180,   180,   186,   186,
     1  192,   198,   204,   210,   210,   210,   216,   216,   222,
     1  228,   234,   240,   240,   240,   240,   240,   246,   246,
     1  252,   258,   264,   270,   270,   270,   276,   276,   282,
     1  288,   294,   300,   300,   300,   300,   300,   306,   306,
     1  312,   318,   324,   330,   330,   330,   336,   336,   342,
     1  348,   354/
      INTEGER CCOEF(   720)
      DATA CCOEF /    1,     0,49138,     0,64206,     0,     0,
     157700,     0,54097,     0, 4216, 4860,     0,59444,     0,
     139682,     0,     0,  680,     0,49192,     0,47832,52183,
     1     0,15670,     0,16595,     0,     0,33455,     0,57636,
     1     0,56245, 9509,     0,37253,     0,28812,     0,     0,
     113194,     0,13704,     0,57335,29338,     0,61606,     0,
     1 4378,     0,     0,53716,     0,31049,     0,54655,     0,
     1    1,     0,49133,     0,64206, 7819,     0,11422,     0,
     161303,     0,     0, 4860,     0,59444,     0,39682,64339,
```

APPENDIX E

```
1    0,16327,     0,17637,     0,     0,22845,     0,20533,
1    0,12217,20261,     0,38932,     0,63929,     0,     0,
129338,     0,51606,     0, 4378,11303,     0,34470,     0,
110864,     0,     0, 9509,     0,37253,     0,28812,52325,
1    0,46815,     0, 7684,     0,32758,     0,57331,     0,
134729,     0,     0,44438,     0,17133,     0,59195,58229,
1    0,41872,     0, 5996,     0,     0,64499,     0,57250,
1    0,59290,20004,     0,40514,     0, 7367,     0,     0,
148096,     0,44584,     0,13911,18496,     0,42399,     0,
122301,     0,     0,45728,     0,37463,     0,11526,21512,
1    0,38629,     0,58952,     0,     0,50464,     0,51705,
1    0,16296,     0,32758,     0,57331,     0,34729,21031,
1    0,48386,     0, 6324,     0,     0,58229,     0,41872,
1    0, 5996, 1020,     0, 8269,     0, 6229,     0,     0,
164011,     0, 1885,     0,14434, 2368,     0, 7121,     0,
1 2385,     0,     0,21512,     0,38629,     0,58952,15055,
1    0,13314,     0,49223,     0,     0,18496,     0,42399,
1    0,22301,19791,     0,28056,     0,53993,     0,     0,
137606,     0,51271,     0,24643,57561,     0,58687,     0,
1 9034,     0,     0,32669,     0, 8303,     0,20548,45349,
1    0,14613,     0,53953,     0,     0,34329,     0, 6228,
1    0, 3095,52627,     0,40142,     0,  407,     0,     0,
158271,     0, 9060,     0,16369, 1823,     0,29360,     0,
125714,     0,     0,10387,     0, 3396,     0,55340,37912,
1    0,50924,     0,40619,     0,27913,     0,14248,     0,
140871,     0,     0,57561,     0,58637,     0, 9034,32850,
1    0,57216,     0,44971,     0,     0,45849,     0,14613,
1    0,53953,41577,     0,62687,     0,52245,     0,     0,
114715,     0,54737,     0,25307,55132,     0,62123,     0,
110179,     0,     0,37912,     0,50924,     0,40619, 7248,
1    0,56459,     0,49150,     0,     0, 1823,     0,29360,
1    0,25714,  273,     0,48798,     0,34665,     0,     0,
137979,     0,38813,     0,28374,16400,     0,45019,     0,
122551,     0,     0,10917,     0, 1979,     0,45664,28336,
1    0,30099,     0, 9624,     0,     0,39445,     0,55451,
1    0,42080,40716,     0,14624,     0, 3396,     0,     0,
1 1583,     0, 4290,     0, 1124,15956,     0,45574,     0,
115852,     0,     0,11788,     0,41093,     0,17517,     0,
165246,     0,16721,     0,30854,37979,     0,38813,     0,
128374,     0,     0,49119,     0,20500,     0,42968,10917,
1    0, 1979,     0,45664,     0,     0,53139,     0,15475,
1    0, 6228,27657,     0,14358,     0,24563,     0,     0,
149563,     0,19945,     0,49667,11788,     0,41093,     0,
117517,     0,     0,24803,     0,50895,     0,62123, 1583,
1    0, 4290,     0, 1124,     0,32350,     0,57841,     0,
146281,     0,     0,41310,     0,40059,     0,22958,40919,
1    0,30750,     0,64452,     0,     0,16384,     0,29791,
1    0,62542,23015,     0,53130,     0,51083,     0,     0,
139111,     0,15102,     0, 2399, 4445,     0,43583,     0,
160425,     0,     0,30335,     0,59084,     0,63833,41595,
1    0,62677,     0,41741,     0,     0,47337,     0,36639,
1    0, 6494,     0,33169,     0, 7673,     0,19238,41319,
1    0,40059,     0,22958,     0,     0,24600,     0,34769,
1    0, 1067,16384,     0,29791,     0,62542,     0,     0,
```

APPENDIX E

```
     113570,    0, 9547,    0,56177,56793,    0,43982,    0,
     161434,    0,    0,23934,    0, 2842,    0,23778,47837,
     1    0,36639,    0, 6484,    0,    0,61074,    0,21936,
     1    0, 5094,30385,    0,59034,    0,63333,    0,    0,
     120045,    0,24083,    0,19553,55112,    0,34915,    0,
     155729,    0,    0,57466,    0,26446,    0,25030, 2648,
     1    0,53209,    0,52913,    0,    0,62919,    0, 3250,
     1    0, 6812,18510,    0,17093,    0,45592,    0,    0,
     113134,    0,16342,    0,52074,39046,    0,21962,    0,
     1 9389,    0,    0,47185,    0,55677,    0,27069,63493,
     1    0,12224,    0,16276,    0,20045,    0,24083,    0,
     119553,    0,    0,10407,    0,30604,    0, 9790,57466,
     1    0,26446,    0,25030,    0,    0,62871,    0, 7310,
     1    0,12606,15734,    0,13092,    0,45262,    0,    0,
     144983,    0,60650,    0,36203,47135,    0,55677,    0,
     127069,    0,    0, 2026,    0,53295,    0,49243,13134,
     1    0,16342,    0,52074,    0,    0,26473,    0,43557,
     1    0,56130/
      REAL*8 DCOEF( 360)
      EQUIVALENCE(DCOEF(1),CCOEF(1 ))
      TS(1)=CA(X(INDEX+2),X(INDEX+5))
      TS(2)=CA(X(INDEX+3),X(INDEX+4))
      S(6)=CS(X(INDEX+2),X(INDEX+5))
      S(5)=CS(X(INDEX+4),X(INDEX+3))
      S(4)=CA(S(5),S(6))
      S(3)=CS(TS(1),TS(2))
      S(2)=CA(TS(1),TS(2))
      S(1)=CA(X(INDEX+1),S(2))
      DO 250 I=1,6
  250 S(I)=CM(S(I),DCOEF(POINT(COUNT)+I))
      COUNT=COUNT+1
      TS(1)=CA(S(1),S(2))
      TS(2)=CA(TS(1),S(3))
      TS(3)=CS(S(4),S(5))
      TS(4)=CS(TS(1),S(3))
      TS(5)=CA(S(4),S(6))
      X(INDEX+5)=CS(TS(2),TS(3))
      X(INDEX+4)=CS(TS(4),TS(5))
      X(INDEX+3)=CA(TS(4),TS(5))
      X(INDEX+2)=CA(TS(2),TS(3))
      X(INDEX+1)=S(1)
      RETURN
      END
```

APPENDIX F

ELEMENTARY THEORY OF ERROR CORRECTING CODES

The material presented in this appendix introduces the basic nomenclature of error correcting codes and serves to relate the material presented in chapter 6 to the work described in the main part of the thesis.

Digital communication is generally achieved by pulse code modulation where each sample of the signal is represented by a codeword of n symbols. For binary systems the symbols are binary bits. These symbols are transmitted and the role of the receiver is to recognise each code word in order to reconstruct the appropiate samples. However errors may occur in transmission as a result of noise. One way to improve the reliability of communication in the presence of noise is to increase the signal to noise ratio. An alternative is to add extra symbols, at the expense of an increase in bandwith, and then detect and possibly correct the errors.

Let a given codeword contain k message symbols and r check symbols. The total number of symbols per word is given by n:

$$n = k + r \qquad \text{(AF.1)}$$

and such a code is referred to as an (n,k) code. The code rate efficiency is defined as k/n and this is an indication of the information rate of the code.

For example it can be seen that an effective error detecting code can be formed by adding an extra symbol at the end of each codeword. This extra symbol is called a parity bit. The simple parity check code is a (k + 1, k) code having a rate efficiency of k/(k+1). It offers a simple and effective method for error detection when the probibility of an error occurring is low. When an error does occur a retransmission can be requested. It should be noted that an error in two symbols of a binary codeword is not detected.

In certain cases it is not possible to request a retransmission of data; however it is possible to add sufficient redundancy into the code so that errrors are not only detected but also corrected.

Let us consider a binary repetetion code which transmits three zeros for every '0' and a sequence of three ones for every '1'. This is an example of a (3,1) code. If we can be certain that not more than one error will occur in each codeword then the following codewords 001, 010, 100 will be decoded as 000 and 110, 101, 011 will be decoded as 111. In this case a majority voting rule may be used for decoding. However if two errors occur in a codeword then the codeword would be incorrectly decoded. Thus a single error correction code is most useful when the probability of error is low.

The distance between two codewords s_1 and s_2 can be defined as the number of symbols in which s_1 and s_2 differ. For example the distance between the two codewords in the 3 bit repetition code is three and it can be seen that the decoding procedure is obtained by choosing the valid codeword with the minimum distance from the received codeword.

It can be seen that in the light of this example that if a given code is capable of correcting t errors then the minimum distance between codewords must satisfy:

$$d > 2t + 1 \qquad \text{(AF.2)}$$

In general it is possible to say that a valid codeword will satisfy r (r = n - k) linear independent equations. These equations can conveniently be expressed in terms of an (n x 1) column matrix $\underline{v}$ representing the codeword and a rectangular (r x n) check matrix $\underline{h}$ and hence

$$\underline{h} \circ \underline{v} = 0 \qquad \text{(AF.3)}$$

Let the received codeword be $\underline{r}$ which may or may not be equal to $\underline{v}$. If $\underline{h} \circ \underline{r} = 0$ we know that $\underline{r}$ is a valid codeword and most likely it is the transmitted codeword. If however $\underline{h} \circ \underline{r} \neq 0$ then $\underline{r}$ is not a codeword and at least one error has been made.

Let the matrix $\underline{e}$ denote an error vector, such that

$$\underline{r} = \underline{v} + \underline{e} \qquad \text{(AF.4)}$$

hence if $\underline{e}$ contains all zeros then no error has been made.

In order to correct the errors we need to determine $\underline{e}$. Let us first determine a matrix $\underline{s}$, called the syndrome, from the received codeword and the check matrix.

$$\underline{s} = \underline{h} \circ \underline{r} \qquad \text{(AF.5)}$$

$$= \underline{h} \circ \underline{v} + \underline{h} \circ \underline{e}$$

$$\underline{s} = \underline{h} \circ \underline{e} \qquad \text{(AF.6)}$$

The task of the decoder is to select one of the several error sequences which are associated with a given syndrome. Ordinarily this selection is based on a minimum distance criterion and codes are usually designed so that the computation of $\underline{s}$ gives an easy technique for determining the error locator sequence $\underline{e}$, and from this, the original codeword $\underline{v}$ may be determined.

The literature on error correcting codes shows much interest in designing codes where each symbol is composed only of a single binary bit. From an algebraic viewpoint such codes can be said to be defined over GF(2). However processors handle data most efficiently in full bytes and so interest is increasing in defining codes over other fields i.e. over GF(q^n).

Reed, Truong and Welch (93) and Justesen (38) have defined Reed-Soloman (85) error correcting codes over GF(q) which can be encoded and decoded using number theoretic transform techniques. MacWilliams and Sloane (121) have shown that Reed-Soloman (RS) codes are maximum distance separable (MDS) which means that an (n,k) RS code will have a minimum distance (d) between codewords of

$$d = n - k + 1 \qquad \text{(AF.7)}$$

or

$$d = r + 1 \qquad \text{(AF.8)}$$

where r is the number of check symbols in the codeword. It can therefore be seen that such codes can correct t errors provided

$$2t + 1 < r + 1$$

or

$$2t < r \qquad \text{(AF.9)}$$

Therefore a Reed-Soloman code with r check symbols can correct upto int(r/2) errors.

The encoding and decoding procedures for such codes are discussed in chapter 6 where it is hoped that the examples presented are sufficient to illustrate the techniques involved.

REFERENCES

(1) R.C.Agarwal C.S.Burrus,
' Fast Convolution using Fermat Number Theoretic Transforms with applications to Digital filtering'
IEEE Trans. Accoustics Speech and Signal Processing
vol ASSP-22 no 2 April 1974 pp 87-97

(2) R.C.Agarwal C.S.Burrus,
' Fast one dimensional Convolution by multidimensional techniques'
IEEE Trans. Accoustics Speech and Signal Processing
vol ASSP-22 February 1974 pp 1-10

(3) R.C.Agarwal C.S.Burrus,
' Number Theoretic Transforms to implement fast Digital Convolution'
Proc. IEEE vol 63 no 4 April 1975 pp 550-60

(4) R.C.Agarwal C.S.Burrus,
' Fast one Dimensional Digital Convolution by Multi-dimensional techniques'
Southwest IEEE conf. Rec. Houston Texas April 1973
pp 538-543

(5) R.C.Agarwal,
' On realisation of Digital filters'
Phd. Dissertation
Dept. Elect. Eng. Rice Univ. Houston Texas

(6) R.C.Agarwal J.C.Cooley,
' Some new algorithms for fast Convolution by multidimensional techniques'
IEEE Arden House workshop on Digital Signal Proc.
Harriman N.Y. February 1976

(7) R.C.Agarwal J.C.Cooley,
' New algorithms for Digital Convolution'
IEEE Trans. Accoustics Speech and Signal Processing
vol ASSP-25 no 5 October 1977 pp 392-410

(8) R.C.Agarwal J.C.Cooley,
' New Algorithms for Digital Convolution.'
IBM Research report No.6446

(9) D.Bailey,
' Winograd's Algorithm applied to Number Theoretic Transforms'
Electronics Letters 1st September 1977 pp 548-9'

REFERENCES

(10) E.R.Berlekamp,
' Algebraic coding theory'
McGraw Hill N.Y. 1968

(11) R.Bernstein,
' Schnelle Faltung mit der Rader Transformation'
Diplomarbeit Institut für Nachrichtentechnis
Universität Erlangen-Nürnberg.

(12) L.I.Bluestein,
' A linear filtering approach to the computation of the DFT.'
Northeast Electronics Research and Engineering meeting
Rec. Vol 10 1968 pp 218-9

(13) R.N.Bracewell,
' The Fourier Transform and its applications'
McGraw-Hill 1978

(14) E.O.Brigham,
' The Fast Fourier Transform'
Prentice-Hall Englewood-Cliffs N.J. 1974

(15) J.D.Brule,
' Fast Convolution with Finite Field Transforms'
IEEE Trans. Accoustics Speech and Signal Processing
vol ASSP-23 April 1975 p 240

(16) P.R.Chevillat,
' Transform domain Digital filtering with Number Theoretic Transforms and limited wordlengths'
IEEE Trans. Accoustics Speech and Signal Processing
vol ASSP-26 no 4 August 1978 pp 284-290

(17) J.C.Cooley P.A.Lewis P.D.Welch,
' Historical Notes on the Fast Fourier Transform'
Trans. Audio Electroacoust.
Vol AU-15 June 1967 pp 76-79

(18) J.C.Cooley J.W.Tukey,
' An algorithm for machine calculation of complex Fourier series'
Math. Comput. Vol 19 1965 pp 297-301

REFERENCES

(19) W.F.Davis,
' A Class of efficient Convolution algoritnms'
Proc. Symp. Applications of Walsh Functions
Washington DC March 1972 pp 310-329

(20) R.C.Debnath D.A.Pucknell,
' On multiplicative overflow detection in a residue number system'
Electronics Letters 2nd March 1973 pp 129-130

(21) M.F.A.Derome,
' Fast Convolution of large one and two-dimensional arrays using Number Theoretic Transforms based on three bit primes'
Optik (Germany) vol.49 No.4 1978 pp 467-77

(22) E.Dubois A.N.Venetsanopoulos,
' Convolution using a conjugate symmetry property for the generalized DFT'
IEEE Trans. Accoustics Speech and Signal Processing
vol.ASSP- 26 no 2 April 1976 pp 165-170

(23) E.Dubois A.N.Venetsanopoulos,
' Fast Convolution using the DFT over commutative rings with identity'
Proc. 20th Midwest Symposium circuits & systems
1977 pp 687-691

(24) E.Dubois A.N.Venetsanopoulos,
' The DFT over finite rings with applications to fast computation'
IEEE Trans. Computers
vol.C-27 no 7 July 1978 pp 586-593

(25) J.W.Eastwood C.R.Jesshope,
' The solution of partial differential equations using Number Theoretic Transforms with application to narrow or limited computer hardware'
Comput. Phys. Commun.(Netherlands), vol.13 No.4
1977 pp 233-9

(26) P.J.Edelsky,
' Exact Convolutions by Number Theoretic Transforms'
Naval Research Centre 2 May 1975
Report AD-A013395

REFERENCES

(27) H.L.Garner,
' The Residue Number System'
I.R.E. Trans. Electron. June 1959 pp 140-147

(28) B.Gold C.M.Rader,
' Digital Processing of signals'
McGraw-Hill N.Y. 1969

(29) S.W.Golomb I.S.Reed T.K.Truong,
' Integer Convolution over the finite field
G.F.(3.2n+1)'
SIAM J.Appl.Math 1977 vol-32

(30) I.J.Good,
' The interaction algorithm and practical Fourier
analysis'
J.Royal Stat.Soc ser B 20 1958 pp 361-372
addendum vol 22 1960 pp 372-375

(31) G.H.Hardy E.M.Wright,
' The Theory of numbers'
Oxford univ. 1960

(32) L.E.Heindel E.Horowitz',
' On decreasing the computation for modular
arithmetic'
12th annual symposium on switching and automation
theory
conf. Paper 13-15 October1971 East Lansing Mich.
U.S.A

(33) I.N.Herstein,
' Topics in algebra'
Waltham Mass. Ginn 1964.

(34) W.K.Jenkins B.J.Leon,
' The use of residue number systems in the design of
finite impulse response Digital filters'
IEEE Trans. Circuits systems April 1977 pp 191-201

(35) W.K.Jenkins,
' Composite Number Theoretic Transforms for Digital
filtering'
9th Asilomar conf. on circuits systems and computers

REFERENCES

(36) W.K.Jenkins,
' Use of residue coding in the design of hardware for non-recursive Digital filters'
8th Asilomar conf. on circuits systems and computers

(37) W.K.Jenkins,
' Recent advances in Residue number techniques for Recursive Digital filtering'
Trans. Accoustics Speech and Signal Processing
vol ASSP-27 no.1 February 1979 pp 19-30

(38) Jørn Justesen,
' On the complexity of decoding Reed-Solomon codes'
Trans. Information Theory
IEEE vol.I.T.-2 No.2 March 1976 pp 237-8

(39) D.Kibler,
' Neccessary and Sufficient Conditions for the existence of the modular Foutier Transform'
Comments on (3)
Proc. IEEE February 1977 pp 265-7

(40) D.P.Kolba T.W.Parks,
' A Prime Factor FFT algorithm using high speed Convolution'
Trans. Accoustics Speech and Signal Processing
vol ASSP-25 August 1977

(41) T.A.Kriz D.A.Bachman,
' Computational Efficiency of Number Theoretic Transform implemented Finite Impulse Response Filters'
Electronics Letters vol.14 No.23 9th November 1978
pp 731-3

(42) W.Lederman,
' Introduction to the theory of finite groups'
Oliver & Boyd London 1957

(43) L.M.Leibowitz,
' Fast Convolution by Number Theoretic Transforms'
Naval Res. Lab. Washington D.C. N.R.L.
Report AD-A015276 September 12 1975

(44) L.M.Leibowitz,
' A simplified Binary arithmetic for the Fermat Number Transforms'
Trans. Accoustics Speech and Signal Processing
IEEE Trans. Vol ASSP-24 no.5 October 1976 pp 356-9

REFERENCES

(45) D.Mandlebaum,
' on decoding of Reed-Soloman codes'
Trans. Information Theory
vol.I.T.-17 no.6 November 1971 pp 707-712

(46) K.Y.Liu I.S.Reed T.K.Truong,
' Fast Number Theoretic Transforms for Digital filtering'
Electronics Letters 25th vol.12 No.24 25th November
1976 pp 644-6

(47) J.H.McClellan,
' Hardware realization of a Fermat Number Transforms'
Trans. Accoustics Speech and Signal Processing
IEEE vol ASSP-24 June 1976 pp 216-225

(48) P.Melnuish,
' Fermat Number Transforms implementation by a minicomputer'
Electronics Letters 5th March 1975 vol.11 No.5
Pp 109-111

(49) L.R.Morris,
' A comparitive study of time efficient FFT and WFTA programs for general purpose computers'
Trans. Accoustics Speech and Signal Processing
IEEE vol.ASSP-26 No.2 April 1978 pp 141-150

(50) Hideo Murakami I.S.Reed,
' Recursive realization of finite impulse filters using finite field arithmetic'
Trans. Information Theory
IEEE vol IT-23 no.2 March 1977 pp 232-242

(51) Hideo Murakami I.S.Reed,
' Multichannel convolutional coding systems over a direct sum of Galois fields'
Trans. Information Theory
IEEE vol IT-24 no.2 March. 1976 pp 205-211

(52) Toheo Nakamura,
' Fast algorithms for computing the powers of a primitive element in G.F.(q?)'
Trans. Accoustics Speech and Signal Processing
IEEE vol.ASSP-26 no.4 August 1978

(53) R.L.Nevin,
' Application of the Rader-Brenner FFT algortinm to Number Theoretic Transforms'
Trans. Accoustics Speech and Signal Processing
IEEE ASSP vol.-25 No.2 April 1977 pp 196-8

REFERENCES

(54) R.L.Nevin,
' Phd. Dissertation'
Complex Convolution using Radix-2 Number Theoretic Transforms
Syracuse University N.Y. October 1976

(55) R.L.Nevin,
' Methods and Modulii for Number Theoretic Transform Convolution'
IEEE Computer Society Repository

(56) P.J.Nicholson,
' Algebraic theory of the finite Fourier Transform'
J. Comput.Syst.Sci vol.5 1971 pp 524-47

(57) H.J.Nussbaumer,
' Complex Convolution via Fermat Number Transforms'
IBM J.Res.Develop vol.20 May 1976 pp 282-4

(58) H.J.Nussbaumer,
' Digital filtering using Mersenne Transforms'
IBM J.Res.Develop vol.20 September 1976 pp 498-504

(59) H.J.Nussbaumer,
' Digital filtering using pseudo-Fermat Number Transforms'
Trans. Accoustics Speech and Signal Processing
IEEE vol.ASSP-23 no.1 February 1977 pp 79-83

(60) H.J.Nussbaumer,
' Digital filtering using Polynomial Transforms'
Electronics Letters 23rd June 1977 vol.13 No.13
pp 386-7

(61) H.J.Nussbaumer,
' Linear Filtering technique for computing Mersenne and Fermat Number Transforms'
IBM J.Res.Develop July 1977 pp 334-339

(62) H.J.Nussbaumer,
' Digital filtering using cotranforms in finite fields'
Electronics Letters vol.12 No.5 4th March 1976
pp 113-4

REFERENCES

(63) H.J.Nussbaumer,
' Generalized Mersenne and Fermat Transforms'
IBM tech disl. Bull. Vol.19 No.5 October 1976

(64) H.J.Nussbaumer,
' Relative evaluation of various Number Theoretic Transforms for Digital filtering applications'
Trans. Accoustics Speech and Signal Processing
IEEE vol.ASSP-26 no.1 February 1978 pp 83-93

(65) H.J.Nussbaumer,
' Overflow detection in the computation of Convolution by some Number Theoretic Transforms'
Trans. Accoustics Speech and Signal Processing
IEEE vol.ASSP-26 no.1 February 1978 pp 108-109

(66) H.J.Nussbaumer,
' Fast multipliers for Number Theoretic Transforms'
Trans. Computers
IEEE vol.C-27 no.8 August 1978 pp 764-5

(67) A.V.Oppenheim R.W.Schafer,
' Digital Signal Processing'
Prentice-Hall Englewood Cliffs N.J. 1975

(68) O.Ore,
' Number theory and its history'
McGraw-hill 1948

(69) R.W.Patterson J.H.McClellan,
' Fixed point error analysis of WFTA'
Trans. Accoustics Speech and Signal Processing
IEEE vol.ASSP-26 no.5 October 1978 pp 447-455

(70) W.W.Peterson,
' Error Correcting Codes'
M.I.T. Press Cambridge MA. 1961

(71) D.A.Pitassi,
' Fast Convolution using Walsh Functions'
Proc. Conf. Applications of Walsh Functions
Washington D.C. April 1971 pp 130-133

(72) J.M.Pollard,
' The Fast Fourier Transform in a finite field'
Math.Computing vol-25 April 1971 pp 365-374

REFERENCES

(73) J.M.Pollard,
' Implementation of Number Theoretic Transforms'
Electronics Letters vol.12 No.15 22nd July 1976
pp 378-9

(74) L.R.Rabiner B.Gold,
' Theory and application of Digital signal
processing'
Prentice-Hall 1975

(75) L.R.Rabiner C.M.Rader,
' Digital signals Processing'
IEEE Press N.Y. 1972

(76) C.M.Rader,
' The Number Theoretic DFT & exact discrete
Convolution'
IEEE Arden House Workshop on Digital signals
processing
Harriiman N.Y. January 1972

(77) C.M.Rader,
' Discrete Convolution via Mersenne Transforms'
Trans. Computers
IEEE vol.C-21 December 1972 pp 1269-73

(78) C.M.Rader,
' On the application of Number Theoretic Transforms
of high speed Convolution to 2-D filtering'
Trans. Circuit Theory
IEEE Journal paper vol.CAS-22 No.6 June 1975 p 575

(79) C.M.Rader,
' Convolution & Correlation using Number Theoretic
Transforms'
in (74) pp 419-433

(80) C.M.Rader,
' Discrete Fourier Transforms when the Number of
Samples is Prime'
Proc. IEEE vol 56 June 1968 pp 1107-8

(81) P.J.Rayner,
' A Fast Cyclic Convolution Algorithm'
Symposium on Digital filtering
Imperial College London August 1971

REFERENCES

(32) P.J.Rayner G.Tacconi,
' Number Theoretic Transforms'
Proc. NATO Advanced Study Institute of Signal Processing
30th August -11th September 1976 pp 333-53

(33) N.S.Reddy V.U.Reddy,
' Complex Convolution using rectangular transforms'
Electronics Letters vol.14 No.15 20th July 1978
pp 458-9

(34) N.S.Reddy V.U.Reddy,
' Implementation of Winograds algorithm in modular arithmetic for digital Convolutions'
Electronics Letters vol.14 No.7 30th March 1978
pp 228-9

(35) I.S.Reed G.Soloman,
' Polynomial codes over certain finite fields'
J.Soc.Ind.Maths. Vol.8 June 1960 pp 300-4

(36) I.S.Reed T.K.Truong,
' The use of Finite Fields to compute Convolutions'
Trans. Information Theory
IEEE vol.IT-21 No.2 March 1975 pp 208-213

(37) I.S.Reed T.K.Truong,
' Complex integer Convolution over a sum of Galois fields'
Trans. Information Theory
IEEE vol.IT-21 November 1975 pp 657-661

(38) I.S.Reed T.K.Truong,
' Convolutions over Residue classes of quadratic integers'
Trans. Information Theory
IEEE vol.IT-22 no.4 July 1976 pp 468-475

(39) I.S.Reed T.K.Truong,
' A fast DFT algorithm using complex integer transforms'
Electronics Letters vol.14 No.6 16th March 1978
pp 191-3

(90) I.S.Reed T.K.Truong,
' Addendum to :
A fast DFT algorithm using complex number theoretic transforms'
Electronics Letters vol.14 No.7 30th March 1978
pp 231-2

REFERENCES

(91) I.S.Reed T.K.Truong,
' A fast computation of complex comvoltions using a hybrid transform'
Trans. Accoustics Speech and Signal Processing
vol.ASSP-26 no.6 December 1978 pp 566-70

(92) I.S.Reed T.K.Truong R.L.Miller,
' Correction to :
A fast DFT algorithm using complex number theoretic transforms'
Electronics Letters 22th June 1978 p 411

(93) I.S.Reed T.K.Truong L.R.Welch,
' The fast decoding of Reed-Soloman codes using Fermat Transforms'
Trans. Information Theory
IEEE vol.IT-24 no.4 July 1978 pp 497-9

(94) I.S.Reed T.K.Truong R.L.Miller,
' A simple method for computing elements of order...'
Electronics Letters vol.14 No.21 12th October 1978
pp 697-8

(95) I.S.Reed T.K.Truong R.L.Miller,
' A fast algorithm for computing a primitive ...'
Electronics Letters vol.14 No.15 20th July 1978
pp 493-4

(96) I.S.Reed,
' The use of finite fields and rings to compute Convolutions'
Lab. Tech. Memo 24L-0012 M.I.T. Lincoln Lab. 1973

(97) I.S.Reed K.Y.Liu,
' Fast algorithm for computing complex Number Theoretic Transforms'
Electronics Letters vol.13 No.10 12th May 1977
pp 278-80

(98) I.S.Reed T.K.Truong L.R.Welch,
' The fast decoding of Reed-Soloman codes using Number Theoretic Transforms'
Deep Space Network Progress Report 42-35
Jet Propulsion Laboratory, Pasadena CA July 1976
pp 64-78

REFERENCES

(99) I.S.Reed R.A.Scholtz T.K.Truong L.R.Welch,
' The fast decoding of Reed-Soloman codes using
Fermat Theoretic Transforms and continued fractions'
Trans. Information Theory
IEEE vol.IT-24 No.1 January 1978 pp 100-106

(100) D.V.Sarwate,
' Comments on (94)'
Electronics Letters vol.15 No.2 18th January 1979
pp 67-70

(101) H.F.Silverman,
' An Introduction to programming the Winograd
transform algorithm'
Trans. Accoustics Speech and Signal Processing
IEEE vol ASSP-25 no.2 April 1977 pp 152-165

(102) H.F.Silverman,
' Corrections and addendum to
An Introduction to programming the Winograd transform
algorithm'
Trans. Accoustics Speech and Signal Processing
IEEE ASSP vol.26 No.3 June 1978 p 268

(103) H.F.Silverman,
' A method for programming the complex general-N
Winograd Fourier Transform Algorithm'
Proc. Int conf. IEEE ASSP Hartford ST May 9-11
1977.

(104) Shu Lin,
' An introduction to Error Correcting Codes'
Prentice-Hall Englewood Cliffs N.J. 1970

(105) R.M.Thrall L.Tornheim,
' Vector Spaces and Matrices'
Wiley N.Y. 1957

(106) T.K.Truong,
' The Application of Finite Fields to Digital
Filters'
Phd. Dissertation
University of Southern California October 1975.

(107) B.D.Tseng W.C.Miller,
' comments on (101)'
Trans. Accoustics Speech and Signal Processing
IEEE vol.ASSP-26 no.3 June 1978 pp 268-9

REFERENCES

(108) M.C.Vanwormhoudt,
' On Number Theoretic Fourier Transforms in Residue class rings'
Trans. Accoustics Speech and Signal Processing
IEEE vol ASSP-25 no.6 December 1977 pp 585-6

(109) M.C.Vanwormhoudt,
' Structural properties of complex residue rings applied to Number Theoretic Fourier Transforms'
Trans. Accoustics Speech and Signal Processing
IEEE vol ASSP-26 no.1 February 1978 pp 99-104

(110) E.Vegh L.M.Leibowitz,
' Fast Complex Convolution using Number Theoretic Transforms'
Naval Res. Lab. Washington D.C.
NRL report 7935 November 1975

(111) E.Vegh L.M.Leibowitz,
' Fast Complex Convolution using Number Theoretic Transforms'
Trans. Accoustics Speech and Signal Processing
IEEE ASSP vol.-24 No.4 August 1976 pp 343-4

(112) L.R.Welch R.A.Scholtz,
' Continued Fractions and Berlekamp's Algorithm'
Trans. Information Theory
vol IT-25 no.1 January 1979 pp 19-27

(113) S.Winograd,
' On computing the discrete Fourier Transform'
IBM Res. Report RC-6291 November 1976

(114) S.Winograd,
' On computing the discrete Fourier Transform'
Proc. Nat. Acad. Sci USA April 1976 pp 1005-6

(115) S.Winograd,
' Some Bilinear forms whose multiplicative complexity depends on the field of constants'
IBM Res. Report RC-5669 October 10 1975

(116) Y.Zalcstein,
' A note on fast convolution'
Trans. Computers
IEEE vol C-20 June 1971 pp 665-6

REFERENCES

(117) A.C.Davies Y.T.Fung ,
'Interfacing a hardware multiplier to a general purpose microprocessor'
Journal Microprocessors
vol.1 No.7 October 1977 pp 425-432

(118) C.H.Moore,
'FORTH: A new way to program a minicomputer'
Astron. Astrophysics. Suppl. No.15 1974 pp 497-511

(119) P.A.Blackburn P.Beadle,
'Private communication'
Plessey Electronic Systems Limited.

(120) S.C.Martin B.J.Stanier,
'Microprocessor Impelemetation of Number Theoretic Transforms'
IEE Journal Electronic Circuits and Systems
vol.3 No.1 January 1979 pp 21-26

(121) F.J.Macwilliams N.J.A.Sloane,
'The Theory of Error-Correcting Codes'
North Holland Publishing Company 1978

(122) J.H.McClellan C.M.Rader
'Number Theory in Digital Signal Processing'
Prentice Hall. Englewood Cliffs 1979.

(123) E.Dubois A.N.Venetsanopoulos,
'The generalized Discrete Fourier transform in rings of Algebraic Integers'
Trans. Accoustics Speech and Signal Processing
vol ASSP-28 no.2 April 1980 pp 169-175

REFERENCES

(124) H.J.Nussbaumer,
'Fast Polynomial transform algorithms for digital convolution'
Trans. Accoustics Speech and Signal Processing
vol ASSP-28 no.2 April 1980 pp 205-215

(125) W.K.Jenkins,
'Complex Residue number arithmetic for high-speed signal processing'
Electronics Letters vol.16 No.17 14th August 1980
pp 660-661

(126) A.Baraneika G.A.Julien,
'Residue Number System Implementation of Number Theoretic Transforms in Complex Residue Rings'
Trans. Accoustics Speech and Signal Processing
vol ASSP-28 1980 pp 205-215

Microprocessor implementation of number theoretic transforms

S.C.P. Martin and B.J. Stanier

Indexing terms: *Computerised signal processing, Transforms*

Abstract: Consideration is given to the suitability of microprocessor systems for the fast implementation of number theoretic transforms (n.t.t.s). Fast-multiply instructions available on some microprocessors, or the use of external multipliers, relax the basic constraints on the choice of a particular n.t.t. A search was made for suitable moduli which allow fast computation of n.t.t.s using Winograd's algorithm. The search was extended for other moduli which allow increased dynamic range when combined using the Chinese remainder theorem. Finally, a description is given of how modular arithmetic may efficiently be performed using microprocessors.

1 Introduction

A numerical procedure important in digital signal processing, the convolution operator, is defined by the relation

$$y_i = \sum_{j=0}^{N-1} h_{i-j} x_j \qquad i \in (0, N-1) \tag{1}$$

and is denoted by

$$y_i = h_i * x_i$$

Certain transforms (T) possess the cyclic-convolution property (c.c.p.) which may be stated as

$$T(y) = T(h) \times T(x) \tag{2}$$

where x denotes pointwise multiplication, or

$$y = T^{-1}\{T(h) \times T(x)\}$$

Hence an isomorphism exists between the convolution operator and the pointwise multiplication operator under such transforms as T.

Transforms with the d.f.t. structure possess the c.c.p. Let $X_k = T(x_i)$ and hence $x_i = T^{-1}(X_k)$, then

$$X_k \triangleq \sum_{j=0}^{N-1} x_j \alpha^{jk} \qquad k \in (0, N-1)$$

$$x_j \triangleq N^{-1} \sum_{k=0}^{N-1} X_k \alpha^{-jk} \qquad j \in (0, N-1) \tag{3}$$

where α is an element of order N. It has been shown[2] that the d.f.t. is the only such transform defined in the complex domain. However, many such transforms exist which are defined in finite rings of integers (Z_M). These transforms are collectively known as number theoretic transforms (n.t.t.s). Agarwal and Burrus[2] have shown that constraints limit practical choices of n.t.t.s and for clarity they are cited here:

(a) N must divide $0(M)$, where $0(M)$ is defined to be the greatest common divisor of the set of prime divisors (p_i) of M, i.e. $0(M) \triangleq$ g.c.d. (p_i)

(b) α must be an element of order N, i.e. $\alpha^N \equiv 1 \bmod M$ and $\alpha^r \neq 1 \bmod M \; \forall \; r \in (1, N-1)$

(c) N^{-1}, a multiplicative inverse of N, must exist in the ring Z_M.

(d) N should be well factored for fast algorithms to exist.

(e) To facilitate fast and simple arithmetic mod M, M must have a simple binary representation, and to facilitate fast multiplication by powers of α, α must also have a simple binary representation.

Considerable interest has been shown in the literature in at least two classes of numbers as choices for moduli. Using Fermat numbers,[1,4,5,17] which are numbers of the form $F_t = 2^{2^t} + 1$, it is possible to perform transforms requiring only bit-shifts and additions. However, such moduli do suffer from disadvantages: microprocessor systems handle most efficiently data in multiples of 8 bits (full bytes). By their definition Fermat number moduli may require an $(8k + 1)$th bit. During processing this may be supplied by a carry flag, but for data storage in memory this requirement can cause embarrassment. For F_5 and larger moduli, it is generally acceptable to ignore the extra bit since it has only a low probability of being required. Finally, the maximum transform length available for use with Fermat moduli is severely limited by the constraint upon a simple binary representation of α.

Mersenne numbers[6] (numbers of the form 2^{q-1}, q prime) have also been considered for moduli. However, corresponding sequence lengths are not factored and so do not have fast algorithms comparable to the f.f.t.

Recently, a new class of potentially fast and efficient Fourier transform and number-theoretic transforms are algorithms (w.f.t.a.), have been described.[7-11] Since the Fourier transform and number theoretic transforms are alike in structure, then any Fourier-transform algorithm may in principle be applied to n.t.t.s.

Winograd has shown how short-N transforms may be performed efficiently. He suggests a technique whereby short-N algorithms may be combined to provide an algorithm for a transform whose length is a product of the short-Ns, provided the short-Ns be relatively prime. Let

$$X_N = T_N x_N \tag{4}$$

where, as is well known[12]

$$T_N = P_2 T_{N_1} \otimes T_{N_2} P_1 \qquad N = N_1 N_2, (N_1, N_2) = 1 \tag{5}$$

Paper T290 E, first received 28th July and in revised form 1st November 1978

Mr. Martin and Dr. Stanier are with the Department of Applied Physics and Electronics, Science Laboratories, South Road, Durham DH1 3LE, England

0308-6984/79/010021 + 06 $01·50/0

and ⊗ denotes the Kronecker product of matrices and $\boldsymbol{P}_1, \boldsymbol{P}_2$ are permutation matrices.

Silverman[8] has described short-N algorithms for $N = 5, 7$, (3 and 9), (2, 4, 8 and 16), and using the nested nature of eqn. 4 various transform lengths are possible between 2 and 5040. Reference 8 shows how the permutation matrices $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ may be derived. In a microprocessor implementation the permutation sequences may conveniently be stored in read only memory.

2 Microprocessors and n.t.t.s

Integer arithmetic can be performed on microprocessor systems more easily than real arithmetic, and using such systems n.t.t.s are in principle easier to implement than the Fourier transform for convolution. Further, microprocessors are becoming available with fast-multiply instructions, and for those that do not have this facility, fast hardware multiplier chips are available.[13] These trends allow fast generalised multiplications, and make such operations competitive with repeated bit-shift and subtract in carry operations such as are required with Fermat number transforms. By waiving constraint (*e*) and allowing nonsimple moduli and αs, many more n.t.t.s become practicable for microprocessor implementation.

A search was made, as outlined in Reference 10, for suitable moduli that would satisfy constraints (*a*) to (*d*) and additionally would support general Winograd transform algorithms, allowing nonsimple Nth roots of unity. Since data is handled most efficiently in full bytes, this search was conducted from 2^{16} downwards.

It was found that the modulus $M = 65521$ satisfied all the constraints, and in particular it has two desirable properties:

(*a*) $0(65521) = 65520 = (5 \times 7 \times 9 \times 16) \times 13$. As can be seen, this modulus will support any Winograd transform length from 2 to 5040.

(*b*) Arithmetic would generally be complex because so little redundancy is incurred in the use of 16-bit arithmetic.

Since any specific-N transform algorithm would be computationally more efficient than a general-N algorithm, we derived various algorithms for specific transform lengths. However, it was found that a general-N program was an invaluable tool for such algorithm development, since the program derives the multiplication coefficients, the pointers into workspace arrays and the permutation sequences required for the specific-N algorithms. This design technique allows rapid development of the specific-N algorithms. Certain points are of interest concerning the general-N program and the subsequent specific-N algorithms.

(*a*) Since the ultimate aim is to derive algorithms for a microprocessor environment where memory workspace should be minimised, the suggestions for reducing memory requirement given in Reference 8 were heeded.

(*b*) Arithmetic would generally be complex because Fourier transform is defined in the complex domain. However, the algorithms described in Reference 8 involve only purely real or purely imaginary data, and so in the inner stages of their computation, certain savings are made by keeping flags to denote the data type. The concepts of real, imaginary and complex do not strictly apply in the number theoretic sense, and so it was not necessary for such flags to be kept.

(*c*) The points given in Reference 10 concerning the interpretation in the number theoretic sense of the trigonometrical expressions referred to in Reference 8, and the incorporation of the N^{-1} normalising factor into the multiplication coefficients for the inverse transform, were also heeded.

The general-N program was run on an IBM 370, and the results of convolutions performed by such n.t.t.s were compared with reference techniques (direct-integer convolution with short lengths, and a Fourier transform technique for long-convolution lengths). In all cases, the n.t.t. convolutions gave results in exact agreement with the direct-convolution technique. The Fourier-transform technique produces 'real' results subject to roundoff error, and within the limits of accuracy of such a technique, the results of the n.t.t. convolutions were also in exact agreement.

A transform of length 60 was written for an Intel 8080 microprocessor using the FORTH programming technique,[14] and this algorithm was employed in a real-time bandstop filtering application.

3 Extension to complex filtering

Vanwormhoudt[16] has shown that there exist two main classes of moduli. Since all moduli M that are useful for number theoretic transforms are odd, the two classes are those for which $M = 1 \bmod 4$ (type A) and those for which $M = 3 \bmod 4$ (type B). In Reference 16 it is shown that for type A moduli there exists an element j in the ring Z_M such that $j^2 \equiv -1 \bmod M$. This is an alternative to showing that there exists an element of order 4. It is also shown that no such element exists for type B moduli.

In References 15 and 17 it is shown that complex convolutions can be efficiently performed through two real convolutions. The ideas expressed in References 15 and 17 can be generalised for any type A modulus.

Let $a_i \triangleq \check{a}_i + j\hat{a}_i$

Where a = complex sequence
$\check{a}_i$ = real part of a_i
$\hat{a}_i$ = imaginary part of a_i
j = element such that $j^2 \equiv -1 \bmod M$

The two real convolutions required to compute y where $y_i = x_i * h_i$ are given in eqn. 6:

$$\left.\begin{aligned} \hat{y}_i + j\hat{y}_i &\equiv a_{xi} * a_{hi} \bmod M \\ \check{y}_i - j\hat{y}_i &\equiv b_{xi} * b_{hi} \bmod M \end{aligned}\right\} \qquad (6)$$

where the following are defined:

$$\left.\begin{aligned} a_{xi} &= \check{x}_i + j\hat{x}_i \bmod M; \quad & a_{hi} &= \check{h}_i + j\hat{h}_i \bmod M \\ b_{xi} &= \check{x}_i - j\hat{x}_i \bmod M; \quad & b_{hi} &= \check{h}_i - j\hat{h}_i \bmod M \end{aligned}\right\} \qquad (7)$$

In this paper much interest is shown in the modulus $M = 65521$ for which the element $j = 24297$ satisfies $j^2 = -1 \bmod M$.

4 Extension to other moduli

For a given modulus the output must be limited to avoid overflow; hence a compromise exists between the data amplitude and the filter impulse digitlisation.[2] In general, the digitisation will degrade the filter response, and so for a given choice of modulus there may be insufficient dynamic range for the filter design to achieve the limits set. In such cases, a larger modulus should be chosen;

Table 1: Table of dual moduli to pair with $M = 65521$ using c.r.t.

Convolution length	Dual modulus	Primary Winograd transform length	Multidimensional factor required
6	65497	6	–
10	65381	10	–
12	65497	12	–
14	65437	14	–
15	65101	15	–
18	65449	18	–
20	65381	20	–
21	65437	21	–
24	65497	24	–
28	65437	28	–
30	65101	30	–
35	65381	35	–
36	65449	36	–
40	64921	40	–
40	65497	8	5
42	65437	42	–
45	64621	45	–
45	65449	9	5
48	65281	48	–
56	65353	56	–
60	65101	60	–
63	65269	63	–
70	65381	70	–
72	65449	72	–
80	64081	80	–
80	65393	16	5
84	65437	84	–
90	64621	90	–
90	65449	18	5
105	65101	105	–
112	64849	112	–
112	65393	16	7
120	64921	120	–
120	65497	24	5
126	65269	126	–
140	65381	140	–
144	65089	144	–
168	65353	168	–
180	64621	180	–
180	65449	36	5
210	65101	210	–
240	64081	240	–
240	65281	48	5
252	65269	252	–
280	63841	280	–
280	65381	35	8
315	59221	315	–
315	65381	35	9
336	64849	336	–
336	65281	48	7
360	64081	360	–
360	65449	72	5
420	65101	420	–
504	64513	504	–
504	65449	72	7
560	63841	560	–
560	65281	16	5*7
630	59221	630	–
630	65381	70	9
720	64081	720	–
720	65089	144	5
840	63841	840	–
840	65353	168	5
1008	64513	1008	–
1008	65089	144	7
1260	59221	1260	–
1260	65381	140	9
1680	63841	1680	–
1680	65281	48	5*7
2520	55441	2520	–
2520	65449	72	5*7
5040	55441	5040	–
5040	65089	144	5*7

however, direct implementation of such a scheme would involve performing arithmetic with greater wordlength. This problem may be circumvented by the use of the Chinese remainder theorem (c.r.t.).[11] This theorem states that if an integer x is such that $x \equiv a_i \bmod m_i$ where a set of moduli m_i is relatively prime then

$$x \equiv a_1\left(b_1\frac{M}{m_1}\right) + a_2\left(b_2\frac{M}{m_2}\right), \ldots, + a_n\left(b_n\frac{M}{m_n}\right) \bmod M \qquad (8)$$

where

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

The b_i are defined such that

$$b_i\left(\frac{M}{m_i}\right) \equiv 1 \bmod m_i \qquad (9)$$

An interpretation of this theorem shows that if calculations are performed with respect to two or more relatively prime moduli (m_1 and m_2), then by using the c.r.t. the results may be determined $\bmod (m_1 m_2)$. This technique has great potential for utilising a parallel processing technique.

Let us consider the case for two moduli

$$x \equiv a_1 c_1 + a_2 c_2 \bmod M \qquad (10)$$

where

$$c_1 = b_1\frac{M}{m_1} \quad \text{and} \quad c_2 = b_2\frac{M}{m_2}$$

Let

$$x = 1 \quad \text{and so} \quad a_1 = 1 = a_2$$

therefore

$$1c_1 + 1c_2 \equiv 1 \bmod M$$

or

$$c_1 + c_2 \equiv 1 \bmod M \qquad (11)$$

This result will be discussed later.

A search was made for other moduli that would combine with 65521 over specific transform lengths. The search was conducted for various transform lengths by scanning from 2^{16} downwards for moduli, other than 65521, for which constraints (*a*) to (*d*) would be satisfied. These results are shown in Table 1.

It can be seen from the last two entries in Table 1 that the highest suitable modulus below 2^{16} that will directly support a transform length of 5040 is $M = 55441$. The choice of such a low modulus is undesirable, since a great loss occurs of the possible dynamic range of 2^{16}.

Agarwal and Cooley[3] described how the c.r.t. may also be employed to convert a 1-dimensional cyclic convolution to a multidimensional convolution which is cyclic in all dimensions. This may be applied to cases where a given long convolution length is a product of shorter mutually prime convolution lengths. They cite an example whereby a number-theoretic-transform technique can be used for convolution of length N, and by using the c.r.t. in this manner, convolutions of length $(Np_1p_2 \ldots p_j)$ may be computed provided $N, p_1, p_2, \ldots, p_j$ are mutually prime. Efficient algorithms are described in Reference 3 for convolutions of lengths 5 and 7 (2, 4, 8) and (3, 9). Using such a scheme, it is possible to perform n.t.t. convolutions of length 144, and using the multidimensional mapping technique it is possible to derive convolutions of length 144 x 5 x 7 (= 5040). Constraints upon the choice of modulus arise only from the n.t.t. length used, and not from the multidimensional factors employed. Therefore, modulus $M = 65089$ (this is the choice for a length 144 transform) may be used for convolutions of length 5040, even though this modulus does not support such a transform length directly, by taking n.t.t. convolutions of length 144 and using the c.r.t. mapping technique to compute the 5040 length convolutions.

This technique can be used to factor transform lengths with inefficient moduli to lengths with more efficient moduli. The other entries in the Table have been derived in a similar manner.

It is advantageous if the two moduli to be combined are used with the same transform lengths, since the same permutation sequences and essentially the same algorithms are used for both moduli, leading to economy in memory utilisation.

5. Microprocessors and modular arithmetic

The moduli previously described are optimally close to 2^{16} and therefore only a small duplicity arises if all 16-bit binary patterns are allowed on input and output with an arithmetic procedure. In all examples in this Section the choice of $M = 65521$ will be taken.

(*a*) Addition mod M: when two numbers are added together they may or may not generate an overflow. If there is no overflow then the result of the addition is returned.

If

$$\begin{aligned} x &= a + b = c + \text{carry} \\ &= c + 2^{16} \\ &\equiv c + (2^{16} - M) \\ &\equiv c + 15 \bmod 65521 \end{aligned} \qquad (12)$$

Therefore, if a carry is detected, 15 must be added into the partial sum. This may generate a further carry, but will not generate more than two carries. An example of suitable coding for such an operation is given for an Intel 8080 microprocessor.

```
PLUS  DAD   D
      RNC
      LXI   D     #15
      JMP   PLUS
```

This subroutine will add mod 65521 the two 16-bit numbers in register pairs (DE) and (HL) returning the answer in (HL).

(*b*) Subtraction mod M: for microprocessors with 16-bit subtraction instructions, the 16-bit addition instruction in the previous example may be directly replaced by such an instruction. However, few 8-bit microprocessors have such instructions, and so for the majority a byte-orientated subtraction should be used.

(*c*) Multiplication mod M: this operation can be classified into subdivisions:

(i) Multiplication of a 16-bit number (x) by a 16-bit fixed constant (k).

This can be considered to be a mapping from a 16-bit number to another 16-bit number. Since multiplication is a distributive operation, then the mapping may be achieved by treating separately the high and low bytes of x, and finally adding their respective outputs together. Fig. 1 shows how such a mapping may be achieved. The obvious design requires two $2^8 \times 16$-bit patterns which may be conveniently be read only memory (r.o.m.). If, however, the memory is restructured to be byte orientated, then a saving is obtained since part of the memory R_1 is duplicated in R_2. This minimised structure is shown in Fig. 2. R.O.M.s R are read once only and r.o.m. R' is read twice. The read operations are controlled by the microprocessor. A final modular add operation is required to bound the output within the 16-bit range.

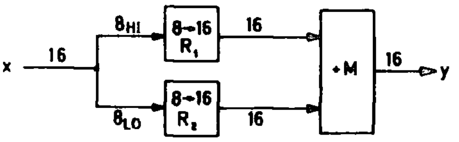


Fig. 1 *Multiplication by fixed constant*

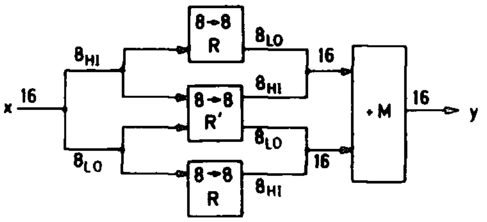


Fig. 2 *Memory minimisation for fixed constant multiplication*

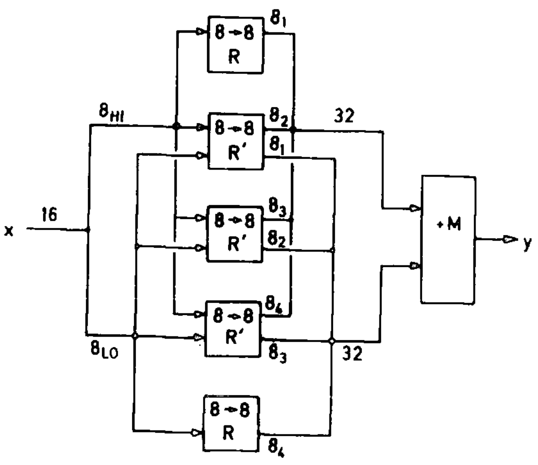


Fig. 3 *Multiplication of 16-bit number by 32-bit constant*

(ii) Multiplication of a 16-bit number by a 32-bit constant.

When combining results using the two moduli, the Chinese remainder theorem requires that calculations be performed of the form

$$x \equiv a_1 c_1 + a_2 c_2 \tag{13}$$

where a_1 and a_2 are results mod m_1 and mod m_2, respectively, and c_1 and c_2 are 32-bit constants derived using the c.r.t. Therefore, this class of operation is useful for deriving results mod $M (M = m_1 m_2)$ given the results mod m_1 and mod m_2. A memory reduction similar to that described previously can also be applied. A suitable structure for performing this operation is shown in Fig. 3. R.O.M.s R are read once and r.o.m.s R' are read twice. A final modular add is required to bound the output within the 32-bit range. This can be achieved using the same principle as has been described for the 16-bit modular add operation.

In general, two such operations would be required for multiplications by c_1 and c_2. However, if the result of eqn. 11 is recalled, then it will be seen that

$$\begin{aligned} 1 &\equiv c_1 + c_2 \bmod M \\ x &\equiv a_1 c_1 + a_2(1 - c_1) \\ &\equiv c_1(a_1 - a_2) + a_2 \bmod M \end{aligned} \tag{14}$$

from eqn. 8 it can be seen that

$$c_1 m_1 \equiv 0 \bmod M$$

and so

$$x \equiv c_1(km_1 + a_1 - a_2) + a_2 \bmod M \qquad (k \in N) \tag{15}$$

If k is chosen such that $(km_1 + a_1 - a_2)$ lies in the range zero to 2^{16}, then only one 16×32-bit multiplication is required to derive x-from eqn. 15. A flowchart to achieve this condition is shown in Fig. 4.

(iii) Multiplication of two 16-bit variables mod M.

The multiplication of two 16-bit numbers generates a 32-bit answer.

$$\begin{aligned} \text{Let } y &= ab = 2^{16} y_h + y_l \qquad 0 \leqslant a, b, y_h, y_l < 2^{16} \\ &\equiv (2^{16} - M) y_h + y_l \\ &\equiv 15 y_h + y_l \bmod 65521 \end{aligned} \tag{16}$$

Therefore, a general 32-bit intermediate answer may be partially reduced mod M by a multiplication of y_h by the fixed constant $(2^{16} - M)$. This can be achieved by the scheme described in (*c*), part (i).

(iv) Multiplication by 2^{-1}.

This is the analogous operation to division by 2.

$$\text{Let } y = 2^{-1} x$$

By applying a shift right to x we may determine from the carry flag if x was even or odd. If x was even then we have already determined y. If x was odd then y may be derived by adding in $(M + 1)/2$.

The procedures described cover the general classes of arithmetic required for convolution performed within the ring Z_M, where M is a product of 65521 and one of the moduli shown in Table 1.

(*d*) Table 2 summarises the memory requirements for the multiplication look-up tables. The unit of one page is used to denote a quantity of memory of 256 bytes. In the case of multiplications of two 16-bit variables, 3 pages are required for the Table to derive the term corresponding to the multiplication of the higher 16-bit intermediate answer with the fixed constant ($2^{16} - M$). The multiplication by j can be most efficiently performed by the technique described in Section 5, (*c*), part (i), and this requires an additional 3 pages.

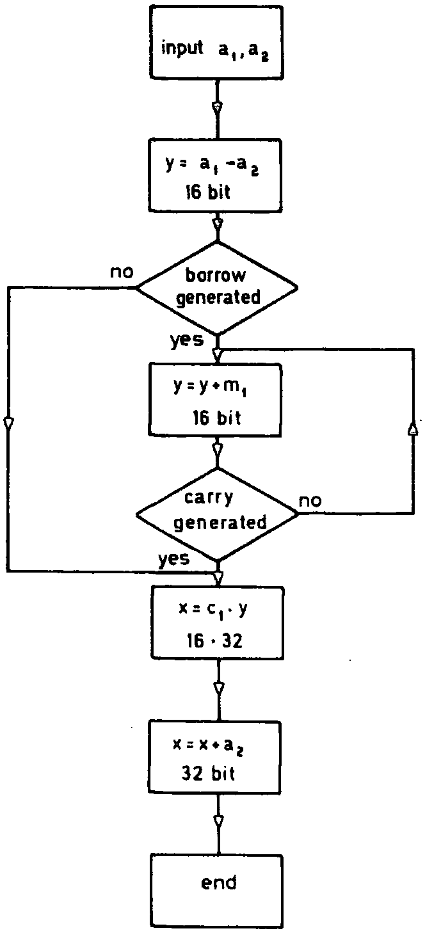


Fig. 4 *Flowchart to perform c.r.t. combination*

Table 2: Summary of memory requirements for multiplication look-up tables

		Memory required pages	
		For each modulus	For 2 moduli
Real convolutions	Generalised multiplications	3	6
	C.R.T. combination	–	5
	Total	3	11
Complex convolutions	Generalised multiplications	3	6
	Multiplication by j	3	6
	C.R.T. combination	–	5
	Total	6	17

6 Conclusions

With a view to utilising a microprocessor-based system for convolution, we have considered the constraints which govern the choice of a particular number theoretic transform (n.t.t.). We have found that recent hardware developments relax these basic constraints, allowing fast computation of much wider classes of n.t.t.s. In particular, the Winograd Fourier transform algorithm (w.f.t.a.) can be applied to n.t.t.s and the work in this paper has been concerned with implementing the w.f.t.a. using moduli just less than 2^{16}. The modulus 65521 was found to have very desirable properties from this aspect since it will support any Winograd transform length. We have also considered other moduli to combine with 65521 using the Chinese remainder theorem for applications which require a greater dynamic range.

It can be seen that microprocessor systems can be used effectively to provide the hardware for small-scale convolution requirements, and therefore such systems can be used wherever digital convolution is required. The range of possible applications is broad, covering areas from the computation of auto and cross correlation and power spectra and nonrecursive and recursive digital signal processing and obtaining the solution of difference equations.

7 References

1 AGARWAL, R.C., and BURRUS, C.S.: 'Fast convolution using Fermat number transforms with applications to digital filtering', *IEEE Trans.*, 1974, **ASSP–22**, pp. 87–99

2 AGARWAL, R.C., and BURRUS, C.S.: 'Number theoretic transforms to implement fast digital convolution', *Proc. IEEE*, 1975, **63**, pp. 550–560

3 AGARWAL, R.C., and COOLEY, J.C.: 'New algorithms for digital convolution', *IEEE Trans.*, 1977, **ASSP–25**, pp. 392–410

4 MELHUISH, P.: 'Fermat transform implementation by a mini-computer', *Electron. Lett.*, 1975, **11**, pp. 109–111

5 McCLELLAN, J.H.: 'Hardware realisation of a Fermat number transform', *IEEE Trans.*, 1976, **ASSP–24**, pp. 216–225

6 RADER, C.M.: 'Discrete convolution via Mersenne transforms', *ibid.*, 1972, **C–21**, pp. 1269–1273

7 WINOGRAD, S.: 'On computing the discrete Fourier transform', *Prod. Nat. Acad. Sci.*, 1976, **73**, pp. 1005–1006

8 SILVERMAN, H.F.: 'An introduction to programming the Winograd Fourier transform algorithm (WFTA)', *IEEE Trans.*, 1977, **ASSP–25**, pp. 152–165

9 SILVERMAN, H.F.: 'Corrections and an addendum to An introduction to the Winograd Fourier transform algorithm (WFTA)', *ibid.*, 1978, **ASSP-26**, pp. 268

10 BAILEY, D.: 'Winograds algorithm applied to number theoretic transforms', *Electron. Lett.*, 1977, **13**, pp. 548–549

11 SRIDHAR REDDY, N., and UMPATHI REDDY, V.: 'Implementation of Winograds algorithm in modular arithmetic for digital convolutions', *ibid.*, 1978, **14**, pp. 228–229

12 NICHOLSON, P.J., 'Algebraic theory of finite Fourier transforms', *J. Comput. & Syst. Sci.*, 1971, **5**, pp. 524–547

13 DAVIES, A.C., and FUNG, Y.T.: 'Interfacing a hardware multiplier to a general purpose microprocessor', *Microprocessors*, 1977, **1**, pp. 425–432

14 MOORE, C.H.: 'FORTH: A new way to program a mini-computer', *Astron. & Astrophys.*, 1974, **15**, pp. 497–511

15 SRIDHAR REDDY, N., and UMPATHI REDDY, V.: 'Complex convolutions using rectangular transforms', *Electron. Lett.*, 1978, **14**, pp. 458–459

16 VANWORMHOUDT, M.C.: 'Structural properties of complex residue rings applied to number theoretic Fourier transforms', *IEEE Trans.*, 1978, **ASSP–26** pp. 99–104

17 NUSSBAUMER, H.J.: 'Complex convolutions via Fermat number transforms', *IBM J. Res. & Dev.*, 1976, **20**, pp. 282–284