

FAST DIGITAL SIGNAL PROCESSING ALGORITHMS FROM
NUMBER THEORETIC TRANSFORMS

Wan-Chi SIU

AP(HK), M.Phil., C.Eng., M.I.E.R.E., M.I.E.E.E.

A Thesis submitted for
the Degree of Doctor of Philosophy
in the Faculty of Engineering,
University of London

September 1984

Department of Electrical Engineering
Imperial College of Science & Technology
University of London

ABSTRACT

The calculation of cyclic convolutions is one of the central operations in digital signal processing. The Number Theoretic Transform(NTT) has recently received a great deal of attention for the computation of cyclic convolutions, because it is possible to obtain a transform that involves no multiplications and no roundoff errors in the calculation. However, all NTTs are limited by wordlength and sequence length constraints. For some of the very promising NTTs, there is no efficient algorithm for their realisation, and so this restricts the applications of NTTs as a whole. It is the purpose of the present research to investigate the use of NTTs from the point of view of designing high-speed, computationally efficient algorithms.

In our study, Number Theoretic Transforms are examined and expressed in a systematic way, using a polynomial formulation, that facilitates the computation of the convolution sum of long sequences each of length-N in a two-dimensional (or multi-dimensional) form. This new formulation shows very clearly the structure of the equations for computing cyclic convolutions arranged in multi-dimensional form.

Mersenne Number Transforms have the advantage of having a very efficient arithmetic. However there is no FFT-type algorithm for the computation, and for this reason little attention has been given to them. We suggest two ways of implementing MNTs. In our first design, we convert the MNT into a cyclic convolution form and propose a simple hardware structure to carry out the fast implementation. In

the second approach we arrange the MNT into a recursive filter form and propose a hardware structure for both serial and parallel implementations. The former structure requires the simplest hardware whereas the latter structure can be used to relax the wordlength and sequence length constraints.

High-speed and low cost ROMs are now available. We suggest a new type of Number Theoretic Transform for the computation of convolutions using ROMs. This new formulation only requires a small ROM size and the control of data flow is extremely simple.

The computation of a q -point DFT can always be rearranged in a length- $(q-1)$ cyclic convolution form, where q is a prime number. By noting the symmetry property of the NTT, we show that the transformed results of the W^n terms using NTT are alternately zero, thereby reducing the number of multiplications to less than one [actually $1-\frac{1}{q}$] multiplication per point for the DFT of a real sequence length- q . This is the smallest number of multiplications that has ever reported for the computation of DFTs.

Further to the basic formulations in the previous paragraph, some NTT modules are derived and suggested to form the basic sequence lengths to compute long DFTs. This leads to the efficient design of algorithms for the computation of DFTs using the Fermat Number Transform and the Mersenne Number Transform.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my supervisor Dr. A.G. Constantinides for his helpful advice and encouragement throughout every stage of the work reported in this thesis. His enthusiasm for the subject and criticism have greatly influenced the course of this research.

The financial support^{*} of the Hong Kong Polytechnic for my research is gratefully acknowledged. I would also like to thank my colleagues of the Department of Electronic Engineering of the Hong Kong Polytechnic, especially Dr. J.S.L. Wong (Head of Department) and Dr. B. Jefferies for their support and encouragement. Thanks should also be given to Prof. C.F. Chen of the Department of Electronics of the Chinese University of Hong Kong for his encouragement and his earnest interest at all times which has been a great stimulus to me.

I am also indebted to all members of the Signal Processing Section of the Department of Electrical Engineering of the Imperial College of Science and Technology, in particular to John Morris for his forbearance and suggestions while proof-reading the manuscript.

^{*}The work reported in this thesis was done while the author was on leave from the Department of Electronic Engineering of the Hong Kong Polytechnic, Hung Hom, Kowloon, Hong Kong.

CONTENTS

	PAGE
TITLE PAGE	1
ABSTRACT	2
ACKNOWLEDGEMENTS	4
CONTENTS	5
STATEMENT OF ORIGINALITY	9
CHAPTER ONE INTRODUCTION	13
1.1 Cyclic Convolutions and Discrete Fourier Transforms	13
1.2 Recent Development	14
1.3 Number Theoretic Transforms	16
1.4 Thesis Structure	20
CHAPTER TWO REVIEW OF BASIC THEORY	23
2.1 Circular Convolution Property	23
2.2 Transforms in a Ring of Integers Modulo an Integer, M	33
2.3 Mersenne Number and Fermat Number Transforms	38
2.4 Extension of the Dynamic Range	53
2.5 Pseudo-Number Theoretic Transform	56
2.6 Complex Number Theoretic Transform	59
2.7 Appendices	69
CHAPTER THREE CYCLIC CONVOLUTION OF LONG SEQUENCES USING NUMBER THEORETIC TRANSFORM	75
3.1 Circular Shift Theorem	76
3.2 One-Dimensional Circular Convolution	78

3.3 Two-Dimensional Circular Convolution	85
3.4 Nested Number Theoretic Transform	95
3.5 Conclusions	97
3.6 Appendices	98
CHAPTER FOUR APPROACH TO THE HARDWARE IMPLEMENTATION OF DIGITAL SIGNAL PROCESSORS USING MERSENNE NUMBER TRANSFORMS	
	109
4.1 Mersenne Number Transform and Cyclic Convolution	110
4.2 Hardware Description	115
4.3 Speed Optimised Convolver using Mersenne Number Transform	119
4.4 Long Convolution from Short Convolution	120
4.5 Extension to Fermat Number Transforms and other Number Theoretic Transforms	123
4.6 Conclusion	127
4.7 Figures	129
CHAPTER FIVE FAST HARDWARE REALISATION OF MERSENNE NUMBER TRANSFORMS FOR DIGITAL CONVOLUTIONS	
	133
5.1 Hardware Design for Mersenne Number Transforms	134
5.2 Sequence Lengths of the Transforms Modulo a Mersenne Number	142
5.3 Long Sequence Lengths	148
5.4 Conclusion	155
5.5 Tables	157
5.6 Figures	163
CHAPTER SIX EFFICIENT HARDWARE REALISATION OF NUMBER THEORETIC TRANSFORM CONVOLVERS USING ROMs	
	169
6.1 Prime Sequence Length Number Theoretic Transforms	170
6.2 Practical Number Theoretic Transforms	174
6.3 Conclusion	184
6.4 Tables	186

6.5 Figure	188
CHAPTER SEVEN VERY FAST DISCRETE FOURIER TRANSFORM USING NUMBER THEORETIC TRANSFORM	189
7.1 Theory	190
7.2 An Example	197
CHAPTER EIGHT COMPUTATION OF DISCRETE FOURIER TRANSFORM USING NUMBER THEORETIC TRANSFORM	200
8.1 Basic Fermat Number Transform Modules	200
8.2 Prime Factor Implementation	210
8.3 Hybrid Implementation	219
8.4 Conclusion	220
8.5 Tables	222
8.6 Figures	226
CHAPTER NINE FAST MERSENNE NUMBER TRANSFORMS FOR THE COMPUTATION OF DISCRETE FOURIER TRANSFORM	229
9.1 Mersenne Number Transforms and Discrete Fourier Transforms	230
9.2 Long Sequence Lengths	238
9.3 Discrete Fourier Transforms using Pseudo-Mersenne Number Transforms	245
9.4 Conclusion	247
9.5 Appendix	248
9.6 Tables	250
9.7 Figure	254
CHAPTER TEN GENERAL CONCLUSIONS AND SUGGESTIONS FOR FUTURE RESEARCH	255
REFERENCES	263

LIST OF PUBLICATIONS OF THE AUTHOR ON WHICH THIS THESIS IS BASED	275
LIST OF PRESENTATIONS CONDUCTED BY THE AUTHOR WITHIN THIS RESEARCH PERIOD	277

STATEMENT OF ORIGINALITY

The following contributions reported in this thesis are claimed to be original:

(i) The introduction of the use of polynomial formulations for the computation of cyclic convolutions of long sequences using Number Theoretic Transforms[chapter 3]

These formulations lead us to be able to demonstrate:

(a) very clearly the structures of the computations of long cyclic convolutions using multi-dimensional techniques for Number Theoretic Transforms[section 3.2],

(b) a new and systematic method for the formulation of the equations for the computation using multi-dimensional techniques for one and two-dimensional cyclic convolutions[sections 3.2 and 3.3] and

(c) the suggestion of using nested NTTs for the computation of cyclic convolutions[section 3.4].

(ii) The formulation of the conversion of the Mersenne Number Transforms into cyclic convolutions suitable for high-speed implementations[chapter 4]

This formulation leads to:

(a) a new and simple hardware design for the serial implementation of the MNT [section 4.2],

- (b) a new and simple hardware design for the parallel implementation of the MNT[section 4.2] and
- (c) an extension of these design to other NTTs[section 4.3].

(iii) The suggestion of the arrangement of the Mersenne Number Transform into recursive filter form[chapter 5]

This suggestion leads to:

- (a) a new and efficient serial hardware structure to implement the MNT[section 5.1],
- (b) a new and efficient parallel hardware structure to implement the MNT[section 5.1],
- (c) a method to compute MNTs with transform lengths not being equal to P or $2P$ [section 5.2] and
- (d) a solution to the problem of the word-length and the sequence length constraints of the MNT[5.3].

(iv) The development and design of a new class of Number Theoretic Transforms for cyclic convolutions using ROM arrays[chapter 6]

This study includes:

- (a) a generalisation of point (iii)(a) to any Number Theoretic Transform with prime sequence length[section 6.1] and
- (b) a suggestion for the design of new NTTs without word-length and sequence length constraints[6.2].

(v) The development of the basic formulations for the computation of

Discrete Fourier Transforms using Number Theoretic
Transforms[chapter 7]

This development gives:

- (a) the conditions and basic equations of the computation of the Discrete Fourier Transforms using Number Theoretic Transforms[section 7.1],
- (b) the properties of the transformed values of W 's[section 7.1] and
- (c) the least number of multiplications that has ever been reported for the computation of DFTs[section 7.1].

(vi) The study on the computation of Discrete Fourier Transforms using Fermat Number Transforms[chapter 8]

This includes:

- (a) the conjugate property of $\{W_n\}$ for the two-dimensional formulation[section 8.1].
- (b) the suggestion of 8 basic FNT modules for computing long DFTs[section 8.1] and
- (c) the discussion on the computation of long DFTs with the minimum number of operations[sections 8.2 and 8.3].

(vii) The study on the computation of Discrete Fourier Transform using Mersenne Number Transform[chapter 9]

This includes:

- (a) the suggestion of using Mersenne-composite Number

Transforms for the computation of DFTs[section 9.1],

(b) the discussion on the computation of long DFTs using MNTs with the number of operations less than the number of operations reported before[section 9.2] and

(c) the suggestion of using the pseudo-Mersenne Number Transform for the computation of Discrete Fourier Transforms[section 9.3].

CHAPTER 1 INTRODUCTION

1.1 Cyclic Convolutions and Discrete Fourier Transforms

In the past few years digital signal processing(D.S.P.) techniques[1-10] have been extensively studied because of their importance in a wide range of applications such as image processing[11-12], speech processing[13], radar processing[10], biomedical engineering[3] and many others. The field of digital signal processing broadly speaking consists of two major subdivisions: (i) digital filtering and (ii) spectrum analysis using the Discrete Fourier Transform(DFT).

Every linear digital filter can be described by the linear convolution sum. For the implementation of these filters[14-15], transform techniques have very often been used to carry out the computation. Most fast convolution algorithms, such as those based on Fast Fourier Transform(FFT)[16-17], apply only to periodic functions and compute cyclic convolutions. Hence, in order to take advantage of various fast cyclic convolution algorithms to speed-up the calculation of digital filtering processes methods like the overlap-add[17] and overlap-save[17] can be employed to effect the computations. Furthermore, the Discrete Fourier Transform (DFT) can also be converted into a cyclic convolution form when the transform length, N is a prime number[18]. It is evident, therefore, that the cyclic convolution is a very basic and essential operation in digital signal processing.

It is almost self-evident to state that if an efficient method for computing the DFT is developed, then there is a high likelihood that the same technique may be used to compute the cyclic convolutions and vice versa. There is therefore a parallel development between the computation of cyclic convolutions and Discrete Fourier Transforms.

One of the most important events in fast algorithm development was the introduction of the Fast Fourier Transform(FFT) by Cooley and Tukey[16-17] in 1965. This algorithm allows a tremendous saving in the calculation of the Discrete Fourier Transform. It reduces the number of real multiplications from $4N^2$, to $2N\log_2 N$, while the number of real additions is reduced from $2N^2$ to $3N\log_2 N$ for N -point complex DFTs. The same order of reduction is also achieved for the computation of cyclic convolutions by the FFT algorithm. The major disadvantages of the FFT algorithm are that (i) it still requires quite a large number of multiplications and (ii) the number of real multiplications per point is almost the same for both real and complex input data. This last point makes the FFT very inefficient for calculating the DFT of real input data. The FFT is, however, still the most popular algorithm for the computations of DFTs and cyclic convolutions because of its simplicity of programming and suitability for general computers.

1.2 Recent Development[19-22]

In 1975, Winograd[23] showed that the minimum number of multiplications required to compute the cyclic convolution of two

length- N sequences is $2N-K$, where K is the number of divisors of N including 1 and N . Agarwal and Cooley[24] used the short convolution algorithms developed by Winograd[25], and Agarwal and Burrus[26], for high-speed computation of long convolutions. This method uses the Chinese Remainder Theorem(CRT) [27-28] for mapping one-dimensional sequences into multi-dimensional sequences and is found to be better than the FFT methods for sequence lengths up to $N=420$.

With the short cyclic convolution algorithms as a basis, it is clear that efficient, short DFT algorithms may also be constructed. Winograd[25,29-30] made use of Rader's Theorem[18] on DFT of sequences with lengths equal to prime numbers to construct algorithms for the computation of Discrete Fourier Transforms. Compared with conventional FFT methods, the Winograd Fourier Transform algorithms(WFTA) require between one-half to one-third of the number of multiplications needed for conventional FFTs with a slightly increased number of additions per point. Kolba and Parks[31] modified some of Winograd's short DFT algorithms and made a detailed analysis of using prime factor algorithm(PFA) for the computation of DFTs. This method requires a relatively larger number of multiplications compared to the WFTA. However, it is easier to write a general prime factor DFT program and the computation can be done using less memory than is required for Winograd Fourier Transform Algorithm[31-32]. Both the WFTA and PFA make use of the original idea, described by Good[33-34], of breaking up a one-dimensional transform into a multi-dimensional transform with prime factors.

In 1977, Nussbaumer[35] first suggested the use of Polynomial Transforms(PT) over the field of polynomials to compute two-dimensional convolutions. Polynomial transforms can be computed without multiplications using ordinary arithmetic, and they convert multi-dimensional convolutions into one-dimensional convolutions and polynomial products. The number of multiplications per point lies in the range 2 to 4 for the computation of 12×12 to 120×120 two-dimensional cyclic convolutions[20]. This approach also led to the development of fast algorithms[35-37] for computing DFTs using polynomial transforms. The method requires two to four real multiplications per point for complex input data. However, for the computation of one-dimensional DFTs, one has to convert the one-dimensional DFT into a two-dimensional form and arrange the major part of the calculation as a two-dimensional convolution suitable for the Polynomial Transform. In this case the saving in the number of multiplications is less than in the two-dimensional case.

It is also interesting to point out that, for the computation of real input data both the Winograd Fourier Transform and the Polynomial Transform require approximately half the number of operations as are needed in the case of complex data.

1.3 Number Theoretic Transforms[38-41]

Both the Winograd Fourier Transform Algorithm and the Polynomial Transform make use of number theory[27-28] to develop the algorithms. An alternative approach, which also relies heavily on number theory, is to use Number Theoretic Transforms(NTT). The Number

Theoretic Transform is defined on finite fields or rings of real integers[38-43]. The transform involves no complex-number multiplications and has properties similar to those of the Discrete Fourier Transform. Thus it can be used as a method for fast and error free calculation of cyclic convolutions applicable for example to the implementation of digital signal processors.

Of the various versions of the NTT proposed, the Fermat Number Transform(FNT) which has been thoroughly investigated by Agarwal and Burrus[40], and the Mersenne Number Transform(MNT) which was introduced by Rader[39] are the most promising cases. Since the transform length N of the Fermat Number Transform must be a power of two, a fast implementation for it similar to the Fast Fourier Transform with radix two is possible. For the computation of FNT, arithmetic operations must be carried out in modulo (2^b+1) . Usually α , the root of unity of order N , is taken to be two, or a power of two. However within the arithmetic operations modulo (2^b+1) multiplication of a number in a register by α^n does not simply involve shifting the contents of the register by n bits, it is also necessary to subtract n overflow bits and to take other precautions. In order to simplify the computation, Agarwal and Burrus[39] made their implementation of FNT using a b -bit representation of integers which can introduce some error into the result. McClellan[44] defined a new code representation for the integers modulo a Fermat number, F_t , using a normal binary weighting with digits ± 1 . McClellan's approach was simplified by Leibowitz[45] who proposed the diminished-one representation to represent the number system used in FNT. Both

McClellan's and Leibowitz's methods effectively convert the arithmetic to be similar to, and of the same complexity as, ordinary ones-complement arithmetic.

Since the transform lengths of the Mersenne Number Transforms are not highly composite, there is no FFT-type algorithm for their computation, and for this reason little attention has been given to them. Mersenne Number Transforms have the advantage of being able to use very efficient arithmetic (which could be in ones-complement form). For example, the required multiplications by powers of two are performed by simple bit rotations. One possible method introduced by Reed and Truong[46-48] of retaining the arithmetic modulo of a Mersenne Number and of utilising the FFT-type algorithm is to define the Number Theoretic Transform in a Galois field with roots of unity being Gaussian integers. Unfortunately, not all the roots of unity of these complex MNTs are simple and so some general multiplications[46-49] are required for the computation of the transform. Nussbaumer[50] suggests an alternative solution by introducing the pseudo-Mersenne Transform. Two-step modulo arithmetic (on two moduli), in addition to larger word-lengths, have to be used to carry out the computation. Furthermore the resultant transform is not always highly composite.

All Number Theoretic Transforms are restricted by wordlength and sequence length constraints[38-43,51]. Nussbaumer[50] increases the sequence length of the MNT from $2P$ to $8P$ by observing that the complex number $(1+i)$ is an element of order $8P$ in $GF(q^2)$. This

complex MNT produces a four-fold increase in sequence lengths and is more suitable for complex convolutions, rather than for the convolution of real data. Reed and Truong[58] have also proposed to employ Winograd's algorithm[23,25] to compute convolutions using Mersenne Number Transforms with 2 or other real integers as the root of unity. They restricted their discussion to moduli of Mersenne primes only and general multiplication operations have to be employed for the computation of the transforms for long sequence lengths.

Another approach, fundamentally different from the previous methods, proposed for the solution of the sequence length problem has been given by Dubois and Venetsanopoulos[52] and Duhamel and Holmann[53-55]. They choose moduli other than Fermat numbers or Mersenne numbers for the Number Theoretic Transform. However, the arithmetic operations needed in those cases are more complicated than those used in MNTs and FNTs. Thomas, Larson and Keller[56] have also defined a generalised Number Theoretic Transform which allows for an independent choice of transform length and modulus base. However, this method sacrifices the very important circular convolution property of the transform. Balla and Antoniou[58] have proposed to use 3 as the root of unity for Number Theoretic Transforms. This choice gives sequence lengths sufficiently large for practical applications but this algorithm requires a relatively complicated ternary arithmetic for its implementation.

Besides the sequence length and wordlength constraints, the Number Theoretic Transform also requires efficient circuit structures

and/or arithmetic for its implementation. Coding is one solution to this problem. Another solution originally suggested by Pollard[59] and with hardwares implemented by Jullien, Miller[60-62] and Nagpal[62] is to use an array of ROMs(Read-Only-Memory) to do the arithmetic. This method uses a relatively large ROM size[60-63], but it can be used to implement very high speed digital signal processors.

1.4 Thesis Structure

The aim of the research presented in this thesis is to investigate efficient implementation of Number Theoretic Transforms for computationally efficient Digital Signal Processors. This includes solutions to the problems mentioned in the previous sections. The major part of this thesis focuses on our new formulations of NTT equations to relax the sequence length and wordlength constraints; and efficient algorithms for the computation of Discrete Fourier Transforms using NTTs. High-speed hardware structures are also proposed for use in Digital Signal Processors.

Chapter 2 of this thesis is devoted to a review of the theoretical development of Number Theoretic Transforms. This includes the conditions for the existence of the Number Theoretic Transform and discussions on the constraints and advantages in using them.

In chapter 3, Number Theoretic Transforms are examined and expressed in a systematic way, using a polynomial formulation, that facilitates the computation of the convolution sum of long sequences each of length-N in two-dimensional (or multi-dimensional) form. This

new formulation shows very clearly the structure of the computations of cyclic convolutions (one- or multi-dimensional) using multi-dimensional techniques when (i) the factors of N are not mutually prime and (ii) the factors of N are mutually prime.

To overcome the lack of FFT-types of algorithm for the computation of Mersenne Number Transform, we suggest two efficient ways to implement MNTs. In chapter 4, we convert the MNT into a cyclic convolution form and propose a simple hardware structure to carry out the fast implementation. In chapter 5, we arrange the MNT into a recursive filter form and propose a hardware structure for both serial and parallel implementations. The former structure requires the simplest hardware whereas the latter structure can be used to relax the wordlength and sequence length constraints.

High-speed and low cost ROMs are now available. Chapter 6 give an account of our new Number Theoretic Transforms suggested for the computation of convolutions using ROMs. This new formulation only requires a small ROM size and the control of data flow is extremely simple.

In chapter 7, we suggest the use of the Number Theoretic Transforms to compute the Discrete Fourier Transform. By noting the symmetry of the NTT, we show that the number of multiplications per point for the DFT for a real sequence length of P is less than one (actually $1 - \frac{1}{P}$). This is the least number of multiplications that has ever been reported for the computation of DFTs.

Further to the basic formulation in chapter 7, chapter 8 shows that in most cases by using Fermat Number Transforms to compute DFTs is less than that the number of operations in the Winograd Fourier Transform Algorithm and the Polynomial Transform. Chapter 9 gives the results of our study using the Mersenne-prime and Mersenne-composite Number Transforms to compute DFTs. The number of multiplications per point remains one and the control of data flow in this new approach is very simple.

In the final chapter, chapter 10, we discuss the conclusions of the present research and give some suggestions for further research.

CHAPTER 2 REVIEW OF BASIC THEORY

In this chapter, we start our discussion with the development of the general structure of transforms with the convolution property and then present the Number Theoretic Transforms modulo an integer as a practical scheme for calculating cyclic convolutions. It is possible to obtain a Number Theoretic Transform that involves no multiplications and no roundoff errors in the calculation. The ease of implementation of the transform often depends on the choice of modulus(M), root of unity(α) and the transform length (N). This chapter also includes the conditions for the existence of the Number Theoretic Transform and the discussions on the constraints and advantages of using NTTs for Digital Signal Processing.

2.1 Circular Convolution Property

In this section we discuss the essential requirements for transforms to have the circular convolution property, as the Discrete Fourier Transform has. The circular convolution property of a transform means that the transform of the cyclic convolution of two sequences is equal to the product of their individual transforms. Consider two sequences

$$\tilde{x} = \{x(0), x(1), \dots, x(N-1)\}$$

$$\tilde{h} = \{h(0), h(1), \dots, h(N-1)\}$$

where N = length of the sequences

For the cyclic convolution of these two sequences, we can write

$$\tilde{y} = \tilde{x} \circledast \tilde{h} \text{ and}$$

$$y(m) = \sum_{n=0}^{N-1} x(n)h(m-n) \quad (2.1)$$

where $m=0,1,\dots,N-1$

The symbol $\circledast$ means circular convolution. This definition assumes the sequences are periodically extended with period N .

Consider an $N \times N$ non-singular matrix $\bar{T}$ whose elements are, $t_{k,n}$ where $k,n=0,1,\dots,N-1$ respectively. Let

$$\bar{x} = \begin{bmatrix} x(0) \\ x(1) \\ \dots \\ \dots \\ x(N-1) \end{bmatrix} \quad \bar{h} = \begin{bmatrix} h(0) \\ h(1) \\ \dots \\ \dots \\ h(N-1) \end{bmatrix} \quad \text{and} \quad \bar{y} = \begin{bmatrix} y(0) \\ y(1) \\ \dots \\ \dots \\ y(N-1) \end{bmatrix}$$

Use matrix $\bar{T}$ to transform $\bar{x}$ and $\bar{h}$ into a new domain and let capital letters denote the transformed sequences, we obtain

$$\begin{aligned} \bar{X} &= \bar{T}\bar{x} \\ \bar{H} &= \bar{T}\bar{h} \\ \bar{Y} &= \bar{T}\bar{y} \end{aligned} \quad (2.2)$$

$\bar{T}$ is now defined to have the property that

$$\bar{Y} = \bar{X} \chi \bar{H} \quad (2.3)$$

where X = term by term multiplication

The individual terms in eqn. 2.3 can be written as

$$Y(k) = X(k)H(k)$$

$$\text{where } X(k) = \sum_{n=0}^{N-1} t_{k,n} x(n) \text{ and}$$

$$H(k) = \sum_{\ell=0}^{N-1} t_{k,\ell} h(\ell)$$

$$\text{for } k=0,1,\dots,N-1$$

Hence

$$\begin{aligned} Y(k) &= \sum_{n=0}^{N-1} t_{k,n} x(n) \sum_{\ell=0}^{N-1} t_{k,\ell} h(\ell) \\ &= \sum_{n=0}^{N-1} \sum_{\ell=0}^{N-1} x(n) h(\ell) t_{k,n} t_{k,\ell} \end{aligned} \quad (2.4)$$

However, $Y(k)$ can also be expressed as

$$Y(k) = \sum_{m=0}^{N-1} t_{k,m} y(m)$$

By eqn. 2.1,

$$Y(k) = \sum_{m=0}^{N-1} t_{k,m} \sum_{n=0}^{N-1} x(n) h(m-n)$$

Let $\ell = m-n$, where $\ell=0,1,\dots,N-1$, we get

$$Y(k) = \sum_{n=0}^{N-1} \sum_{\ell=0}^{N-1} x(n)h(\ell)t_{k,\ell+n} \quad (2.5)$$

Eqn. 2.4 must be equivalent to eqn. 2.5, therefore

$$t_{k,\ell+n} = t_{k,\ell}t_{k,n} \quad (2.6)$$

where $k, \ell, n=0,1,\dots,N-1$ respectively

The properties of the t 's give the essential features of the transforms having the convolution property, hence it is necessary to elaborate eqn. 2.6 in more detail.

(i) It is clear from eqn. 2.6 that

$$t_{k,1+1} = t_{k,1}t_{k,1}$$

and
$$t_{k,2} = t_{k,1}^2$$

$$t_{k,3} = t_{k,2+1} = t_{k,1}^3$$

Therefore, in general

$$t_{k,n} = t_{k,1}^n \quad (2.7)$$

(ii) From eqn. 2.6, it is also clear that

$$t_{k,\ell+0} = t_{k,\ell}t_{k,0}$$

Hence

$$t_{k,0} = 1 \quad (2.8)$$

(iii) For the cyclic property of $\ell+n$,

$$t_{k,i+N} = t_{k,i}$$

From eqn. 2.6 again

$$\begin{aligned} t_{k,i+N} &= t_{k,i} t_{k,N} \\ &= t_{k,i} t_{k,1}^N \end{aligned}$$

Hence

$$t_{k,1}^N = 1 \quad (2.9)$$

Note that

$$t_{k,nN} = t_{k,\underbrace{n+n+\dots}_N} = t_{k,n}^N$$

Interchange n and N ,

$$\begin{aligned} t_{k,nN} \text{ also } &= t_{k,N}^n \\ &= t_{k,0}^n \quad (\text{cyclic property}) \\ &= 1 \end{aligned}$$

Therefore

$$t_{k,n}^N = 1 \quad (2.10)$$

This shows that $t_{k,n}$'s are N th roots of unity.

(iv) For $\bar{T}$ to be nonsingular, hence $\bar{T}^{-1}$ exists, $t_{k,1}$'s should be

distinct.

(v) $\bar{T}$ can now be written in the following form

$$\bar{T} = \begin{bmatrix} 1 & t_{0,1} & t_{0,1}^2 & \dots & t_{0,1}^{N-1} \\ 1 & t_{1,1} & t_{1,1}^2 & \dots & t_{1,1}^{N-1} \\ \cdot & \cdot & \cdot & \dots & \cdot \\ 1 & t_{N-1,1} & t_{N-1,1}^2 & \dots & t_{N-1,1}^{N-1} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & \alpha_0 & \alpha_0^2 & \dots & \alpha_0^{N-1} \\ 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{N-1} \\ \cdot & \cdot & \cdot & \dots & \cdot \\ 1 & \alpha_{N-1} & \alpha_{N-1}^2 & \dots & \alpha_{N-1}^{N-1} \end{bmatrix} \quad (2.11)$$

where α 's are N^{th} root of unity.

To make a further development of point (v), eqn. 2.9 shows that

$$t_{k,1}^N = 1$$

That is

$$t_{1,1}^N = 1$$

$$t_{2,1}^N = 1$$

$$\cdot \cdot \cdot$$

$$t_{N-1,1}^N = 1$$

It is essential that $t_{k,1}$'s are distinct roots of unity. Note that in a certain field, for $t_{1,1}^N = 1$, there are only N distinct N^{th} roots of unity, where N is the least positive integer such that $t_{1,1}^N = 1$.

For the sake of simplicity and without loss of generality, let

$$t_{k,1} = t_{1,1}^k \quad (2.12)$$

Let also $\alpha = t_{1,1}$ and combine eqns. 2.12 and 2.7. We find

$$t_{k,n} = t_{1,1}^{kn} = \alpha^{kn} \quad (2.13)$$

In order to transform the result back to the original domain, we have to find $\bar{T}^{-1}$. Let $t_{k,n}^{-1}$ be the elements of $\bar{T}^{-1}$ and have the following form

$$t_{k,n}^{-1} = N^{-1} \alpha^{-kn}$$

where $k=0,1,\dots,N-1$

To prove the validity of this assumption:

$$\text{Since } X(k) = \sum_{n=0}^{N-1} t_{k,n} x(n)$$

$$= \sum_{n=0}^{N-1} \alpha^{kn} x(n)$$

Now
$$x(n) = \sum_{\ell=0}^{N-1} N^{-1} \alpha^{-\ell n} X(\ell)$$

hence
$$X(k) = \sum_{n=0}^{N-1} \alpha^{kn} \sum_{\ell=0}^{N-1} N^{-1} \alpha^{-\ell n} X(\ell)$$

$$X(k) = \sum_{\ell=0}^{N-1} X(\ell) \left(N^{-1} \sum_{n=0}^{N-1} \alpha^{(k-\ell)n} \right) \quad (2.14)$$

Now consider the term $N^{-1} \sum_{n=0}^{N-1} \alpha^{(k-\ell)n}$. Due to the circular property, the value of $(k-\ell)$ should modulo N .

(i) If $(k-\ell) = 0 \pmod{N}$, then

$$\alpha^{k-\ell} = 1$$

Therefore $N^{-1} \sum_{n=0}^{N-1} \alpha^{(k-\ell)n} = 1$ for $(k-\ell)=0$

(ii) If $(k-\ell) \neq 0 \pmod{N}$, then

$$\alpha^{k-\ell} \neq 1$$

That is

$$\alpha^{k-\ell} - 1 \neq 0 \quad (2.15)$$

Multiply the expression $N^{-1} \sum_{n=0}^{N-1} \alpha^{(k-\ell)n}$ by $(\alpha^{k-\ell} - 1)$, hence we get,

$$\begin{aligned}
 & N^{-1}(\alpha^{k-\ell}-1) \sum_{n=0}^{N-1} \alpha^{(k-\ell)n} \\
 & = N^{-1} [\alpha^{(k-\ell)N} - 1] \\
 & = 0 \quad \text{for } \alpha^N = 1
 \end{aligned} \tag{2.16}$$

Since $(\alpha^{k-\ell}-1) \neq 0$, we conclude that

$$\sum_{n=0}^{N-1} \alpha^{(k-\ell)n} = 0 \tag{2.17}$$

for $(k-\ell) \neq 0$

Hence from (i) and (ii), it is clear that $N^{-1} \sum_{n=0}^{N-1} \alpha^{(k-\ell)n}$ plays the same role as the Kronecker delta function. That is to say

$$\delta_{k,\ell} = N^{-1} \sum_{n=0}^{N-1} \alpha^{(k-\ell)n} = \begin{cases} 1 & \text{if } (k-\ell) = 0 \bmod N \\ 0 & \text{otherwise} \end{cases} \tag{2.18}$$

Therefore eqn. 2.14 reduces to

$$\begin{aligned}
 X(k) &= \sum_{\ell=0}^{N-1} X(\ell) N^{-1} \sum_{n=0}^{N-1} \alpha^{(k-\ell)n} \\
 &= X(k).
 \end{aligned}$$

Thus the expression for $\bar{t}_{k,n}^{-1}$ is confirmed.

At this juncture this proof needs to be examined carefully.

This proof works very well for ordinary arithmetic, however this argument may not be valid for systems using special arithmetic operations, for example, the modulo arithmetic. In this case, even though eqn. 2.16 is true and $(\alpha^{k-l}-1) \neq 0$, we still cannot conclude that eqn. 2.17 is valid. Some other conditions may have to be imposed to confirm the validity of the transform (see below). However, if there is no simpler means available, one can check the transform against eqn. 2.18, which is a more rigid condition for its existence.

The following points give a short summary of what we have discussed so far:

(i) The structure of the elements of the transform matrix $\bar{T}$ and its inverse $\bar{T}^{-1}$ are given by

$$(a) \quad t_{k,n} = \alpha^{kn} \text{ and}$$

$$(b) \quad t_{k,n}^{-1} = N^{-1} \alpha^{-kn}$$

where $n, k=0,1,\dots,N-1$ respectively

(ii) The existence of an $N \times N$ transform having the cyclic convolution property depends on

(a) the existence of an α that is a root of unity of order N ,

(b) the existence of N^{-1} and

(c) the validity of $N^{-1} \sum_{n=0}^{N-1} \alpha^{nm} = \begin{cases} 1 & \text{if } m=0 \text{ mod } N \\ 0 & \text{otherwise} \end{cases}$

(2.19)

Conditions (i) and (ii) are the basic requirements for the existence of a transform having cyclic convolution property. Recall

the arithmetic systems that we are using, the complex number field with the root of unity, $\alpha = e^{-j\frac{2\pi}{N}}$ may form a transform having the cyclic convolution property. This is the kernel of the Discrete Fourier Transform and it is straightforward to check that the DFT satisfies all requirements in (i) and (ii).

On the other hand, in a finite field or ring of integers modulo an integer M, we may also find a transform with cyclic convolution property if the system satisfies (i) and (ii).

2.2 Transforms in a Ring of Integers Modulo an Integer, M

In order to fulfil the requirements in section 2.1 for cyclic convolution we may use modulo arithmetic in a ring of integers. Let

M be the base for modulo arithmetic,

N be the order of the transform (the smallest N
to satisfy eqn. 2.20)

α be a root of unity of order N modulo M

Hence $\alpha^N = 1 \text{ mod } M$, or in the notation of number theory,

$$\langle \alpha^N \rangle_M = 1 \quad (2.20)$$

Consider now eqn. 2.20 and let α and M have the highest common factor, q. That is

$$(\alpha, M) = q$$

Hence $\frac{\alpha^N}{M} = \text{product} \dots \text{remainder}.$

Therefore, we can write

$$\alpha^N = \underset{(p)}{\text{product.M}} + \underset{(r)}{\text{remainder}}$$

But q is the common factor of α and M, it follow that

$$(\alpha')^N q^N = (q.M')p + r$$

or
$$[(\alpha')^N q^{N-1} - M'.p]q = r$$

Hence q must also be a factor of r and the remainder r can never be 1.

This implies that

$$\alpha^N \neq 1 \text{ mod } M, \text{ if } (\alpha, M) \neq 1$$

The implication from this result is that in order to have $\alpha^N = 1 \text{ mod } M$, α and M must be relatively prime. That is

$$(\alpha, M) = 1 \quad (2.21)$$

We have also pointed out that in order to form the inverse transform, N^{-1} must exist in mod M. Let us discuss the relationship of N and M for the existence of N^{-1} . Let $(N, M) \neq 1 = q$ and x be an integer.

$$\frac{N.x}{M} = \underset{(p)}{\text{product.....}} \underset{(r)}{\text{remainder}}$$

$$N.x = M.p + r$$

$$q(N'.x - M'.p) = r$$

Since q is a factor of r, r can never be 1. This implies that there

is no inverse for N . Hence for the existence of an inverse for N ,

$$(N, M) = 1 \quad (2.22)$$

From eqn. 2.19, it is clear that the next condition to be considered for the existence of the transform is that $N^{-1} \sum_{n=0}^{N-1} \alpha^{nm} = 0$ for $m \neq 0 \pmod N$. For the transform in a ring of integers modulo an integer, M , and with the root of unity being an integer, this condition can be simplified to the following form,

$$((\alpha^m - 1), M) = 1 \quad (2.23)$$

$$\text{for } m=1, 2, \dots, N-1$$

Eqn. 2.23 can be obtained from the fact that the product of $(\alpha^{k-\ell} - 1)$ and $\sum_{n=0}^{N-1} \alpha^{(k-\ell)n}$ of eqn. 2.16 is always zero. Hence, if $(\alpha^{k-\ell} - 1)$'s are relatively prime to M , it is impossible to form a product of $(\alpha^{k-\ell} - 1)$ and a number, say x to be a multiple of M , unless x is a multiple of M . This implies that if all $(\alpha^{k-\ell} - 1)$'s are relatively prime to M , $\sum_{n=0}^{N-1} \alpha^{(k-\ell)n}$ must be zero.

From the above discussion, it is obvious that for forming a proper number theoretic transform, $\{\alpha, M, N\}$ should be chosen on the ring of integers under the following conditions

$$\left. \begin{array}{ll} \text{(i)} & (\alpha, M) = 1 \\ \text{(ii)} & (N, M) = 1 \quad \text{and} \\ \text{(iii)} & \left\langle \sum_{n=0}^{N-1} \alpha^{nm} = 0 \right\rangle_M, \text{ for } M \neq 0 \end{array} \right\} \quad (2.24)$$

or (iii) can be written as

$$(iv) ((\alpha^m - 1), M) = 1 \quad (2.25)$$

The nature of the transform depends very much on the choice of α , M and N . Let us derive more characteristics of α , M and N .

Recall that if $(\alpha, M) = 1$, by Euler's Theorem[27-28]

$$\langle \alpha^{\Phi(M)} \rangle_M = 1$$

where $\Phi(M)$ = Euler's totient function, which is the number of integers relatively prime to M . Eqn. 2.20 says that N is the smallest value to give

$$\langle \alpha^N \rangle_M = 1$$

Therefore $\Phi(M)$ must be a multiple of the length, N ,

$$N \mid \Phi(M) \quad (2.26)$$

Since every number can be represented by powers of prime factors of the number, let

$$M = p_1^{r_1} p_2^{r_2} \dots p_\ell^{r_\ell}$$

Recall $\langle \alpha^N = 1 \rangle_M$

Therefore $\alpha^{N-1} = 0 \pmod{p_1^{r_1} p_1^{r_2} \dots p_\ell^{r_\ell}}$

Hence $p_1^{r_1} p_2^{r_2} \dots p_\ell^{r_\ell} \mid (\alpha^{N-1})$

This shows that $p_1^{r_1} p_2^{r_2} \dots p_\ell^{r_\ell}$ divides $(\alpha^N - 1)$ and $(\alpha^N - 1)$ is also divisible by $p_i^{r_i}$ alone, for $i=1,2,\dots$. We can then write

$$\alpha^N = 1 \pmod{p_i^{r_i}} \quad (2.27)$$

From the property of Euler's function[27-28],

$$\Phi(p_i^{r_i}) = p_i^{r_i-1} (p_i - 1)$$

Therefore, from eqns. 2.27 and 2.26, we get

$$N | p_i^{r_i} (p_i - 1) \quad (2.28)$$

From eqn. 2.24, N and M must be mutually prime, clearly N must contain no p_i and N must be a factor of $(p_i - 1)$; or

$$N | (p_i - 1) \quad (2.29)$$

where $i=1,2,\dots$

This implies that

$$N | (p_1 - 1)$$

$$N | (p_2 - 1)$$

..

..

$$N | (p_\ell - 1)$$

To achieve the maximum length

$$N | \gcd\{(p_1-1), (p_2-1), \dots, (p_\ell-1)\} \quad (2.30)$$

2.3 Mersenne Number and Fermat Number Transforms

In order to make the Number Theoretic Transform more attractive, it should be computationally efficient. The transform should involve only simple arithmetic manipulations. It is desirable for the power of α , the root of unity to have a simple binary representation using very few bits. It is obvious that the best choice of α is 2. To facilitate modulo arithmetic, M should also be represented by simple binary word. It seems that the best choice of M is 2 raised to a certain power. However this is contrary to eqn. 2.24 (i) if we also choose α to be 2 or powers of 2. Furthermore, this choice also gives rise to sequence length problem as shown below. The choice of N depends very much on the choice of α and M . If $\alpha=2$, it is also desirable for N to be a power of 2 so that FFT-type of algorithms may be used for fast computation.

The following paragraphs describe the characteristics of various choices of M , α and N .

(i) $M=\text{Even}$

If M = an even number, 2 must be one of the p_i 's in $M = \prod_{i=1}^{\ell} p_i$. Let $p_1 = 2$, therefore $p_1-1 = 1$. By eqn. 2.28, N must divide p_1-1 . This implies that $N|1$. Hence the maximum possible transform length is 1. Of course, this is impractical. M must be odd.

$$(ii) M = 2^k - 1$$

We have emphasized before that M is usually chosen to be an integer with a simple binary representation. The simplest odd numbers are of the form $2^k + 1$. Let us now consider the case, $M = 2^k - 1$.

Let k be a composite number and $k=PQ$, where P is a prime number. Hence $M = 2^{PQ} - 1$. Since

$$2^{PQ} - 1 = (2^P - 1)(2^{(Q-1)P} + 2^{(Q-2)P} + \dots + 2^P + 1),$$

$$(2^P - 1) \mid (2^{PQ} - 1)$$

From eqn. 2.29 it is clear that the maximum possible length of the transform is governed by the possible length of the transform with base M , where $M = 2^P - 1$. In case of choosing $\alpha = 2$, the maximum possible transform length is P . In order to get a reasonable transform length, only prime powers of 2 need to be considered.

Let $M = 2^P - 1$, where $P =$ a prime number and $\alpha = 2$. Numbers of the form $2^k - 1$ are traditionally called Mersenne numbers, hence this transform is called a Mersenne Number Transform. Now, define the Mersenne Number Transform as

$$X(k) = \sum_{n=0}^{N-1} x(n) 2^{nk}$$

$$\text{where } k=0,1,\dots,N-1$$

To find the transform length N , we may write

$$M = 2^P - 1, \text{ hence}$$

$$\langle 2^P = 1 \rangle_M$$

This implies that the length $N = P$. Hence, we may write the Mersenne Number Transform pair as

$$X(k) = \langle \sum_{n=0}^{P-1} x(n) 2^{nk} \rangle_M \quad (2.31)$$

and

$$x(n) = \langle \frac{1}{P} \sum_{k=0}^{P-1} X(k) 2^{-nk} \rangle_M \quad (2.32)$$

where $n, k=0, 1, \dots, P-1$

This transform requires very simple arithmetic operations. The addition of two numbers modulo $M (=2^P-1)$ is simply ones-complement addition in a P -bit word. This can be implemented very simply by using a conventional binary full adder of P bits and by folding the most significant carry bit output back into the least significant carry bit input. For the negation of a number, we can simply make a ones-complement of the number (negating all P bits). For the multiplication of two numbers we may use normal arithmetic adding the P least significant bits to the P most significant bits to get the result.

One even more interesting point is that the multiplication of $\alpha^k (=2^k)$ by a P -bit word, a_n , modulo $M (=2^P-1)$ just requires a circular shift of a_n by k bits.

From eqns. 2.31 and 2.32, it is clear that both the forward transform and the inverse transform require no actual multiplication.

For the computation of cyclic convolutions, the term by term multiplications of the transformed results of the two sequences under question are the only multiplications required for the whole computation process. Hence the number of operations per point required for the computation of cyclic convolution using the MNT becomes:

2(P-1) Mersenne Number Transform additions and

1 Mersenne Number Transform multiplications.

The major drawback of the transform is that the length of the transform is P, which is a fixed length and may be too short for certain applications. Furthermore, P is not a composite number. This implies that it does not have a FFT-type computational algorithm for fast transform computation.

The transform length of the Mersenne Number Transform can be doubled if we choose

$$\alpha = \langle -2^n \rangle_M$$

where n = any number in the set {1,2,...P-1}

In this case

$$\alpha^{2P} = 1 \text{ mod } M$$

However this only produces a minor saving using the FFT-type decomposition. It is easy to check that the choice of α , M and N satisfies the conditions written in eqn. 2.24.

(iii) $M = 2^{k+1}$

If k be an odd number, it can be proved by simple mathematical induction that $M (=2^{k+1})$ is divisible by 3. Hence, $M = 3S$ where S is an integer. By eqn. 2.28, $N|(p_i-1)$. Hence the largest possible transform length is 2 which is not practical. This shows that k must be even.

Now let k be an even number. The most general form of an even number is 2^{2^t} , where S is an odd number and t is an integer. Therefore we can write

$$M = 2^{k+1} = 2^{S2^t+1}$$

However $2^{2^t} + 1$ divides 2^{S2^t+1} . This can actually be seen by simply using the Remainder Theorem.

$$\text{Let } 2^{2^t} + 1 = 0 \text{ and}$$

$$\text{substitutes } 2^{2^t} = -1 \text{ into } 2^{S2^t+1}.$$

$$\text{Hence } 2^{S2^t+1} = (-1)^S + 1 = 0$$

By eqn. 2.29 N must divide the factors, $[(p_1-1), (p_2-1), \dots, (p_\ell-1)]$ of $2^{S2^t} + 1$. The length of the transform is governed by the length possible for $(2^{2^t} + 1)$. This means that if we consider the transform length to be important, the number $(2^{2^t} + 1)$ is of interest.

Since $(2^{2^t} + 1)$ is a Fermat Number, the transform with this base is called the Fermat Number Transform.

For Fermat Number Transforms, $M = F_t = 2^{2^t} + 1$ and α is usually chosen to be 2 or a power of 2. We can now write the Fermat Number Transform pair as

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n) \alpha^{nk} \right\rangle_{F_t} \quad (2.33)$$

and

$$x(n) = \left\langle \frac{1}{N} \sum_{k=0}^{N-1} X(k) \alpha^{-nk} \right\rangle_{F_t} \quad (2.34)$$

for $k, n=0, 1, \dots, N-1$

To find the possible values of the transform length N , we recall $\langle \alpha^N \rangle_{F_t} = 1$. From eqn. 2.29

$$N | (p_1 - 1), (p_2 - 1), \dots, (p_\ell - 1) \text{ of } F_t$$

All F_1, F_2, F_3 and F_4 are primes, hence for F_t , where $t = 1, 2, 3$ or 4

$$N | (F_t - 1)$$

or $N | 2^{2^t}$

We can therefore have a FNT with a transform length as long as $N = 2^{2^t}$.

This result seems very desirable since $N = 2^{2^t}$ is a power of 2 and so a FFT-type computation algorithm can be used to compute the transform. Furthermore, the length of the transform should also be more than enough for most applications.

However, the major restriction comes from a basic property of number theory. After finding $N = 2^{2^t}$, we have to choose an α which satisfies the equation $\alpha^N = 1 \pmod{F_t}$. In the case, α is actually the primitive root of F_t and it is defined as the integer which can generate the largest order. As we mentioned before, the most desirable value for α is 2. However, if we make a detailed examination of the possible values of α with $N = 2^{2^t}$, it shows that 2 is not a primitive root of F_t .

Example 2.1

Let $t = 2$. $F_t = 2^{2^t} + 1 = 17$.

The expected largest order is $(F_t - 1) = 16$. If $\alpha = 2$, we get:

n	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8
Residue	2	4	8	16	15	13	9	1

This shows that 2 is not a primitive root of F_2 . It is a generator of the cyclic group listed above. If $\alpha = 3$

n	3^1	3^2	3^3	3^4	3^5	3^6	3^7	3^8	3^9
Residue	3	9	10	13	5	15	11	16	14

3^{10}	3^{11}	3^{12}	3^{13}	3^{14}	3^{15}	3^{16}
8	7	4	12	2	6	1

This shows that 3 is a primitive root of order $N (=2^{2^2} = 16)$ which is the largest possible transform length.

From the computational point of view, it is better to choose α equal to 2 and to make the adjustment to N to get the proper

transform.

For $\alpha = 2$, consider $F_t = 2^{2^t} + 1$.

$$2^{2^t} + 1 = 0 \pmod{F_t}$$

Let us consider the expression $(2^{2^{t+1}} - 1)$.

$$(2^{2^{t+1}} - 1) = (2^{2^t} + 1)(2^{2^t} - 1)$$

Hence $(2^{2^{t+1}} - 1)$ is divisible by F_t .

$$(2^{2^{t+1}} - 1) = 0 \pmod{F_t}$$

$$2^{2^{t+1}} = 1 \pmod{F_t} \tag{2.35}$$

Now $\alpha = 2$ and

$$2^N = 1 \pmod{F_t} \tag{2.36}$$

Comparing eqns. 2.35 and 2.36 shows that the order of 2 mod F_t does not exceed 2^{t+1} . Hence

$$N \leq 2^{t+1} \tag{2.37}$$

for $\alpha = 2$ or a power of 2

Note that if an integer α is of order N , α^ℓ will be of order $\frac{N}{\ell}$ if $\frac{N}{\ell}$ is an integer. The above result can be seen from the following simple identity

$$(\alpha^\ell)^{\frac{N}{\ell}} = \alpha^N = 1 \pmod{F_t}.$$

Example 2.2

If $3^{16} = 1 \pmod{17}$, $N = 16$ and $\alpha = 3$.

Hence if $\alpha = 3^2$, that is $\ell = 2$,

the order is $\frac{N}{\ell} = \frac{16}{2} = 8$.

Therefore $(3^2)^8 = 1 \pmod{17}$

Only 4 (F_1, F_2, F_3, F_4) Fermat primes have been found. Fermat numbers with t larger than four seem to be composite numbers.

Example 2.3

$$F_5 = 2^{2^5} + 1 = 641.6700417$$

$$F_6 = 2^{2^6} + 1 = 274177.67280421310721$$

However Lucas[27] has proved that every prime factor of a composite F_t is of the form:

$$(K \cdot 2^{t+2} + 1)$$

.where K is an integer

By eqn. 2.29

$$N | 2^{t+2}$$

This shows that the longest possible transform length for the FNT with composite modulo base, F_t is 2^{t+2} . Note that if α is chosen to be 2, eqn. 2.37 is also applicable to the present situation. That is the largest possible transform length is 2^{t+1} for $\alpha = 2$.

Example 2.3

$$\text{For } F_5, N_{\max} = 2^7 = 128 \quad \text{and}$$

$$F_6, N_{\max} = 2^8 = 256$$

There are a number of methods by which we can increase the transform length, N . The simplest method is to choose a better α (of course this α should have properties similar to 2).

If we choose $\alpha = \sqrt{2}$, $\alpha^2 = 2$. Since $\langle 2^{2^{t+1}} = 1 \rangle$, we can increase the transform length by a factor of 2. That is $N = 2^{t+2}$.

Example 2.4

If $M = F_2 = 17$ and $\alpha = 2$, the transform length is $2^{t+1} = 8$.

If we choose $\alpha = \sqrt{2} = 6$, as $6^2 = 2 \pmod{17}$ then $\alpha^{16} = 1$ and the transform length, $N = 16$.

To make the ideas contained here clearer an example is included in Appendix A2.1.

Since, for the Fermat Number Transform, N is a power of 2, we may replace $W = e^{-j\frac{2\pi}{N}}$ by 2 in the FFT algorithm.

Any word defined modulo a Fermat number $F_t = 2^{2^t} + 1$ can be represented as a (2^{t+1}) -bit word. The most significant bit is equal to 1 only if all other bits are equal to zero. All negative numbers can be represented by adding F_t to them.

Example 2.5

If $M = F_2 = 17$

$$-2 = 17 - 2 = 15$$

For a hardware implementation, negation can be realised by

complementing all bits of the word, except for the most significant bit which has to be treated separately; and then add 2 to the result. Hence, negation requires an addition and a complementation together with several auxiliary operations in order to deal with the most significant bit.

The addition is done by an N -bit addition followed by subtracting the carry bit if the $(2^{t+2})^{\text{th}}$ bit is equal to zero. The result is equal to -2 if the $(2^{t+2})^{\text{th}}$ bit is equal to 1.

For the multiplication of a word by 2^d , we may shift the word d times and then subtract the most significant (2^t-1) bits from the least significant 2^t bits if the $(2^{t+1})^{\text{th}}$ bit of the original word is 0. If the $(2^{t+1})^{\text{th}}$ bit of the original word is equal to 1, the result is -2^d . For general multiplication, we can obtain the product by subtracting the higher order (2^t-1) bits of the product register from the lower order 2^t bits of the product register.

This shows that arithmetic modulo a Fermat number is significantly more complex than arithmetic modulo a Mersenne number.

One method to simplify the arithmetic for FNT was introduced by Agarwal and Burrus[40]. They suggested using a 2^t -bit representation of integers which introduces some errors to the results. A technique for the exact computation of the FNT and its implementation were described by McClellan[44]. McClellan's approach involves normal binary weighting with digits ± 1 for the representation of numbers modulo F_t . This approach has been simplified by Leibowitz

whose technique involves the representation of numbers modulo F_t in diminished-one form. The following paragraphs give a brief discussion of this method.

To represent all integers in the ring of integers modulo F_t requires 2^t+1 bits. The additional bit is required in order to represent the number $2^{2^t} = -1 \bmod F_t$. To avoid additions and multiplications involving the additional bit, allow the additional bit to be a 1 only when the number to be represented is 0, i.e. $2^{2^t} = 0 \bmod F_t$. This is achieved by subtracting 1 from the normal binary representation of every integer in the ring. The normal representation and this diminished-1 representation are indicated in Table 2.1 for $2^t = 4$.

Normal (Negative Value)	Binary Representation	Diminished-1 Value
0	00000	1
1 (-1=16)	00001 (10000)	2 (0=0)
2 (-2=15)	00010 (01111)	3 (-1=16)
3 (-3=14)	00011 (01110)	4 (-2=15)
4 (-4=13)	00100 (01101)	5 (-3=14)
5 (-5=12)	00101 (01100)	6 (-4=13)
6 (-6=11)	00110 (01011)	7 (-5=12)
7 (-7=10)	00111 (01010)	8 (-6=11)
8 (-8=9)	01000 (01001)	-8 (-7=10)

Table 2.1: Diminished-1 representation of integers mod F_2

After making this code conversion, the arithmetic for mod F_t

operations is shown to be similar to, and as complex as ordinary ones-complement arithmetic.

Let A and B be two integers in the ring of integers modulo F_t . Hence the diminished-1 representations of these two numbers are $(A-1)$ and $(B-1) \bmod F_t$ respectively. Denoting also the 2^t -least significant bits (lsb) complement by an overbar, hence

$$\begin{aligned}\overline{A-1} &= 2^{2^t} - 1 - (A-1) \\ &= 2^{2^t} + 1 - A - 1 \\ &= (-A) - 1\end{aligned}$$

Thus with this system the negative of a nonzero number in diminished-1 representation is the complement of the 2^t lsb's.

Example 2.6

$$\begin{aligned}\text{If } M = F_2 &= 17 \\ -4 &= \overline{00011} = 01100 = 13 \bmod 17\end{aligned}$$

If the most significant bit (msb) is 1, the negation is inhibited.

To perform addition of two numbers, consider

$$(A-1) + (B-1) = (A+B-1) - 1 \quad \text{and}$$

the desired result is

$$(A+B-1) = [(A-1) + (B-1)] + 1$$

Since the $(2^t + 1)^{\text{th}}$ bit of the addends is used only to inhibit

addition if an addend is zero, addition of nonzero addends involves only the 2^t lsb's. The above equations indicate that a 1 must be added to the sum of two diminished-1 numbers to provide the correct result. However, when a carry is generated from the 2^t -bit sum, a residue reduction modulo F_t requires the subtraction of a 1 since $2^{2^t} = -1 \pmod{F_t}$ and no corrective addition is necessary. Hence the rules for addition in diminished-1 representation become:

- (i) If the msb of either addend is 1, no addition is required.
- (ii) If the msb of both addends are 0, make the addition of the 2^t -bits as normal binary addition and add the complemented carry to the lsb of the sum.

Example 2.7

00110	7	
+ 01100	13	
10010	20	= 3 mod 17
↘ 0		
00010		
00010		

Example 2.8

00001	2	
+ 00010	3	
00011	5	= 5 mod 17
↘ 1		
00100		
00100		

Another important operation for the implementation of FNT is the multiplication of powers of 2. Consider

$$(A-1)2 = (2A-1) - 1 \text{ and}$$

thus $(2A-1) = (A-1)2 + 1$

Therefore, each multiplication by 2 involves a left shift, ignoring the msb, and a corrective addition of a 1. If the bit shifted out from the (2^L) th position is a zero, it is complemented and shifted into the lsb in order to accomplish the adding of a 1. If this bit is a 1, a subtraction of 1 is also required to accomplish a residue reduction. With the corrective addition of +1, these cancel out and a 0 is shifted into the lsb. Hence for each multiplication of 2, a left circular shift of the b lsb's is required and the bit circulated into the lsb is complemented. For the multiplication of 2^k , k left circular shifts of the b lsb's is required and the bits circulated into the k lsb's have to be complemented as well.

Example 2.9

$$6 \times 2^3 = 14 (= -3)$$

0101	6	
0101	3 shifts	
101	complement and circulation	
1101	result	

We have seen that the basic operations (negation, addition and multiplication by a power of 2) for the Fermat Number Transform using the diminished-1 arithmetic are very simple and are similar to

the ones-complement arithmetic. However, for general multiplication, the procedure is more complicated, since

$$\begin{aligned}(A-1)(B-1) &= A.B - (A+B) + 1 \\ &= (A.B-1) - (A+B-1) + 1\end{aligned}$$

and the desired result is

$$(A.B-1) = (A-1)(B-1) + (A+B-1) - 1$$

Besides the multiplication of $(A-1)$ and $(B-1)$, the term $(A+B-1)$ has to be added back to get the final result. Since for the computation of cyclic convolution of using FNT the number of general multiplications required is very small, this is not a serious drawback to the method.

2.4 Extension of the Dynamic Range

Consider the integer ring, $R(0,1,\dots,M-1)$. In order to avoid the convolution sum $y(k)$ being outside the range allowed by R it is necessary to keep

$$y(m) \leq (M-1) \tag{2.38}$$

for positive input data

and
$$-\frac{M-1}{2} \leq y(m) \leq \frac{M-1}{2} \tag{2.39}$$

for positive and negative input data

where $y(m)$ =individual convolution sum
of x 's and h 's

The output dynamic range is defined as

$$y(m)_{\max} = M-1 \approx 2^K \quad (2.40)$$

for positive input data,

where $K = \log_2(M-1)$

and

$$y(m)_{\max} = \frac{M-1}{2} \approx 2^K \quad (2.41)$$

for positive and negative input data,

where $K = \log_2\left(\frac{M-1}{2}\right)$

Consider the case with positive and negative input data, to achieve the results in eqn. 2.39, limits have to be imposed on sequences $\{x(n)\}$ and $\{h(n)\}$.

$$|y(m)| \leq \frac{M-1}{2} \text{ and}$$

$$y(m) = \sum_{n=0}^{N-1} x(n)h(m-n)$$

Note that

$$\left| \sum_{n=0}^{N-1} x(n)h(m-n) \right| \leq \sum_{n=0}^{N-1} |x(n)| \cdot |h(m-n)|$$

Therefore, to avoid overflow

$$\left| \sum_{n=0}^{N-1} x(n)h(m-n) \right| \leq \sum_{n=0}^{N-1} |x(n)| \cdot |h(m-n)| \leq \frac{M-1}{2}$$

Usually there are upper limits for $x(n)$ and $h(n)$. Let $x(n) \leq x_{\max}$

and $h(n) \leq h_{\max}$, therefore

$$\left| \sum_{n=0}^{N-1} x(n)h(m-n) \right| \leq N \cdot x_{\max} h_{\max} \leq \frac{M-1}{2}$$

The condition for input dynamic range is

$$N \cdot x_{\max} h_{\max} \leq \frac{M-1}{2} \quad (2.42)$$

If $x_{\max} = h_{\max}$, the input dynamic range is

$$x_{\max} = \sqrt{\frac{M-1}{2N}} \quad (2.43)$$

It is clear from eqn. 2.43 that the maximum allowable input value, $x(n)$, is restricted. The following paragraphs discuss a method of increasing the dynamic range of the NTT.

The dynamic range of $y(n)$ can be increased by defining the Number Theoretic Transform on the ring of integers modulo M , where

$$M = \prod_{i=1}^{\ell} M_i \quad (2.44)$$

for all $(M_i, M_j) = 1$, for $i \neq j$

The integer ℓ gives the number of factors of M . All the factors of M have to be mutually prime. The arithmetic for the convolution is done mainly over the constituent rings of integers modulo M_i . The final results are then reconstructed by using the Chinese Remainder Theorem (CRT) as shown below

$$y(m) = \left\langle \sum_{i=1}^{\ell} L_i L_i^{-1} y_i(m) \right\rangle_M \quad (2.45)$$

where $L_i = \frac{M}{M_i}$

$\langle L_i L_i^{-1} \rangle_{M_i} = 1$ and

$$y_i(k) = \text{the convolution sum mod } M_i$$

This method gives an output dynamic range of $(M-1) = \prod_{i=1}^{\ell} M_i - 1$. In order to make the point clear, an example is included in Appendix A2.2.

2.5 Pseudo-Number Theoretic Transform

Because of the simplicity of their arithmetic, the FNT and MNT are the most efficient NTTs available for practical implementation. In order to have a wider choice of sequence lengths for FNT and to have FFT-type of algorithms for MNT, we can use pseudo-Number Theoretic Transforms. The pseudo-Mersenne Number Transform[50] is defined modulo an integer, M_1 which is a factor of the number of the form of $M = 2^s - 1$, for $s = \text{composite}$; whereas the pseudo-Fermat Number Transform[51] is defined modulo an integer M_1 , which is a factor of a number of the form of $M = 2^s + 1$, for $s \neq 2^t$. In both cases, M is composite and can always be defined as the product of two factors (not necessary to be prime)

$$M = M_1 M_2$$

In this case, if an N length NTT which supports circular convolution, can be defined modulo M_1 , this NTT computes the N -length convolution $y(m)$ modulo M_1 with

$$y(m) = \left\langle \sum_{n=0}^{N-1} h(n)x(m-n) \right\rangle_{M_1}$$

The arithmetic operations for modulo M_1 are usually difficult.

However, this problem can be solved by making the computation mainly using modulo M arithmetic and then adjusting the final result modulo M_1 at the last stage,

$$y(m) = \left\langle \sum_{n=0}^{N-1} h(n)x(m-n) \right\rangle_{M_1} \right\rangle_M \quad (2.46)$$

The major disadvantage of this method is that all operations modulo M must be executed on word-lengths longer than that of the final result. Furthermore, for the pseudo-Fermat Number Transform, the transform lengths of the resultant NTT is relatively short for α being 2 or a power of 2. Let us consider only the pseudo-Mersenne Number Transform.

In this case M is defined as

$$M = 2^s - 1$$

for $s =$ a composite number

The general form of s is $\prod_{i=1}^{\ell} p_i^{r_i}$, where ℓ is the number of prime factors of s and r_i 's are powers of the prime factors. However for practical cases, s is always either a product of two primes or the square of a prime. If s is a product of two primes, p_1 and p_2 , we can write

$$M = 2^s - 1 = 2^{p_1 p_2} - 1 = (2^{p_1} - 1)(2^{p_2} - 1)K$$

where K is a constant

and

$$2^{p_1 p_2} = 1 \pmod{M} \text{ or } \pmod{\text{a factor of } M}$$

We restrict our discussion to the case where the root of unity is 2 or a power of 2; hence the possible transform lengths are p_1 , p_2 and $p_1 p_2$ with the roots of unity equal to 2^{p_2} , 2^{p_1} and 2 respectively. From eqn. 2.25, it is clear that if the transform length is chosen to be p_1 , the modulus can be defined as

$$M_1 = \frac{M}{2^{p_2} - 1} \quad (2.47)$$

At this stage, we should check the factors of M_1 against eqn. 2.25 for correctness. Further division may be necessary to divide M_1 by factors of M_1 , which is not relatively prime to $(\alpha^m - 1)$, for $m = 1, 2, \dots, N-1$. On the other hand, if we choose $N = p_1 p_2$, the modulus becomes

$$M_1 = \frac{M}{(2^{p_1} - 1)(2^{p_2} - 1)}$$

In this case the dynamic range is unavoidably shorter than in the previous case.

Example 2.10

	N	α	M_1
For $M = 2^{14} - 1$,	7	2^2	$(2^{14} - 1)/(2^2 - 1)$
	14	2	$(2^{14} - 1)/[(2^2 - 1)(2^7 - 1)]$
$M = 2^{15} - 1$,	3	2^5	$(2^{15} - 1)/(2^5 - 1)$
	5	2^3	$(2^{15} - 1)/(2^3 - 1)$
	15	2	$(2^{15} - 1)/[(2^3 - 1)(2^5 - 1)]$

and $M = 2^{35}-1$, 35 2 $(2^{35}-1)/[(2^5-1)(2^7-1)]$

The last set of parameters represents a more practical case.

If s is the square of a prime, p_1 , we can define

$$M_1 = \frac{M}{2^{p_1}-1} \quad (2.48)$$

Example 2.12

If $M = 2^{49}-1$, we find that

$$N = 49, \alpha = 2 \text{ and}$$

$$M_1 = (2^{49}-1)/(2^7-1)$$

This last example gives a transform which can be computed with a 2-stage FFT-type algorithm.

2.6 Complex Number Theoretic Transform

In the previous section, the pseudo-Number Theoretic Transform has been defined to give a large choice of sequence lengths for convolutions. In some cases the pseudo-NTT may also be computed with a FFT-type of algorithm. This method does not, however, give a possible solution to the problem that both the FNT and MNT have relatively short sequence lengths. In this section, we discuss the use of complex Number Theoretic Transforms to solve the sequence length constraint of the NTT. This kind of transforms may also be defined to have a highly composite sequence length N . Hence a fast algorithm can be used to compute the transform.

We begin our discussion of complex Number Theoretic

Transforms with the root of unity being a Gaussian integer, $a+jb$, where

(i) the arithmetic is done over the finite field

$GF(M^2)$, for $M =$ a prime number and

(ii) $\langle j^2 = -1 \rangle_M$ (2.49)

The values of a and b are defined in the field of integers $GF(M)$. We first restrict our discussion to M equal to an odd prime and then we extend the theory to a more general situation: M is a composite number.

Now, let us consider some properties of the powers of the root of unity over the $GF(M^2)$. Consider

$$(a+jb)^M = \sum_{i=0}^M C_k^i a^i b^{M-i} j^{M-i} \quad (2.50)$$

where C_k^i are binomial coefficients
the expansion of $(a+jb)^M$ and is
defined as

$$C_k^i = \frac{M!}{i!(M-i)!}$$

Since all binomial coefficients are integers, $[i!(M-i)!]$ must divide $M!$. M is now a prime number, hence $[i!(M-i)!]$ must divide $(M-1)!$.
Eqn. 2.50 reduces to

$$(a+jb)^M = a^M + j^M b^M \mod M \quad (2.51)$$

From Fermat's theorem, eqn. 2.50 can be written as

$$(a+jb)^M = a + j^M b \mod M \quad (2.52)$$

Now let us raise $(a+j^M b)$ to M power, we get

$$(a+jb)^{M^2} = a + j^{M^2} b$$

Recall the fact that the square of any odd number can always be expressed as $4k+1$, or we can write

$$\langle M^2 \rangle_4 = 1 \quad (2.53)$$

This diophantine equation has only two solutions

$$\langle M_1 = 1 \rangle_4$$

$$\langle M_2 = 3 \rangle_4$$

From eqn. 2.49, we get

$$(a+jb)^{M^2} = a+jb \mod M \quad (2.54)$$

Eqn. 2.54 shows that any NTT defined on the finite field $GF(M^2)$ must satisfy the condition

$$N | (M^2 - 1) \quad (2.55)$$

At the beginning of our discussion, we did not consider the conditions for the existence of j . However, in reality, the existence of j depends on the choice of M for the transform. Since j cannot be a member of $GF(M)$, eqn. 2.49 must not be soluble. The nonsolvability

of eqn. 2.49 is equivalent to the statement that -1 is a quadratic nonresidue[27-28] modulo M . By Euler's criterion, this is further equivalent to

$$(-1/M) = (-1)^{(M-1)/2} = -1 \quad (2.56)$$

where (x/M) is the Legendre symbol

defined by

$$\begin{aligned} (x/M) &= +1, \text{ if } x \text{ is a quadratic} \\ &\quad \text{residue mod } M, \text{ or} \\ &= -1, \text{ if } x \text{ is a quadratic} \\ &\quad \text{nonresidue mod } M \end{aligned}$$

Recall that M is an odd prime, then

$$\frac{M-1}{2} \text{ is even, for } M = 4k + 1$$

$$\frac{M-1}{2} \text{ is odd, for } M = 4k + 3$$

Hence from eqn. 2.56, this imposes a condition for the existence of the complex NTT,

$$\langle M = 3 \rangle_4 \quad (2.57)$$

If M is not a prime number, its prime factorisation is given by

$$M = \prod_{i=1}^{\lambda} q_i^{r_i}$$

where q_i 's are primes,

for ℓ = the number of prime factors of M

For most cases, the values of r_i' s are equal to one. An N -point complex NTT with the cyclic convolution property can be defined in the complex residue ring of Gaussian integers modulo M provided N -point transforms can be defined separately in the constituent finite fields. We may consider that this follows directly from the Chinese Remainder Theorem and so this leads to the conditions for the existence of the transform. It is also clear that not all the constituent finite fields need to be defined on $GF(q^2)$ of Gaussian integers. A constituent field may be defined on a finite field, $GF(q)$ of real integers. However, at least one of the constituent field must be defined on a finite field, $GF(q^2)$ of Gaussian integers in order to form a proper complex Number Theoretic Transform.

From the above discussion, we can also deduce the conditions required for the constituent fields as shown below[59]:

(a) for each prime factor $q = 4k + 1$ of M ,

$$N|(q-1) \quad (2.58)$$

and

(b) for each prime factor $q = 4k + 3$ of M ,

$$N|(q^2-1) \quad (2.59)$$

Example 2.13

If $M = 35$, N must divide $(5-1)$ and (7^2-1) ; hence

$N = 4$ is possible. A possible α is $7+15j$

Consider the computation of two complex sequences $\{a+jb\}$ and $\{c+jd\}$ using complex NTT. Let $\{A_k+jB_k:k=0,1,\dots,N-1\}$ and $\{C_k+jD_k:k=0,1,\dots,N-1\}$ be the transformed results of $\{a+jb\}$ and $\{c+jd\}$ respectively. The term by term products of (A_k+jB_k) and (C_k+jD_k) , for $k=0,1,\dots,N-1$ have to be found before taking the inverse transform. It can be seen that direct computation of each complex product requires 4 multiplications.

$$Y_k + jZ_k = (A_k C_k - B_k D_k) + j(A_k D_k + B_k C_k) \quad (2.60)$$

This figure can be reduced to 3 by computing Y_k with the following algorithm[64].

$$Y_k = (A_k - B_k)(C_k + D_k) - A_k D_k + B_k C_k \quad (2.61)$$

The complex NTT requires three general multiplications per point for the computation of complex cyclic convolutions in addition to α multiplications in the forward and inverse transforms.

Consider now an NTT defined on a ring of integers with each prime factor, $q = 4k + 1$ of the modulus, M . Hence no complex root of unity can be formed for the Number Theoretic Transform and j must be an integer in the ring modulo M . Let $j = \xi$. Hence eqn. 2.60 becomes a real equation as shown below,

$$R_k = (A_k + \xi B_k)(C_k + \xi D_k) \quad (2.62)$$

In order to separate Y_k and Z_k from eqn. 2.62, let us define an

auxiliary convolution

$$U_k = (A_k - \xi B_k)(C_k - \xi D_k) \quad (2.63)$$

Combining eqns. 2.62 and 2.63 gives

$$Y_k = \frac{1}{2} (R_k + U_k) \quad (2.64)$$

$$Z_k = \frac{1}{2\xi} (R_k - U_k) \quad (2.65)$$

This shows that, if ξ is a power of 2, the computation of complex cyclic convolution may simply require 2 multiplications per point using real Number Theoretic Transforms. This makes complex NTTs less attractive for the computation of of complex cyclic convolutions than real NTTs if a suitable real NTT is available.

Because of the simplicity of the arithmetic, the obvious choices of moduli for complex Number Theoretic Transforms are the Fermat numbers and the Mersenne numbers. Since

$$\begin{aligned} \langle F_t \rangle_4 &= 1 \\ \text{for } t=1,2,\dots \end{aligned} \quad (2.66)$$

This shows that no complex NTT can be defined on modulo base F_t . $j = \sqrt{-1}$ can be represented in this ring by $2^{2^{t-1}}$ which is a real number. It is therefore possible to compute directly a complex convolution[65] on modulus F_t by eqns. 2.62-2.65. However, this gives no increase in the sequence length of the transform.

Let us examine the situation for Mersenne numbers. In this

case

$$\langle 2^P - 1 \rangle_4 = \langle 2^{2k'+1} - 1 \rangle_4$$

where $P=2k'+1$, an odd prime

That is

$$\langle 2^P - 1 \rangle_4 = \langle 2^{2k'+1} - 1 \rangle_4 = 3 \quad (2.67)$$

Hence complex Number Theoretic Transforms can be defined, with the root of unity involving $j = \sqrt{-1}$, on the finite ring $GF(M_p^2)$. Note that eqn. 2.67 is true for both Mersenne primes and Mersenne composites. For the case of Mersenne composites, eqn. 2.67 shows that at least one factor of M_p is of the form $(4k+3)$ because the product of $(4k+1)$'s modulo 4 always equal to 1.

For Mersenne primes, $\langle M_p \rangle_4 = 3$, hence from eqn. 2.55

$$N | (M_p^2 - 1)$$

That is

$$N | (M_p + 1)(M_p - 1) \quad (2.68)$$

Both $(M+1)$ and $(M-1)$ are even numbers and the product of two consecutive even numbers must be divisible by 8. Hence a length $8P$ complex NTT may be defined using Mersenne primes.

For Mersenne composites, all factors of M_p of the form $(4k+3)$ have similar properties as prime Mersenne numbers. All prime divisors of M_p are of the form $\langle q_i \pm 1 \rangle_8$ [27]. Hence the prime factors of the

form $(4k+1)$ of M_p are equivalent to the case $\langle q_1=+1 \rangle_8$. Therefore a length $8P$ complex NTT may also be defined using Mersenne composites.

Nussbaumer[50] observed that $(1+j)$ is a root of unity of order $8P$ for complex Mersenne-prime Number Theoretic Transform. From the above discussion, we extend this idea to Mersenne-composite Number Transforms. Hence $(1+j)$ is a root of unity of order $8P$ for both Mersenne-prime and Mersenne-composite complex NTTs.

These length- $8P$ complex Mersenne Number Transforms are promising because: (i) they can be implemented with ones-complement arithmetic; (ii) their sequence length is composite, leading to an efficient FFT-type implementation; and (iii) the roots of unity are simple and no general multiplication is required for the transform.

This method requires 3 general multiplications for the computation of complex cyclic convolutions and so is not a very efficient way to calculate cyclic convolutions of real data.

Reed and Truong[46-47] have also studied the Mersenne-prime NTT with roots of unity being general Gaussian integers. From eqn. 2.68, we have

$$N \mid 2^{P+1}(2^{P-1}-1) \quad (2.69)$$

Hence a complex Mersenne-prime Number Theoretic Transform can be defined with sequence length equal to 2^{P+1} and very efficient radix-2 FFT-type algorithm can be used for the implementation. However, in this case the roots of unity are not in simple binary representations

and so general multiplications are required for the transform.

2.7 Appendices

Appendix A2.1 An Example for the Cyclic Convolution using FNT

Consider the cyclic convolution of the sequences

$$\tilde{x} = \{0, 1, 3, -2\}$$

$$\tilde{h} = \{2, 0, 0, 1\}$$

(a) By direct Calculation

$$y(0) = 0.2 + 1.1 + 3.0 + (-2).0 = 1$$

$$y(1) = 0.0 + 1.2 + 3.1 + (-2).0 = 5$$

$$y(2) = 0.0 + 1.0 + 3.2 + (-2).1 = 4$$

$$y(2) = 0.1 + 1.0 + 3.0 + (-2).2 = -4$$

(b) By Fermat Number Transform

Choose $M = F_2 = 17$, for $t = 2$.

The maximum transform length with powers of 2 as roots of unity is $2^{t+1} = 8$.

$$\zeta^8 = 1 \rangle_{17}$$

For the present calculation, $N = 4$ and $\alpha = 4$.

The transformation matrix $\bar{T}$ is given by

$$\bar{T} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & 4^2 & 4^3 \\ 1 & 4^2 & 4^4 & 4^6 \\ 1 & 4^3 & 4^6 & 4^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & -1 & -4 \\ 1 & -1 & 1 & -1 \\ 1 & -4 & -1 & 4 \end{bmatrix}$$

Note that $4^{-1} = 4^{-1}4^4 = -4$

$$\bar{T}^{-1} = 4^{-1} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4^{-1} & 4^{-2} & 4^{-3} \\ 1 & 4^{-2} & 4^{-4} & 4^{-6} \\ 1 & 4^{-3} & 4^{-6} & 4^{-9} \end{bmatrix} = -4 \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -4 & -1 & 4 \\ 1 & -1 & 1 & -1 \\ 1 & 4 & -1 & -4 \end{bmatrix}$$

Therefore

$$\bar{X} = \bar{T} \cdot \bar{x} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & -1 & -4 \\ 1 & -1 & 1 & -1 \\ 1 & -4 & -1 & 4 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 3 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 9 \\ 4 \\ -15 \end{bmatrix} = \begin{bmatrix} 2 \\ 9 \\ 4 \\ 2 \end{bmatrix} \pmod{17}$$

$$\bar{H} = \bar{T} \cdot \bar{h} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & -1 & -4 \\ 1 & -1 & 1 & -1 \\ 1 & -4 & -1 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ -2 \\ 1 \\ 6 \end{bmatrix} \pmod{17}$$

$$\bar{Y} = \bar{X} \cdot \bar{H} = \begin{bmatrix} 6 \\ -18 \\ 4 \\ 12 \end{bmatrix} = \begin{bmatrix} 6 \\ -1 \\ 4 \\ -5 \end{bmatrix} \pmod{17}$$

Taking the inverse transform, we have

$$\bar{y} = -4 \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -4 & -1 & 4 \\ 1 & -1 & 1 & -1 \\ 1 & 4 & -1 & -4 \end{bmatrix} \begin{bmatrix} 6 \\ -1 \\ 4 \\ -5 \end{bmatrix} = \begin{bmatrix} 1 \\ 5 \\ 4 \\ -4 \end{bmatrix} \pmod{17}$$

Appendix A2.2 An Example of the Extension of Dynamic Range by CRT

The dynamic range of $y(k)$ can be increased by choosing two relatively primes bases, M_a and M_b . We can represent $x(n)$ and $h(n)$ on these bases respectively and do the convolution on each of these bases to obtain two sets of $y_i(n)$'s. By making use of the Chinese Remainder Theorem, we can then obtain the result. The dynamic range of the convolution is $(M_a M_b - 1)$. This method can be generalized by choosing as many as necessary mutually prime bases, so that the dynamic range can reach $(M_b M_b \dots - 1)$.

$$\text{Let } \tilde{x} = \{4, 0\}$$

$$\tilde{h} = \{1, 2\}$$

(a) Without using a transform technique,

$$y(m) = \sum_{n=0}^{N-1} x(n)h(m-n)$$

$$y(0) = x(0)h(0) + x(1)h(1) = 4 \cdot 1 + 0 \cdot 2 = 4$$

$$y(1) = x(0)h(1) + x(1)h(0) = 4 \cdot 2 + 0 \cdot 1 = 8$$

(b) Using a Number Theoretic Transform

(i) Choose $M_a = 3$ and

$$M_b = 5$$

The dynamic range = $3 \times 5 - 1$

$$= 14$$

(ii) The convolution of $\tilde{x}$ and $\tilde{h}$ over M_a

$$\bar{x}_{\text{mod } 3} = \bar{x}_a = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \bar{h}_{\text{mod } 3} = \bar{h}_a = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Use an NTT of $\alpha = 2$, $N = 2$ and $M = 3$.

$$\begin{aligned} x_k^a &= \sum_{n=0}^{N-1} x(n) 2^{nk} = \sum_{n=0}^1 x(n) 2^{nk} \\ &= x(0) 2^0 + x(1) 2^k \end{aligned}$$

Hence

$$x_0^a = 1.1 + 0.2^0 = 1$$

$$x_1^a = 1.1 + 0.2 = 1$$

$$h_k^a = h(0) 2^0 + h(1) 2^k$$

Hence

$$h_0^a = 1.1 + 2.2^0 = 0$$

$$h_1^a = 1.1 + 2.2 = 2$$

where the superscript signifies that the operation is done over $M_a=3$

$$y_k^a = x_k^a \cdot h_k^a$$

$$y_0^a = 1.0 = 0$$

$$y_1^a = 1.2 = 2$$

$$\begin{aligned} y_m^a &= (N^a)^{-1} \sum_{k=0}^{N-1} y_k^a 2^{-mk} \\ &= 2^{-1} [y_0^a 2^0 + y_1^a 2^{-1}] \end{aligned}$$

Hence

$$y_0^a = 2[0.1 + 2.1] = 1$$

$$y_1^a = 2[0.1 + 2.2^{-1}] = 2[4] = 2$$

(iii) The convolution of $\tilde{x}$ and $\tilde{h}$ over M_b

$$\bar{x}_{\text{mod } 5} = \bar{x}_b = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$$

$$\bar{h}_{\text{mod } 5} = \bar{h}_b = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Similarly, using the NTT with $\alpha = 4$, $N = 2$ and $M = 5$

we get

$$y_0^b = 4 \text{ mod } 5$$

$$y_1^b = 3 \text{ mod } 5$$

(iv) By the Chinese Remainder Theorem

$$y_k = \left[\sum_{i=a}^b y_{kL_i}^i L_i^{-1} \right] \text{ mod } M_a M_b$$

where $L_i = \frac{M_a M_b}{M_i}$ and

$$M_i = M_a \text{ or } M_b$$

for $k=0,1$

$$= y_{kL_a}^a L_a^{-1} + y_{kL_b}^b L_b^{-1}$$

That is

$$y_0 = y_{0L_a}^a L_a^{-1} + y_{0L_b}^b L_b^{-1}$$

$$y_1 = y_{1L_a}^a L_a^{-1} + y_{1L_b}^b L_b^{-1}$$

To find L_a , L_b and their reciprocals,

$$L_a = \frac{3 \times 5}{3} = 5$$

$$L_b = \frac{3 \times 5}{5} = 3$$

$$L_a^{-1} = 5^{-1} = 2 \pmod{3}$$

$$L_b^{-1} = 3^{-1} = 2 \pmod{5}$$

$$\text{Hence } y_k = [y_k^a 5.2 + y_k^b 3.2]$$

$$y_k = 10y_k^a + 6y_k^b$$

$$y_0^a = [10.1 + 6.4] = 4 \pmod{15}$$

$$y_1^a = [10.2 + 6.3] = 8 \pmod{15}$$

CHAPTER THREE CYCLIC CONVOLUTION OF LONG SEQUENCES USING NUMBER THEORETIC TRANSFORM

It has been pointed out in chapter 2 that all Number Theoretic Transforms are limited by word-length and sequence length constraints. The normal transform length of the MNT is P , for a modulo base of $(2^P - 1)$ and the MNT transform length can be increased to $2P$ when $\alpha = -2$. Agarwal and Burrus[40] have pointed out that a more practical choice of Fermat number, the modulus, is F_5 , in which case $M = F_5 = 2^{32} + 1$. The corresponding maximum transform length is 128 and $\alpha = \sqrt{2}$. This length may be too short for certain applications, such as radar or picture processing.

In this chapter Number Theoretic Transforms are examined and expressed in a systematic way using polynomial formulations. We first discuss the relationship between the the circular shift theorem of the z -transform and the circular shift theorem of the Number Theoretic Transform. This forms an extremely useful tool for the computation of cyclic convolutions using multi-dimensional techniques. We then derive some new formulations for the efficient computation of one-dimensional convolutions in section 3.2 and two-dimensional convolutions in section 3.3. This approach leads to our development of the nested Number Theoretic Transforms for fast cyclic convolutions. In order to make our discussion clear appendices are included to give numerical examples.

The contents of the present chapter have been published in

[67].

3.1 Circular Shift Theorem

Consider the following time-domain sequence

$$\tilde{x}(n) = \{ x(0), x(1), \dots, x(N-1) \}$$

where N is the length of the sequence

Let M be the base for modulo arithmetic in a Number Theoretic Transform. The NTT of the sequence $\tilde{x}(n)$ can be written as

$$T\{x(n)\} = X(k) = \left\langle \sum_{n=0}^{N-1} x(n) \alpha^{kn} \right\rangle_M \quad (3.1)$$

where $k = 0, 1, \dots, N-1$

Hence

$$T\{x(n-\ell)\} = \left\langle \sum_{n=0}^{N-1} x(n-\ell) \alpha^{kn} \right\rangle_M$$

In this last equation, substitute $r=n-\ell$, so that

$$\begin{aligned} T\{x(n-\ell)\} &= \left\langle \alpha^{\ell k} \sum_{r=0}^{N-1} x(r) \alpha^{kr} \right\rangle_M \\ &= \alpha^{\ell k} X(k) \pmod{M} \end{aligned} \quad (3.2)$$

Equ. 3.2 shows that a circular shift of the input sequence gives rise to a circular shift of the transformed output. This is actually the Shift Theorem [40] of cyclic convolution.

Now let us define the z-transform* of the same sequence [24]

$$Z\{x(n)\} = X(z) = \sum_{n=0}^{N-1} x(n)z^n \quad (3.3)$$

Hence

$$\begin{aligned} Z\{x(n-l)\} &= z^l \sum_{n=0}^{N-1} x(n)z^n \\ &= z^l X(z) \end{aligned} \quad (3.4)$$

It can be seen from eqns. 3.2 and 3.4 that the Shift Theorem for the NTT and for the z-transform are related by the following expression

$$z^l X(z) \longleftrightarrow \alpha^{lk} X(k) \quad (3.5)$$

*This is the normal z-transform, except that for convenience positive instead of negative powers of z are used in order to simplify the presentation using polynomial arithmetic.

3.2 One-Dimensional Circular Convolution

Consider the length of a sequence to be $N = N_1 N_2$, a composite number. The one-dimensional z-transform of this sequence can be mapped into a two-dimensional form by making use of the following equations,

$$n = N_2 n_1 + n_2 \quad (3.6)$$

$$\text{where } n_1 = 0, 1, \dots, N_1 - 1$$

$$n_2 = 0, 1, \dots, N_2 - 1$$

and

$$z_1 = z^{N_2} \quad (3.7)$$

Under these conditions, we can write

$$x(n) = x_{n_2, n_1}$$

and eqn 3.3 becomes

$$X(z, z_1) = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x_{n_2, n_1} z_1^{n_1} z^{n_2} \quad (3.8)$$

Notice that (i) the map is unique and cyclic [68] in n_1 , (ii) the map is unique also but not cyclic in n_2 . Similarly, we may define the two-dimensional transform of the sequence $\tilde{h}(m) = \{h(0), h(1), \dots, h(N-1)\}$ as

$$H(z, z_1) = \sum_{m_2=0}^{N_2-1} \sum_{m_1=0}^{N_1-1} h_{m_2, m_1} z_1^{m_1} z^{m_2} \quad (3.9)$$

The cyclic convolution of $x(n)$ and $h(m)$ can be written as

$$y_\ell = \sum_{n=0}^{N-1} x_n h_{\ell-n} \quad (3.10)$$

where $\ell=0,1,\dots,N-1$

This can be expressed as a two-dimensional z -transform using eqns. 3.8 and 3.9

$$\begin{aligned} X(z, z_1) H(z, z_1) &= \left(\sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x_{n_2, n_1} z_1^{n_1} z^{n_2} \right) \\ &\quad \left(\sum_{m_2=0}^{N_2-1} \sum_{m_1=0}^{N_1-1} h_{m_2, m_1} z_1^{m_1} z^{m_2} \right) \\ &\quad \text{mod}(z_1^{N_1-1}) \\ &= \sum_{n_2=0}^{N_2-1} \sum_{m_2=0}^{N_2-1} \left[\left(\sum_{n_1=0}^{N_1-1} x_{n_2, n_1} z_1^{n_1} \right) \right. \\ &\quad \left. \left(\sum_{m_1=0}^{N_1-1} h_{m_2, m_1} z_1^{m_1} \right) \right] z^{n_2+m_2} \\ &\quad \text{mod}(z_1^{N_1-1}) \end{aligned} \quad (3.11)$$

Since n_1 and m_1 are cyclic, the product inside the square bracket actually corresponds to a cyclic convolution. Let us define

$$x_{n_2}(z_1)H_{m_2}(z_1) = \left(\sum_{n_1=0}^{N_1-1} x_{n_2, n_1} z_1^{n_1} \right) \left(\sum_{m_1=0}^{N_1-1} h_{m_2, m_1} z_1^{m_1} \right) \quad (3.12)$$

$$\text{mod}(z_1^{N_1-1})$$

where $n_2, m_2 = 0, 1, \dots, N_2 - 1$

Hence we may write

$$X(z, z_1)H(z, z_1) = \sum_{n_2=0}^{N_2-1} \sum_{m_2=0}^{N_2-1} x_{n_2}(z_1)H_{m_2}(z_1)z^{n_2+m_2} \quad (3.13)$$

Evaluating eqn 3.12 at $z_1 = \alpha^{k_1}$, so as to obtain the Number Theoretic Transforms of the sequences $\{x_{n_2, n_1} : n_2=0, 1, \dots, N_2 \text{ and } n_1=0, 1, \dots, N_1\}$ and $\{h_{m_2, m_1} : m_2=0, 1, \dots, N_2 \text{ and } m_1=0, 1, \dots, N_1\}$ with respect to n_1 and m_1 , and substituting into eqn. 3.13, we obtain

$$X_{k_1}(z)H_{k_1}(z) = \sum_{n_2=0}^{N_2-1} \sum_{m_2=0}^{N_2-1} x_{n_2, k_1} H_{m_2, k_1} z^{n_2+m_2} \quad (3.14)$$

where $k_1 = 0, 1, \dots, N_1 - 1$,

$$x_{n_2, k_1} = \left\langle \sum_{n_1=0}^{N_1-1} x_{n_2, n_1} \alpha^{n_1 k_1} \right\rangle_M \quad (3.15)$$

$$H_{m_2, k_1} = \left\langle \sum_{m_1=0}^{N_1-1} h_{m_2, m_1} \alpha^{m_1 k_1} \right\rangle_M \quad (3.16)$$

At this juncture eqn. 3.14 needs to be examined carefully. It should be noted that this expression is not cyclic in n_2 and m_2 , and the contribution given by $z^{N_2} (z^{N_2} = z_1)$ has not been taken into account. Usually $z^{n_2+m_2}$ can be expressed as $z_1^{\gamma_{n_2,m_2}} z^{\eta_{n_2,m_2}}$ where γ_{n_2,m_2} is defined as the integral part of $(n_2+m_2)/N_2$ and η_{n_2,m_2} is defined as $(n_2+m_2 - N_2 \cdot \gamma_{n_2,m_2})$. As we have already seen from eqn. 3.5 the term $z_1^{\gamma_{n_2,m_2}}$ introduces only circular shifts (if α is a power of 2) to the transformed sequences of X_{n_2,k_1} and H_{m_2,k_1} , or indeed that the transformed product $X_{n_2,k_1} H_{m_2,k_1}$ should be multiplied by $\alpha^{k_1 \gamma_{n_2,m_2}}$. However, this modification could be done at a later stage or after the expansion of the polynomial eqn. 3.14 has been computed.

Eqn. 3.14 is actually a linear convolution sum of the sequences $\{X_{n_2,k_1}, n_2=0,1,\dots,N_2-1\}$ and $\{H_{m_2,k_1}, m_1=0,1,\dots,N_2-1\}$ or

$$\begin{aligned} X_{k_1}(z)H_{k_1}(z) &= \left(\sum_{n_2=0}^{N_2-1} X_{n_2,k_1} z^{n_2} \right) \left(\sum_{m_2=0}^{N_2-1} H_{m_2,k_1} z^{m_2} \right) \\ &\triangleq W_{0,k_1} + W_{1,k_1} z + \dots W_{2N_2-2,k_1} z^{2N_2-2} \end{aligned} \quad (3.17)$$

where $k_1=0,1,\dots,(N_1-1)$

A direct implementation of eqn. 3.14 requires N_2^2 multiplications for N_2 points. However, we may implement this with a much lower number of multiplications using eqn. 3.17. Two ways for doing so are shown below.

(i) Lagrange Interpolating Formula

Eqn. 3.17 is a product of two polynomials and indeed the actual answers to the polynomial multiplication problem are the coefficients, $W_{0,k_1}, W_{1,k_1}, \dots, W_{2N_2-2,k_1}$ of the resulting product polynomial. These $(2N_2-1)$ coefficients of the $(2N_2-2)^{th}$ order polynomial resulting from the product can be interpolated exactly through $(2N_2-1)$ points using the Lagrange Interpolation Formula (or more precisely, the Cook-Toom Algorithm [24,69]). By Lagrange Interpolation Formula, it is meant that,

$$\begin{aligned} Y_{k_1}(z) &= W_{0,k_1} + W_{1,k_1}z + \dots + W_{2N_2-2,k_1}z^{2N_2-2} \\ &= \sum_{i=0}^{2N_2-2} m_{k_1,i} L_i(z) \end{aligned} \quad (3.18)$$

where (a) the interpolating polynomials
are given by

$$L_i(z) = \prod_{j \neq i} \frac{z - z_j}{z_i - z_j} \quad (3.19)$$

and

(b) the interpolating constants are

given by

$$m_{k_1,i} = X_{k_1}(z_i) H_{k_1}(z_i) \quad (3.20)$$

for $i=0,1,\dots,2N_2-2$

In the above expression $\{z_0, z_1, \dots, z_{2N_2-2}\}$ are distinct numbers. As shown in eqn. 3.20, this method requires only $(2N_2-1)$ multiplications for the calculation of N_2 points, or $(2 - \frac{1}{N_2})$ multiplications per point. This method is extremely useful for small values of N_2 , $N_2 \leq 3$ for example, but for relatively large values of N_2 the interpolating polynomials become more and more complicated, thereby making it difficult to apply the Lagrange Interpolation Formula to find the result.

(ii) A Second Number Theoretic Transform

As already pointed out before eqn. 3.17 is in fact a linear convolution sum of two sequences each of length N_2 . The result of this linear convolution can be found by a length- $(2N_2-1)$ cyclic convolution [1-2, 26] of two $(2N_2-1)$ sequences formed by appending (N_2-1) zeros at the end of the original sequences. For example, the linear convolution of $\{x_0, x_1, x_2\}$ and $\{h_0, h_1, h_2\}$ can be found by the cyclic convolution of $\{x_0, x_1, x_2, 0, 0\}$ and $\{h_0, h_1, h_2, 0, 0\}$; or in general, it may be found from

$$\{x_{0,k_1}, x_{1,k_1}, \dots, x_{N_2-1,k_1}, \underbrace{0, \dots, 0}_{(N_2-1)\text{zeros}}\} \circledast \{h_{0,k_1}, h_{1,k_1}, \dots, h_{N_2-1,k_1}, \underbrace{0, \dots, 0}_{(N_2-1)\text{zeros}}\}$$

where $\circledast$ means cyclic convolution. This cyclic convolution may be computed using the Winograd short convolution algorithms[25] or the Number Theoretic Transform again. The

former method requires $(2(2N_2-1)-K)$ multiplications for N_2 points, whereas the latter method requires $(2N_2-1)$ multiplications for N_2 points, where K is the number of divisors of $(2N_2-1)$ including 1 and N . In the case of the Number Theoretic Transform the best transform length is $(2N_2-1)$ which is an odd number and so Mersenne Number Transforms, Pseudo Mersenne Transforms [50] and Pseudo Fermat Transforms [51] can be used.

$$x'_{k'_2, k_1} = \left\langle \sum_{n'_2=0}^{2N_2-2} x_{n'_2, k_1} \delta^{k'_2 n'_2} \right\rangle_{M'} \quad (3.21)$$

where $k'_2 = 0, 1, \dots, 2N_2-2$

M' = the base for modulo arithmetic

δ = the root of unity of the transform

Since the results of the second NTT are the sum of the products $x_{n'_2, k_1} h_{n'_2, k_1}$, the word-length for M' should be at least twice as large as for M . This is the major requirement that has to be satisfied in order to reduce the number of multiplications involved in the convolution. However, this particular requirement is eliminated if M' is chosen to be M . This is possible since the evaluation of eqn. 3.17 is to be done modulo M . This reduction in modulo length is of great importance in actual applications. This not only reduces the word-length of the multiplication but also simplifies some other processes involved in the transformation. For example,

the results of the second NTT have to be reduced to M before they can be applied to eqn. 3.17.

$$W_{n'_2, k_1} = \langle \langle \frac{1}{2^{N_2}-1} \sum_{k'_2=0}^{2^{N_2}-2} X'_{k'_2, k_1} H'_{k'_2, k_1} \delta^{-k'_2 n'_2} \rangle_{M'} \rangle_M \quad (3.22)$$

where $n'_2=0,1,\dots,(2^{N_2}-2)$

However, if $M' = M$, eqn. 3.22 reduces to

$$W_{n'_2, k_1} = \langle \frac{1}{2^{N_2}-1} \sum_{k'_2=0}^{2^{N_2}-2} X'_{k'_2, k_1} H'_{k'_2, k_1} \delta^{-k'_2 n'_2} \rangle_M \quad (3.23)$$

Only one reduction is required for this step if we use eqn. 3.23 instead of eqn. 3.22. Simplicity in programming the software implementation and also simplicity in the construction of hardware implementations are the consequences of choosing both moduli to be the same. For example, the cyclic convolution of two length 496 ($=31 \times 16$) sequences can be implemented by two length 31 cyclic convolution using a MNT with a modulo base of $M=2^{31}-1$, $\alpha=2$ and $N=31$. The number of multiplications per output point is always less than two.

To make the ideas contained here clearer an example is included in Appendix A3.1

3.3 Two-Dimensional Circular Convolution

In this section we examine the two-dimensional case of the

cyclic convolution[71] and derive results parallel to the previous cases. The basic concepts remain the same and the saving in the number of multiplications achievable is considerable.

The z-transform of the two-dimensional sequences $\tilde{x}(n,r)=\{x(n,r):n,r=0,1,\dots,N-1\}$ and $\tilde{h}(m,s)=\{h(m,s):m,s=0,1,\dots,N-1\}$ can be written as

$$X(z,q) = \sum_{n=0}^{N-1} \sum_{r=0}^{N-1} x_{n,r} q^r z^n \quad (3.24)$$

$$H(z,q) = \sum_{m=0}^{N-1} \sum_{s=0}^{N-1} h_{m,s} q^s z^m \quad (3.25)$$

Now if the length N is a composite number, then let $N=N_1N_2$ and

$$n = N_2 n_1 + n_2$$

$$r = N_2 r_1 + r_2$$

$$m = N_2 m_1 + m_2$$

$$s = N_2 s_1 + s_2$$

where $n_1, \dots, N_1-1$

$n_2, \dots, N_2-1$

hence

$$x(n,r) = x_{n_2, n_1, r_2, r_1}, \text{ and}$$

$$h(m,s) = h_{m_2, m_1, s_2, s_1}$$

Also let

$$z_1 = z^{N_2}$$

$$q_1 = q^{N_2}$$

Eqns. 3.24 and 3.25 can now be written as

$$X(z, z_1, q, q_1) = \sum_{n_2=0}^{N_2-1} \sum_{r_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} \sum_{r_1=0}^{N_1-1} x_{n_2, n_1, r_2, r_1} q_1^{r_1} z_1^{n_1} q^{r_2} z^{n_2} \quad (3.26)$$

$$H(z, z_1, q, q_1) = \sum_{m_2=0}^{N_2-1} \sum_{s_2=0}^{N_2-1} \sum_{m_1=0}^{N_1-1} \sum_{s_1=0}^{N_1-1} h_{m_2, m_1, s_2, s_1} q_1^{s_1} z_1^{m_1} q^{s_2} z^{m_2} \quad (3.27)$$

On taking the Number Theoretic Transform with respect to (n_1, r_1) and (m_1, s_1) respectively we obtain

$$X_{k_1, \ell_1}(z, q) = \sum_{n_2=0}^{N_2-1} \sum_{r_2=0}^{N_2-1} x_{n_2, r_2, k_1, \ell_1} q^{r_2} z^{n_2}$$

and

$$H_{k_1, \ell_1}(z, q) = \sum_{m_2=0}^{N_2-1} \sum_{s_2=0}^{N_2-1} h_{m_2, s_2, k_1, \ell_1} q^{s_2} z^{m_2}$$

where

$$x_{n_2, r_2, k_1, \ell_1} = \left\langle \sum_{n_1=0}^{N_1-1} \sum_{r_1=0}^{N_1-1} x_{n_2, n_1, r_2, r_1} \beta^{r_1 \ell_1} \alpha^{n_1 k_1} \right\rangle_M \quad (3.28)$$

$$H_{m_2, s_2, k_1, \ell_1} = \left\langle \sum_{m_1=0}^{N_1-1} \sum_{s_1=0}^{N_1-1} h_{m_2, m_1, s_2, s_1} \beta^{s_1 \ell_1} \alpha^{m_1 k_1} \right\rangle_M \quad (3.29)$$

$$k_1, \ell_1 = 0, 1, \dots, N_1-1$$

In the above equations β and α are the roots of unity of the transform and usually $\beta = \alpha$. The NTT of the convolution of the sequences $x(n, r)$ and $h(m, s)$ can now be written as

$$\begin{aligned} X_{k_1, \ell_1}(z, q) H_{k_1, \ell_1}(z, q) \\ = \left[\sum_{n_2=0}^{N_2-1} \sum_{r_2=0}^{N_2-1} x_{n_2, r_2, k_1, \ell_1} q^{r_2} z^{n_2} \right] \left[\sum_{m_2=0}^{N_2-1} \sum_{s_2=0}^{N_2-1} h_{m_2, s_2, k_1, \ell_1} q^{s_2} z^{m_2} \right] \end{aligned} \quad (3.30)$$

$$\text{where } k_1, \ell_1 = 0, 1, \dots, (N_1-1)$$

This equation is similar to eqn 3.17 for the one-dimensional convolution. Direct computation of eqn. 3.30 requires $[N_2^2]^2 = N_2^4$ multiplications for N_2^2 points. However the computation of eqn. 3.30 using the two-dimensional Lagrange Interpolation Formula as before requires only $(2N_2-1)^2$ multiplications for N_2^2 points. The number of multiplications per point is then $(2-\frac{1}{N_2})^2$ which is always less than 4.

In order to clarify the idea presented in this section, let us consider a practical example. If $N=256$ then we can let $N_2=2$,

$N_1=128$ and use Fermat number F_5 as the base for modulo arithmetic, that is $M=F_5=2^{32}+1$. The cyclic convolution of the sequences $\{x(n,r): n,r=0,1,\dots,255\}$ and $\{h(m,s): m,s=0,1,\dots,255\}$ can then be written as

$$y_{u,v} = \sum_{n=0}^{255} \sum_{r=0}^{255} x(n,r)h(u-n,v-r)$$

where $u,v = 0,1,\dots,255$

Hence eqn. 3.30 becomes

$$X_{k_1, \ell_1}(z, q) H_{k_1, \ell_1}(z, q) = \left[\sum_{n_2=0}^1 \sum_{r_2=0}^1 x_{n_2, r_2, k_1, \ell_1} q^{r_2} z^{n_2} \right] \left[\sum_{m_2=0}^1 \sum_{s_2=0}^1 H_{m_2, s_2, k_1, \ell_1} q^{s_2} z^{m_2} \right]$$

where

$$x_{n_2, r_2, k_1, \ell_1} = \left\langle \sum_{n_1=0}^{127} \sum_{r_1=0}^{127} x_{n_2, n_1, r_2, r_1} \beta^{r_1 \ell_1} \alpha^{n_1 k_1} \right\rangle_{F_5}$$

$$H_{m_2, s_2, k_1, \ell_1} = \left\langle \sum_{m_1=0}^{127} \sum_{s_1=0}^{127} h_{m_2, m_1, s_2, s_1} \beta^{s_1 \ell_1} \alpha^{m_1 k_1} \right\rangle_{F_5}$$

and

$$k_1, \ell_1 = 0, 1, \dots, 127$$

$$\begin{aligned} &= X_{0,0,k_1,\ell_1} H_{0,0,k_1,\ell_1} + (X_{0,0,k_1,\ell_1} H_{0,1,k_1,\ell_1} \\ &+ X_{0,1,k_1,\ell_1} H_{0,0,k_1,\ell_1})q + (X_{0,1,k_1,\ell_1} H_{0,1,k_1,\ell_1})q^2 \\ &+ (X_{1,0,k_1,\ell_1} H_{0,0,k_1,\ell_1} + X_{0,0,k_1,\ell_1} H_{1,0,k_1,\ell_1})z \\ &+ (X_{1,0,k_1,\ell_1} H_{1,0,k_1,\ell_1})z^2 + (X_{0,0,k_1,\ell_1} H_{1,1,k_1,\ell_1} \\ &+ X_{1,1,k_1,\ell_1} H_{0,0,k_1,\ell_1} + X_{0,1,k_1,\ell_1} H_{1,0,k_1,\ell_1} \end{aligned}$$

$$\begin{aligned}
& + X_{1,0,k_1,\ell_1} H_{0,1,k_1,\ell_1} zq + (X_{0,1,k_1,\ell_1} H_{1,1,k_1,\ell_1} \\
& + X_{1,1,k_1,\ell_1} H_{0,1,k_1,\ell_1} zq^2 + (X_{1,0,k_1,\ell_1} H_{1,1,k_1,\ell_1} \\
& + X_{1,1,k_1,\ell_1} H_{1,0,k_1,\ell_1} z^2 q + (X_{1,1,k_1,\ell_1} H_{1,1,k_1,\ell_1} z^2 q^2 \\
& = W_{0,0,k_1,\ell_1} + W_{0,1,k_1,\ell_1} q + W_{0,2,k_1,\ell_1} q^2 + W_{1,0,k_1,\ell_1} z \\
& + W_{2,0,k_1,\ell_1} z^2 + W_{1,1,k_1,\ell_1} qz + W_{1,2,k_1,\ell_1} zq^2 + W_{2,1,k_1,\ell_1} z^2 q \\
& + W_{2,2,k_1,\ell_1} z^2 q^2 \quad (3.31)
\end{aligned}$$

For a direct implementation of this equation we need to make 2^4 multiplications. If the two-dimensional Lagrange Interpolation Formula is used, a total of $(2^2-1)^2$ (=9) multiplications are enough to evaluate 2^2 points. The two-dimensional Lagrange Interpolation Formula can be written as

$$W_{k_1,\ell_1}(z,q) = \sum_{i=0}^{2N_2-2} \sum_{j=0}^{2N_2-2} m_{k_1,\ell_1,i,j} L_{i,j}(z,q) \quad (3.32)$$

$$\text{where } m_{k_1,\ell_1,i,j} = X_{k_1,\ell_1}(z_i, q_j) H_{k_1,\ell_1}(z_i, q_j) \quad (3.33)$$

$$\text{and } L_{i,j}(z,q) = \prod_{u \neq i} \frac{z-z_u}{z_i-z_u} \prod_{v \neq j} \frac{q-q_v}{q_j-q_v} \quad (3.34)$$

for $i,j = 0,1,\dots,2N_2-2$

(z_i, q_i) are distinct ordered pairs for interpolation. Now let us

choose $\{-1,0,1\}$ for z_i, q_j in eqn. 3.33

$$\{(z_i, q_j), i, j=0,1,2\}$$

$$= \{(-1,-1), (-1,0), (-1,1), (0,-1), (0,0), (0,1), (1,-1), (1,0), (1,1)\}$$

Eqn. 3.33 gives

$$\begin{aligned} m_{k_1, \ell_1, 0, 0} &= [X_{0,0,k_1, \ell_1}^{-} X_{0,1,k_1, \ell_1}^{-} X_{1,0,k_1, \ell_1}^{+} X_{1,1,k_1, \ell_1}^{+}] \\ &\quad [H_{0,0,k_1, \ell_1}^{-} H_{0,1,k_1, \ell_1}^{-} H_{1,0,k_1, \ell_1}^{+} H_{1,1,k_1, \ell_1}^{+}] \\ &= X'_{0,0,k_1, \ell_1} H'_{0,0,k_1, \ell_1} \end{aligned}$$

$$\begin{aligned} m_{k_1, \ell_1, 0, 1} &= [X_{0,0,k_1, \ell_1}^{-} X_{1,0,k_1, \ell_1}^{-}] [H_{0,0,k_1, \ell_1}^{-} H_{1,0,k_1, \ell_1}^{-}] \\ &= X'_{0,1,k_1, \ell_1} H'_{0,1,k_1, \ell_1} \end{aligned}$$

$$\begin{aligned} m_{k_1, \ell_1, 0, 2} &= [X_{0,0,k_1, \ell_1}^{+} X_{0,1,k_1, \ell_1}^{+} X_{1,0,k_1, \ell_1}^{-} X_{1,1,k_1, \ell_1}^{-}] \\ &\quad [H_{0,0,k_1, \ell_1}^{+} H_{0,1,k_1, \ell_1}^{+} H_{1,0,k_1, \ell_1}^{-} H_{1,1,k_1, \ell_1}^{-}] \\ &= X'_{0,2,k_1, \ell_1} H'_{0,2,k_1, \ell_1} \end{aligned}$$

$$\begin{aligned} m_{k_1, \ell_1, 1, 0} &= [X_{0,0,k_1, \ell_1}^{-} X_{0,1,k_1, \ell_1}^{-}] [H_{0,0,k_1, \ell_1}^{-} H_{0,1,k_1, \ell_1}^{-}] \\ &= X'_{1,0,k_1, \ell_1} H'_{1,0,k_1, \ell_1} \end{aligned}$$

$$m_{k_1, \ell_1, 1, 1} = [X_{0,0,k_1, \ell_1}^{-}] [H_{0,0,k_1, \ell_1}^{-}]$$

$$=X'_{1,1,k_1,\ell_1} H'_{1,1,k_1,\ell_1}$$

$$\begin{aligned} m_{k_1,\ell_1,1,2} &= [X_{0,0,k_1,\ell_1} + X_{0,1,k_1,\ell_1}] [H_{0,0,k_1,\ell_1} + H_{0,1,k_1,\ell_1}] \\ &= X'_{1,2,k_1,\ell_1} H'_{1,2,k_1,\ell_1} \end{aligned}$$

$$\begin{aligned} m_{k_1,\ell_1,2,0} &= [X_{0,0,k_1,\ell_1} - X_{0,1,k_1,\ell_1} + X_{1,0,k_1,\ell_1} - X_{1,1,k_1,\ell_1}] \\ &\quad \cdot [H_{0,0,k_1,\ell_1} - H_{0,1,k_1,\ell_1} + H_{1,0,k_1,\ell_1} - H_{1,1,k_1,\ell_1}] \\ &= X'_{2,0,k_1,\ell_1} H'_{2,0,k_1,\ell_1} \end{aligned}$$

$$\begin{aligned} m_{k_1,\ell_1,2,1} &= [X_{0,0,k_1,\ell_1} + X_{1,0,k_1,\ell_1}] [H_{0,0,k_1,\ell_1} + H_{1,0,k_1,\ell_1}] \\ &= X'_{2,1,k_1,\ell_1} H'_{2,1,k_1,\ell_1} \end{aligned}$$

$$\begin{aligned} m_{k_1,\ell_1,2,2} &= [X_{0,0,k_1,\ell_1} + X_{0,1,k_1,\ell_1} + X_{1,0,k_1,\ell_1} + X_{1,1,k_1,\ell_1}] \\ &\quad [H_{0,0,k_1,\ell_1} + H_{0,1,k_1,\ell_1} + H_{1,0,k_1,\ell_1} + H_{1,1,k_1,\ell_1}] \\ &= X'_{2,2,k_1,\ell_1} H'_{2,2,k_1,\ell_1} \end{aligned} \tag{3.35}$$

The interpolating polynomials $L_{i,j}(z,q)$ is obtained from eqn. 3.34

$$L_{0,0}(z,q) = 1/4 (z^2 q^2 - z^2 q - z q^2 + z q)$$

$$L_{0,1}(z,q) = -1/2 (z^2 q^2 - z^2 - z q^2 + z)$$

$$L_{0,2}(z,q) = 1/4 (z^2 q^2 + z^2 q - z q^2 - z q)$$

$$L_{1,0}(z,q) = -1/2 (z^2 q^2 - z^2 q - q^2 + q)$$

$$L_{1,1}(z,q) = (z^2 q^2 - z^2 - q^2 + 1)$$

$$L_{1,2}(z,q) = -1/2 (z^2 q^2 + z^2 q - q^2 - q)$$

$$L_{2,0}(z,q) = 1/4 (z^2 q^2 - z^2 q + z q^2 - z q)$$

$$L_{2,1}(z,q) = -1/2 (z^2 q^2 - z^2 + z q^2 - z)$$

$$L_{2,2}(z,q) = 1/4 (z^2 q^2 + z^2 q + z q^2 + z q)$$

Hence eqn. 3.32 becomes

$$\begin{bmatrix} W_{0,0,k_1,\ell_1} \\ W_{0,1,k_1,\ell_1} \\ W_{0,2,k_1,\ell_1} \\ W_{1,0,k_1,\ell_1} \\ W_{1,1,k_1,\ell_1} \\ W_{1,2,k_1,\ell_1} \\ W_{2,0,k_1,\ell_1} \\ W_{2,1,k_1,\ell_1} \\ W_{2,2,k_1,\ell_1} \end{bmatrix} = 1/4 \begin{bmatrix} 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & -4 & 2 & 0 & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 1 \\ -1 & 2 & -1 & 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 2 & 0 & 0 & -4 & 0 & 0 & 2 & 0 \\ -1 & 0 & 1 & 2 & 0 & -2 & -1 & 0 & 1 \\ 1 & -2 & 1 & -2 & 4 & -2 & 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} m_{k_1,\ell_1,0,0} \\ m_{k_1,\ell_1,0,1} \\ m_{k_1,\ell_1,0,2} \\ m_{k_1,\ell_1,1,0} \\ m_{k_1,\ell_1,1,1} \\ m_{k_1,\ell_1,1,2} \\ m_{k_1,\ell_1,2,0} \\ m_{k_1,\ell_1,2,1} \\ m_{k_1,\ell_1,2,2} \end{bmatrix}$$

Using eqn. 3.5, eqn. 3.31 can be written as

$$Y_{k_1, \ell_1}(z, q) = (W_{0,0,k_1, \ell_1} + W_{0,2,k_1, \ell_1} \beta^{\ell_1} + W_{2,0,k_1, \ell_1} \alpha^{k_1} + W_{2,2,k_1, \ell_1} \beta^{\ell_1} \alpha^{k_1}) + (W_{0,1,k_1, \ell_1} + W_{2,1,k_1, \ell_1} \alpha^{k_1})q + (W_{1,0,k_1, \ell_1} + W_{1,2,k_1, \ell_1} \beta^{\ell_1})z + (W_{1,1,k_1, \ell_1})zq$$

and

$$Y_{0,0,k_1, \ell_1} = \langle W_{0,0,k_1, \ell_1} + W_{0,2,k_1, \ell_1} \alpha^{\ell_1} + W_{2,0,k_1, \ell_1} \alpha^{k_1} + W_{2,2,k_1, \ell_1} \alpha^{\ell_1+k_1} \rangle_M$$

$$Y_{0,1,k_1, \ell_1} = \langle W_{0,1,k_1, \ell_1} + W_{2,1,k_1, \ell_1} \alpha^{k_1} \rangle_M$$

$$Y_{1,0,k_1, \ell_1} = \langle W_{1,0,k_1, \ell_1} + W_{1,2,k_1, \ell_1} \alpha^{\ell_1} \rangle_M$$

$$Y_{1,1,k_1, \ell_1} = \langle W_{1,1,k_1, \ell_1} \rangle_M$$

when $\beta = \alpha$

The result y_{n_2, r_2, n_1, r_1} can then be found by four length-128 inverse FNT.

$$y_{n_2, r_2, n_1, r_1} = \langle \frac{1}{128} \sum_{k_1=0}^{127} \sum_{\ell_1=0}^{127} Y_{n_2, r_2, k_1, \ell_1} \alpha^{-(n_1 k_1 + r_1 \ell_1)} \rangle_M$$

where $n_1, r_1 = 0, 1, \dots, 127$ and

$$n_2, r_2 = 0, 1$$

From eqn. 3.35, it is clear that the number of multiplications per

point is 2.25. This example is similar to the one given by Kraats and Venetsanopoulos [72], but our derivation is much more general and systematic. It is also interesting to point out that the calculation of $\{X'_{0,0,k_1,\ell_1}, X'_{0,2,k_1,\ell_1}, X'_{2,0,k_1,\ell_1}, X'_{2,2,k_1,\ell_1}\}$ actually corresponds to a length-4 Hadamard Transform and hence the Fast Hadamard Transform algorithm [73] can be applied to this computation, producing as intermediate results the required values $\{X'_{1,0,k_1,\ell_1}, X'_{1,2,k_1,\ell_1}\}$.

It should be noted that, as in the one-dimensional case, if N_2 is relatively large, a second two-dimensional NTT can be used instead of the two-dimensional Lagrange Interpolation Formula. If the same modulo base is also employed (that is $M'=M$) then N_1 is usually equal to $(2N_2-1)$. The number of multiplication per point is then always less than 4.

3.4 Nested Number Theoretic Transform

If the factors N_1 and N_2 of a given sequence length are relatively prime, then one may map the one-dimensional cyclic convolution into a two-dimensional cyclic convolution. The map can be written in a linear form as follows

$$n = K_1 n_1 + K_2 n_2 \pmod{N}$$

Burrus [68] shows that the map is cyclic in n_1 and n_2 if and only if $K_1 = CN_2$ and $K_2 = DN_1$. For a unique map, C and N_1 (also D and N_2) have to be relatively prime.

$$\text{Let } n = \langle N_2 n_1 + N_1 n_2 \rangle_N$$

where $n_1 = 0, \dots, N_1-1$ and

$$n_2 = 0, \dots, N_2-1$$

Hence eqn. 3.8 becomes

$$X(z, z_1) = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x_{n_2, n_1} z_1^{n_1} z^{n_2} \quad (3.36)$$

$$\text{mod}(z_1^{N_1}-1)$$

$$\text{mod}(z^{N_2}-1)$$

We can also make the same arrangement for $\tilde{h}(n)$. Eqns 3.15 and 3.16 remain the same as before. But eqn 3.17 becomes

$$X_{k_1}(z) H_{k_1}(z) = \left(\sum_{n_2=0}^{N_2-1} x_{n_2, k_1} z^{n_2} \right) \left(\sum_{m_2=0}^{N_2-1} h_{m_2, k_1} z^{m_2} \right) \quad (3.37)$$

$$\text{mod}(z^{N_2}-1)$$

Eqn. 3.37 is the cyclic convolution of the sequences $\{ x_{0, k_1}, x_{1, k_1}, \dots, x_{N_2-1, k_1} \}$ and $\{ h_{0, k_1}, h_{1, k_1}, \dots, h_{N_2-1, k_1} \}$. This cyclic convolution sum can be found by any fast cyclic convolution algorithm, for example the Winograd short convolution algorithms[25]. However, we suggest here that this can be done by a second Number Theoretic Transform. Since N_1 and N_2 are relatively prime, only one of the two NTTs may be calculated with a FNT, the other NTT can be a Mersenne Transform, Pseudo Mersenne Transform, Pseudo Fermat Transform or any NTT with an odd transform length. In Appendix A3.2 the same set of data as in Appendix A3.1 is used to

calculate the convolution sum of two sequences. It can be seen that no multiplications are required for the first NTT, and $N_2 \times N_1 (=N)$ multiplications are required for the computation of the second NTT. This method can easily be generalized to compute convolutions of multi-dimensional sequences. The number of multiplications remains as one per output point irrespective of the dimension of the sequence.

3.5 Conclusion

We have seen that the computation of the cyclic convolution sum of two length- N sequences using Number Theoretic Transform can always be done by two-dimensional (or multi-dimensional) techniques. The NTT can always be applied to the dimension with cyclic rearrangement. If the factors (N_1, N_2) of N are not mutually prime, the Lagrange Interpolation Formula or a second NTT can be used to find the convolution sum. This requires $(2 - \frac{1}{N_i})$ multiplications per point, where $i=1$ or 2 . However, if N_1 and N_2 are mutually prime, a nested NTT approach can be used to compute the cyclic convolution sum. This requires one multiplication per point. For two-dimensional convolutions, a two-dimensional Lagrange Interpolation Formula or a second NTT can be used in the computation to give less than 4 multiplications per point if N_1 and N_2 are not mutually prime. However if N_1 and N_2 are mutually prime, one multiplication per point is enough for the computation.

3.6 Appendices

Appendix A3.1 An Example of the Computation of Cyclic Convolutions using Multi-dimensional techniques

Consider the circular convolution of the sequences $\tilde{x}(n) = \{4, 0, 1, 2, 3, 1\}$ and $\tilde{h}(n) = \{4, 2, 5, 0, 0, 1\}$. Eqn. 3.10 can be written as

$$y = \sum_{n=0}^5 x_n h_{\ell-n}$$

where $\ell = 0, 1, \dots, 5$

A direct evaluation of this equation gives $y(n) = \{33, 14, 26, 13, 22, 24\}$.

To compute the cyclic convolution of these two sequences by eqns. 3.14 to 3.16, let $N_2=2$, $N_1=3$ and

$$n = n_2 + 2n_1$$

where $n_2 = 0, 1$ (non-cyclic)

$n_1 = 0, 1, 2$ (cyclic)

From eqn. 3.7,

$$z_1 = z^2$$

hence

$$x_{0,0} = x_0 = 4 \qquad h_{0,0} = h_0 = 4$$

$$x_{0,1} = x_2 = 1 \qquad h_{0,1} = h_2 = 5$$

$$x_{0,2} = x_4 = 3 \quad h_{0,2} = h_4 = 0$$

$$x_{1,0} = x_1 = 0 \quad h_{1,0} = h_1 = 2$$

$$x_{1,1} = x_3 = 2 \quad h_{1,1} = h_3 = 0$$

$$x_{1,2} = x_5 = 1 \quad h_{1,2} = h_5 = 1$$

Use a Pseudo Mersenne Transform, with

$$M = \frac{2^9 - 1}{2^3 - 1} = 73$$

$$\alpha = 2^3 \quad \text{and}$$

$$N_1 = 3$$

Eqn. 3.15 becomes

$$\begin{bmatrix} x_{n_2,0} \\ x_{n_2,1} \\ x_{n_2,2} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2^3 & 2^6 \\ 1 & 2^6 & 2^{12} \end{bmatrix} \begin{bmatrix} x_{n_2,0} \\ x_{n_2,1} \\ x_{n_2,2} \end{bmatrix} \pmod{73}$$

and

$$\begin{bmatrix} x_{0,0} \\ x_{0,1} \\ x_{0,2} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2^3 & 2^6 \\ 1 & 2^6 & 2^3 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 8 \\ 58 \\ 19 \end{bmatrix}$$

Similarly,

$$\begin{bmatrix} X_{1,0} \\ X_{1,1} \\ X_{1,2} \end{bmatrix} = \begin{bmatrix} 3 \\ 7 \\ 63 \end{bmatrix}, \quad \begin{bmatrix} H_{0,0} \\ H_{0,1} \\ H_{0,2} \end{bmatrix} = \begin{bmatrix} 9 \\ 44 \\ 32 \end{bmatrix}, \quad \begin{bmatrix} H_{1,0} \\ H_{1,1} \\ H_{1,2} \end{bmatrix} = \begin{bmatrix} 3 \\ 66 \\ 10 \end{bmatrix}$$

Eqn. 3.17 becomes

$$\begin{aligned} X_{k_1}(z)H_{k_1}(z) &= \left(\sum_{n_2=0}^1 X_{n_2,k_1} z^{n_2} \right) \left(\sum_{m_2=0}^1 H_{m_2,k_1} z^{m_2} \right) \\ &= W_{0,k_1} + W_{1,k_1} z + W_{2,k_1} z^2 \\ &= (W_{0,k_1} + z W_{2,k_1}) + W_{1,k_1} z \end{aligned}$$

By eqn. 3.5

$$X_{k_1}(z)H_{k_1}(z) = (W_{0,k_1} + 2^{3k_1} W_{2,k_1}) + W_{1,k_1} z \quad (\text{A3.1})$$

(i) By Lagrange Interpolating Formula

Note that the procedures are mainly used to illustrate the steps of the implementation and no attempt has been made to optimize the number of additions. Let $z_i = -1, 0$ and 1 for $i=0,1,2$ in eqn. 3.20. Hence

$$\begin{aligned} m_{k_1,0} &= (X_{0,k_1} - X_{1,k_1})(H_{0,k_1} - H_{1,k_1}) \\ m_{k_1,1} &= (X_{0,k_1})(H_{0,k_1}) \\ m_{k_1,2} &= (X_{0,k_1} + X_{1,k_1})(H_{0,k_1} + H_{1,k_1}) \end{aligned}$$

and

$$\begin{bmatrix} m_{0,0} \\ m_{0,1} \\ m_{0,2} \end{bmatrix} = \begin{bmatrix} (8-3)(9-3) \\ 8 \times 9 \\ (8+3)(9+3) \end{bmatrix} = \begin{bmatrix} 30 \\ 72 \\ 132 \end{bmatrix}$$

Similarly,

$$\begin{bmatrix} m_{1,0} \\ m_{1,1} \\ m_{1,2} \end{bmatrix} = \begin{bmatrix} -1122 \\ 2552 \\ 7150 \end{bmatrix}, \quad \begin{bmatrix} m_{2,0} \\ m_{2,1} \\ m_{2,2} \end{bmatrix} = \begin{bmatrix} -968 \\ 608 \\ 3444 \end{bmatrix}$$

The interpolating polynomials become (by eqn. 3.19)

$$L_0(z) = \left(\frac{z-0}{-1-0} \right) \left(\frac{z-1}{-1-1} \right) = \frac{1}{2}(z^2 - z)$$

Similarly,

$$L_1(z) = -(z^2 - 1) \quad \text{and}$$

$$L_2(z) = 1/2(z^2 + z)$$

Hence, by eqn. 3.18

$$\begin{bmatrix} w_{0,k_1} \\ w_{1,k_1} \\ w_{2,k_1} \end{bmatrix} = 1/2 \begin{bmatrix} 0 & 2 & 0 \\ -1 & 0 & 1 \\ 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} m_{k_1,0} \\ m_{k_1,1} \\ m_{k_1,2} \end{bmatrix}$$

This gives

$$\begin{bmatrix} w_{0,0} \\ w_{1,0} \\ w_{2,0} \end{bmatrix} = \begin{bmatrix} 72 \\ 51 \\ 9 \end{bmatrix}, \quad \begin{bmatrix} w_{0,1} \\ w_{1,1} \\ w_{2,1} \end{bmatrix} = \begin{bmatrix} 2552 \\ 4136 \\ 462 \end{bmatrix}, \quad \begin{bmatrix} w_{0,2} \\ w_{1,2} \\ w_{2,2} \end{bmatrix} = \begin{bmatrix} 608 \\ 2206 \\ 630 \end{bmatrix}$$

and

$$\begin{bmatrix} w_{0,0} \\ w_{1,0} \\ w_{2,0} \end{bmatrix} = \begin{bmatrix} 72 \\ 51 \\ 9 \end{bmatrix}, \quad \begin{bmatrix} w_{0,1} \\ w_{1,1} \\ w_{2,1} \end{bmatrix} = \begin{bmatrix} 70 \\ 48 \\ 24 \end{bmatrix}, \quad \begin{bmatrix} w_{0,2} \\ w_{1,2} \\ w_{2,2} \end{bmatrix} = \begin{bmatrix} 24 \\ 16 \\ 46 \end{bmatrix} \pmod{73}$$

(ii) By a Second Number Theoretic Transform

Let us use the same M , α and N for the second NTT.

$$\begin{bmatrix} x'_{0,k_1} \\ x'_{1,k_1} \\ x'_{2,k_1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2^3 & 2^6 \\ 1 & 2^6 & 2^3 \end{bmatrix} \begin{bmatrix} x_{0,k_1} \\ x_{1,k_1} \\ 0 \end{bmatrix} \pmod{73}$$

$$= \begin{bmatrix} 1 & 1 \\ 1 & 2^3 \\ 1 & 2^6 \end{bmatrix} \begin{bmatrix} x_{0,k_1} \\ x_{1,k_1} \end{bmatrix} \pmod{73}$$

Hence

$$\begin{bmatrix} x'_{0,0} \\ x'_{1,0} \\ x'_{2,0} \end{bmatrix} = \begin{bmatrix} 11 \\ 32 \\ 54 \end{bmatrix}, \quad \begin{bmatrix} x'_{0,1} \\ x'_{1,1} \\ x'_{2,1} \end{bmatrix} = \begin{bmatrix} 65 \\ 41 \\ 68 \end{bmatrix}, \quad \begin{bmatrix} x'_{0,2} \\ x'_{1,2} \\ x'_{2,2} \end{bmatrix} = \begin{bmatrix} 9 \\ 12 \\ 36 \end{bmatrix}$$

Similarly,

$$\begin{bmatrix} h'_{0,0} \\ h'_{1,0} \\ h'_{2,0} \end{bmatrix} = \begin{bmatrix} 12 \\ 33 \\ 55 \end{bmatrix}, \quad \begin{bmatrix} h'_{0,1} \\ h'_{1,1} \\ h'_{2,1} \end{bmatrix} = \begin{bmatrix} 37 \\ 61 \\ 34 \end{bmatrix}, \quad \begin{bmatrix} h'_{0,2} \\ h'_{1,2} \\ h'_{2,2} \end{bmatrix} = \begin{bmatrix} 42 \\ 39 \\ 15 \end{bmatrix}$$

Let $w'_{k'_2, k_1} = x'_{k'_2, k_1} h'_{k'_2, k_1} \bmod 73$.

Therefore

$$\begin{bmatrix} w'_{0,0} \\ w'_{1,0} \\ w'_{2,0} \end{bmatrix} = \begin{bmatrix} 59 \\ 34 \\ 50 \end{bmatrix}, \quad \begin{bmatrix} w'_{0,1} \\ w'_{1,1} \\ w'_{2,1} \end{bmatrix} = \begin{bmatrix} 69 \\ 19 \\ 49 \end{bmatrix}, \quad \begin{bmatrix} w'_{0,2} \\ w'_{1,2} \\ w'_{2,2} \end{bmatrix} = \begin{bmatrix} 13 \\ 30 \\ 29 \end{bmatrix}$$

and

$$\begin{bmatrix} w_{0,k_1} \\ w_{1,k_1} \\ w_{2,k_1} \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2^{-3} & 2^{-6} \\ 1 & 2^{-6} & 2^{-12} \end{bmatrix} \begin{bmatrix} w'_{0,k_1} \\ w'_{1,k_1} \\ w'_{2,k_1} \end{bmatrix} \bmod 73$$

Hence

$$\begin{bmatrix} w_{0,0} \\ w_{1,0} \\ w_{2,0} \end{bmatrix} = \begin{bmatrix} 72 \\ 51 \\ 9 \end{bmatrix}, \quad \begin{bmatrix} w_{0,1} \\ w_{1,1} \\ w_{2,1} \end{bmatrix} = \begin{bmatrix} 70 \\ 48 \\ 24 \end{bmatrix}, \quad \begin{bmatrix} w_{0,2} \\ w_{1,2} \\ w_{2,2} \end{bmatrix} = \begin{bmatrix} 24 \\ 16 \\ 46 \end{bmatrix}$$

These results are exactly the same as what we obtained in (i).

Eqn. A3.1 can now be written as

$$Y_{0,k_1} = \langle w_{0,k_1} + 2^{3k_1} w_{2,k_1} \rangle_{73}$$

$$Y_{1,k_1} = \langle w_{1,k_1} \rangle_{73}$$

This gives us

$$Y_{0,0} = 8$$

$$Y_{1,0} = 51$$

$$Y_{0,1} = 43$$

$$Y_{1,1} = 48$$

$$Y_{0,2} = 48 \quad \text{and}$$

$$Y_{1,2} = 16$$

The final results can be obtained by

$$\begin{bmatrix} y_{n_2,0} \\ y_{n_2,1} \\ y_{n_2,2} \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2^{-3} & 2^{-6} \\ 1 & 2^{-6} & 2^{-12} \end{bmatrix} \begin{bmatrix} y_{n_2,0} \\ y_{n_2,1} \\ y_{n_2,2} \end{bmatrix} \pmod{73}$$

where $n_2 = 0, 1$

Hence

$$\begin{bmatrix} y_{0,0} \\ y_{0,1} \\ y_{0,2} \end{bmatrix} = \begin{bmatrix} 33 \\ 26 \\ 22 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} y_{1,0} \\ y_{1,1} \\ y_{1,2} \end{bmatrix} = \begin{bmatrix} 14 \\ 13 \\ 24 \end{bmatrix}$$

The results become

$$y(n) = \{ 33, 14, 26, 13, 22, 24 \}$$

Appendix A3.2 An Example of the Nested NTT

Let us now find the convolution sum of the sequences $x(n) = \{4, 0, 1, 2, 3, 1\}$ and $h(n) = \{4, 2, 5, 0, 0, 1\}$ of Appendix A3.1 by a nested NTT. Let $N_2=2$ and $N_1=3$ and

$$n = \langle 3n_2 + 2n_1 \rangle_6$$

where $n_2 = 0, 1$ and

$$n_1 = 0, 1, 2$$

Hence

$$x_{0,0} = x(0) = 4 \quad h_{0,0} = h(0) = 4$$

$$x_{0,1} = x(2) = 1 \quad h_{0,1} = h(2) = 5$$

$$x_{0,2} = x(4) = 3 \quad h_{0,2} = h(4) = 0$$

$$x_{1,0} = x(3) = 2 \quad h_{1,0} = h(3) = 0$$

$$x_{1,1} = x(5) = 1 \quad h_{1,1} = h(5) = 1$$

$$x_{1,2} = x(1) = 0 \quad h_{1,2} = h(1) = 2$$

Again, let us use a Pseudo Mersenne Transform with $M = (2^9 - 1)/(2^3 - 1)$, $\alpha = 2^3$ and $N_1 = 3$ for the first NTT. This gives

$$\begin{bmatrix} x_{0,0} \\ x_{0,1} \\ x_{0,2} \end{bmatrix} = \begin{bmatrix} 8 \\ 58 \\ 19 \end{bmatrix}, \quad \begin{bmatrix} x_{1,0} \\ x_{1,1} \\ x_{1,2} \end{bmatrix} = \begin{bmatrix} 3 \\ 10 \\ 66 \end{bmatrix}, \quad \begin{bmatrix} h_{0,0} \\ h_{0,1} \\ h_{0,2} \end{bmatrix} = \begin{bmatrix} 9 \\ 44 \\ 32 \end{bmatrix}, \quad \begin{bmatrix} h_{1,0} \\ h_{1,1} \\ h_{1,2} \end{bmatrix} = \begin{bmatrix} 3 \\ 63 \\ 7 \end{bmatrix}$$

Since $N_2 = 2$, $(z^2 - 1) = 0$ and eqn. 3.37 becomes

$$\begin{aligned} Y_{k_1}(z) &= H_{k_1}(z)X_{k_1}(z) \\ &= (x_{0,k_1}h_{0,k_1} + x_{1,k_1}h_{1,k_1}) + (x_{0,k_1}h_{1,k_1} + x_{1,k_1}h_{0,k_1})z \\ &= Y_{0,k_1} + Y_{1,k_1}z \end{aligned}$$

where $k_1 = 0, 1, 2$

The second transformation can be done by a FNT with $M = F_4 = 2^{16} + 1$, $N_2 = 2$ and $\alpha = 2^{16}$.

$$\begin{bmatrix} X'_{0,k_1} \\ X'_{1,k_1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2^{16} \end{bmatrix} \begin{bmatrix} X_{0,k_1} \\ X_{1,k_1} \end{bmatrix} \pmod{2^{16+1}}$$

This gives

$$\begin{bmatrix} X'_{0,0} \\ X'_{1,0} \end{bmatrix} = \begin{bmatrix} 11 \\ 5 \end{bmatrix}, \quad \begin{bmatrix} X'_{0,1} \\ X'_{1,1} \end{bmatrix} = \begin{bmatrix} 68 \\ 48 \end{bmatrix}, \quad \begin{bmatrix} X'_{0,2} \\ X'_{1,2} \end{bmatrix} = \begin{bmatrix} 85 \\ 65490 \end{bmatrix}$$

Similarly,

$$\begin{bmatrix} H'_{0,0} \\ H'_{1,0} \end{bmatrix} = \begin{bmatrix} 12 \\ 6 \end{bmatrix}, \quad \begin{bmatrix} H'_{0,1} \\ H'_{1,1} \end{bmatrix} = \begin{bmatrix} 107 \\ 65518 \end{bmatrix}, \quad \begin{bmatrix} H'_{0,2} \\ H'_{1,2} \end{bmatrix} = \begin{bmatrix} 39 \\ 25 \end{bmatrix}$$

Since

$$Y'_{k'_2, k_1} = H'_{k'_2, k_1} X'_{k'_2, k_1} \pmod{2^{16+1}}$$

(A3.2)

where $k'_2 = 0, 1$

$k_1 = 0, 1, 2$

$$Y'_{0,0} = 132 \quad Y'_{1,0} = 30$$

$$Y'_{0,1} = 7276 \quad Y'_{1,1} = 64625$$

$$Y'_{0,2} = 3315 \quad Y'_{1,2} = 64362$$

The inverse transform is

$$\begin{bmatrix} Y_{0,k_1} \\ Y_{1,k_1} \end{bmatrix} = \ll 1/2 \begin{bmatrix} 1 & 1 \\ 1 & 2^{-16} \end{bmatrix} \begin{bmatrix} Y'_{0,k_1} \\ Y'_{1,k_1} \end{bmatrix} >_{(2^{16}+1)^{73}}$$

Hence

$$\begin{bmatrix} Y_{0,0} \\ Y_{1,0} \end{bmatrix} = \begin{bmatrix} 8 \\ 51 \end{bmatrix}, \quad \begin{bmatrix} Y_{0,1} \\ Y_{1,1} \end{bmatrix} = \begin{bmatrix} 43 \\ 6 \end{bmatrix}, \quad \begin{bmatrix} Y_{0,2} \\ Y_{1,2} \end{bmatrix} = \begin{bmatrix} 48 \\ 55 \end{bmatrix}$$

Making the inverse transform of base 73, we get

$$y_{0,0} = 33 \quad y_{1,0} = 13$$

$$y_{0,1} = 26 \quad y_{1,1} = 24$$

$$y_{0,2} = 22 \quad y_{1,2} = 14$$

This gives the same set of results as given in Appendix A3.1. As shown in eqn. A3.2, the total number of multiplication is 6. In general the nested Number Theoretic Transform requires only one multiplication per output.

CHAPTER FOUR APPROACH TO THE HARDWARE IMPLEMENTATION OF DIGITAL SIGNAL PROCESSORS USING MERSENNE NUMBER TRANSFORMS

Since the transform lengths of the Mersenne Number Transforms are not highly composite, there is no FFT-type algorithm for their computation and for this reason little attention has been given to them. However Mersenne Number Transforms have the advantage of very efficient arithmetic (which could be in ones-complement form). For example, the required multiplications by powers of 2 are performed by simple bit rotations.

Hitherto, the main concern has been to look for highly composite transform lengths and most of the implementations rely heavily on FFT-type algorithms [43-51,74]. These FFT-type algorithms, no doubt, significantly reduce the number of operations in the computation of Number Theoretic Transforms. However, because of recent advances in fabrication technology of electronic devices, arithmetic units are no longer the limiting factor. The control section of the realisation is probably the largest and the most expensive part. If FFT-type algorithms are used for the implementation of Number Theoretic Transforms, the control of the process flow is unavoidably more complicated. As shown on the timing diagram of paper [44] not less than half of the total time spent on the FNT butterfly is used only for "read" and "write" operations on the data.

In this chapter we convert the Mersenne Number Transform into

a cyclic convolution form and propose simple hardware structures to carry out their fast implementation. Such realisation structures are proposed that can make the computation of Mersenne Number Transforms very fast indeed. This new approach can be extended to other Number Theoretic Transforms, for instance to Fermat Number Transforms.

The contents of this chapter have been published in [75].

4.1 Mersenne Number Transform and Cyclic Convolution

Recall the Mersenne Number Transform and its inverse:

$$X(k) = \left\langle \sum_{n=0}^{P-1} x(n) \alpha^{nk} \right\rangle_M \quad (4.1)$$

$$x(n) = \left\langle \frac{1}{P} \sum_{k=0}^{P-1} X(k) \alpha^{-nk} \right\rangle_M \quad (4.2)$$

where $M=2^P-1$, a Mersenne number

α =root of unity of order P and is

equal to 2 for this chapter

Eqns. 4.1 and 4.2 can be defined for both prime or composite Mersenne numbers.

Since the transform length P is a prime number, it is possible to find a primitive root, g , to generate all nonzero elements inside the field modulo P . Using the mappings of $k \rightarrow \langle g^k \rangle_P$ and $n \rightarrow \langle g^{-n} \rangle_P$, eqn. 1 can be reordered as shown below

$$X(0) = \sum_{n=0}^{P-1} x(n) \quad \text{and} \quad (4.3)$$

$$X(\langle g^k \rangle_P) = x(0) + \sum_{n=1}^{P-1} x(\langle g^{-n} \rangle_P) \alpha^{\langle g^{k-n} \rangle_P} \quad (4.4)$$

for $k=1,2,\dots,P-1$

We can write eqn. 4.4 as

$$X(\langle g^k \rangle_P) = x(0) + X'(\langle g^k \rangle_P) \quad (4.5)$$

$$\text{where } X'(\langle g^k \rangle_P) = \sum_{n=1}^{P-1} x(\langle g^{-n} \rangle_P) \alpha^{\langle g^{k-n} \rangle_P} \quad (4.6)$$

for $k=1,2,\dots,P-1$

Eqn. 4.6 represents a backward circular convolution of length $(P-1)$.

That is the convolution

$$\{x(g^{-1}), x(g^{-2}), \dots, x(g^{-P+1})\} \circledast \{\alpha^g^0, \alpha^g^1, \dots, \alpha^g^{P-2}\} \quad (4.7)$$

where $\circledast$ means circular convolution and the indices and subscripts are modulo P

In terms of matrix notation, we have

$$\begin{bmatrix} X'(\langle g^1 \rangle_P) \\ X'(\langle g^2 \rangle_P) \\ \dots \\ X'(\langle g^{P-1} \rangle_P) \end{bmatrix} = \begin{bmatrix} \langle g^{1-1} \rangle_P & \langle g^{1-2} \rangle_P & \dots & \langle g^{2-P} \rangle_P \\ \alpha \langle g^{2-1} \rangle_P & \alpha \langle g^{2-2} \rangle_P & \dots & \alpha \langle g^{3-P} \rangle_P \\ \dots & \dots & \dots & \dots \\ \alpha \langle g^{P-2} \rangle_P & \alpha \langle g^{P-3} \rangle_P & \dots & \alpha \langle g^0 \rangle_P \end{bmatrix} \begin{bmatrix} x(\langle g^{-1} \rangle_P) \\ x(\langle g^{-2} \rangle_P) \\ \dots \\ x(\langle g^{-P+1} \rangle_P) \end{bmatrix} \quad (4.8)$$

By making the substitution of $-t = k-n$, eqn. 4.6 can be written as

$$X'(\langle g^k \rangle_P) = \sum_{t=1}^{P-1} x(\langle g^{-(k+t)} \rangle_P) \alpha^{\langle g^{-t} \rangle_P} \quad (4.9)$$

which becomes, for MNT coefficients

$$\begin{bmatrix} X'(\langle g^1 \rangle_P) \\ X'(\langle g^2 \rangle_P) \\ \dots \\ X'(\langle g^{P-1} \rangle_P) \end{bmatrix} = \begin{bmatrix} x(\langle g^{-1-1} \rangle_P) & x(\langle g^{-1-2} \rangle_P) & \dots & x(\langle g^{-P} \rangle_P) \\ x(\langle g^{-2-1} \rangle_P) & x(\langle g^{-2-2} \rangle_P) & \dots & x(\langle g^{-P-1} \rangle_P) \\ \dots & \dots & \dots & \dots \\ x(\langle g^{-P} \rangle_P) & x(\langle g^{-P-1} \rangle_P) & \dots & x(\langle g^{-2P+2} \rangle_P) \end{bmatrix} \begin{bmatrix} \alpha^{\langle g^{-1} \rangle_P} \\ \alpha^{\langle g^{-2} \rangle_P} \\ \dots \\ \alpha^{\langle g^{-P+1} \rangle_P} \end{bmatrix} \quad (4.10)$$

Note that in this form the square matrix is circulant.

A simple example will illustrate the above steps. Consider the sequence $\{x(0), x(1), x(2), x(3), x(4)\}$ where $P=5$. In this case 2 is a primitive root which can be used to generate the non-zero elements inside the field modulo 5. Hence the sequence in eqn. 4.7 can be written as

$$\{x_n : n=1, 2, 3, 4\} = \{x(\langle g^{-n} \rangle_P) : n=1, 2, 3, 4\}$$

$$= \{x(3), x(4), x(2), x(1)\}$$

and

$$\{\alpha^{g^n} : n=0,1,2,3\} = \{\alpha^1, \alpha^2, \alpha^4, \alpha^3\}$$

This implies that the convolution is now written

$$\{x(3), x(4), x(2), x(1)\} \otimes \{\alpha^1, \alpha^2, \alpha^4, \alpha^3\}$$

As stated earlier, we restrict $\alpha=2$, hence eqns. 4.8 and 4.10 become

$$\begin{bmatrix} X'(2) \\ X'(4) \\ X'(3) \\ X'(1) \end{bmatrix} = \begin{bmatrix} 2^1 & 2^3 & 2^4 & 2^2 \\ 2^2 & 2^1 & 2^3 & 2^4 \\ 2^4 & 2^2 & 2^1 & 2^3 \\ 2^3 & 2^4 & 2^2 & 2^1 \end{bmatrix} \begin{bmatrix} x(3) \\ x(4) \\ x(2) \\ x(1) \end{bmatrix}$$

and

$$\begin{bmatrix} X'(2) \\ X'(4) \\ X'(3) \\ X'(1) \end{bmatrix} = \begin{bmatrix} x(4) & x(2) & x(1) & x(3) \\ x(2) & x(1) & x(3) & x(4) \\ x(1) & x(3) & x(4) & x(2) \\ x(3) & x(4) & x(2) & x(1) \end{bmatrix} \begin{bmatrix} 2^3 \\ 2^4 \\ 2^2 \\ 2^1 \end{bmatrix} \quad (4.11)$$

It is evident from the above equation that the square data matrix is circulant.

Now let us consider the convolution of two sequences, $\{x(n): n=0,1,\dots,P-1\}$ and $\{h(n): n=0,1,\dots,P-1\}$,

$$y(m) = \sum_{n=0}^{P-1} x(n)h(m-n) \quad (4.12)$$

for $m=0,1,\dots,P-1$

For the computation of eqn. 4.12 using MNT, $y(m)$ can be found from the Inverse Mersenne Number Transform (IMNT),

$$y(m) = \frac{1}{P} \sum_{k=0}^{P-1} Y(k) \alpha^{-mk} \quad (4.13)$$

for $m=0,1,\dots,P-1$

The coefficients $Y(k)$ are the products of the term by term multiplications of corresponding $X(k)$ and $H(k)$, and $H(k)$ is the MNT of the sequence $\{h(n):n=0,1,\dots,P-1\}$.

Now, let us apply the following mapping to eqn. 4.13

$$k \rightarrow \langle g^k \rangle_P$$

$$m \rightarrow \langle -g^{-m} \rangle_P$$

from which we obtain,

$$\begin{aligned} y(0) &= \frac{1}{P} \sum_{k=0}^{P-1} Y(k) \\ y(\langle -g^{-m} \rangle_P) &= \frac{1}{P} \left[Y(0) + \sum_{k=1}^{P-1} Y(\langle g^k \rangle_P) \alpha^{\langle g^{k-m} \rangle_P} \right] \end{aligned} \quad (4.14)$$

or

$$y'(\langle -g^{-m} \rangle_P) = \frac{1}{P} \sum_{k=1}^{P-1} Y(\langle g^k \rangle_P) \alpha^{\langle g^{k-m} \rangle_P} \quad (4.15)$$

for $m=1,2,\dots,P-1$

$$\begin{bmatrix} y'(\langle -g^{-1} \rangle_P) \\ y'(\langle -g^{-2} \rangle_P) \\ \dots \\ \dots \\ y'(\langle -g^{-P+1} \rangle_P) \end{bmatrix} = \begin{bmatrix} \alpha \langle g^{1-1} \rangle_P & \alpha \langle g^{2-1} \rangle_P & \dots & \alpha \langle g^{P-2} \rangle_P \\ \alpha \langle g^{1-2} \rangle_P & \alpha \langle g^{2-2} \rangle_P & \dots & \alpha \langle g^{P-3} \rangle_P \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \alpha \langle g^{2-P} \rangle_P & \alpha \langle g^{3-P} \rangle_P & \dots & \alpha \langle g^0 \rangle_P \end{bmatrix} \begin{bmatrix} Y(\langle g^1 \rangle_P) \\ Y(\langle g^2 \rangle_P) \\ \dots \\ \dots \\ Y(\langle g^{P-1} \rangle_P) \end{bmatrix} \quad (4.16)$$

If $P=5$, we obtain

$$\begin{bmatrix} y'(2) \\ y'(1) \\ y'(3) \\ y'(4) \end{bmatrix} = \begin{bmatrix} 2^1 & 2^2 & 2^4 & 2^3 \\ 2^3 & 2^1 & 2^2 & 2^4 \\ 2^4 & 2^3 & 2^1 & 2^2 \\ 2^2 & 2^4 & 2^3 & 2^1 \end{bmatrix} \begin{bmatrix} Y(2) \\ Y(4) \\ Y(3) \\ Y(1) \end{bmatrix}$$

Eqns. 4.8, 4.10 and 4.11 and 4.16 are the four essential expressions that are used for the hardware design in the next section.

4.2 Hardware Description

Our hardware design for the computation of Mersenne Number Transform and Inverse Mersenne Number Transform relies mainly on the fact that the basic equations of the MNT and IMNT can be converted into convolution forms. Fig. 4.1 shows the register arrangement of a 5-point Mersenne Number Transform unit. The registers used in this circuit are D-type registers (or Flip-flops) with tri-state outputs and data enable input controls such that all inputs and outputs can be

tied to the same bus.

The multiplication of a power of 2 in MNT arithmetic simply involves a circular left shift of the data. However, circular shifts may still be time consuming for very fast implementations of digital signal processors. In our design, a circular shift is done at the same time at which the data is added from registers into the accumulator (not shown on Fig. 4.1). For example, if register 3 is activated (by output control lines) to send data out to be added into the accumulator, the input data enable line of register 2 should also be activated. The contents in registers 3 and 2 should be equal to the original data multiplied by 2^2 and 2^4 respectively. The circular shift of two bits ($2^{4-2} = 2^2$) when the contents in register 3 are loaded into register 2 is done simply by rearranging (shifting circularly) the pin connection of register 2 as shown in Fig. 4.1. This shows that the shifting time is entirely saved.

Suppose that the contents in registers 1, 2, 3 and 4 are $x(4)$, $x(2)$, $x(1)$ and $x(3)$ respectively and the accumulator has been loaded with $x(0)$ initially. The addition of the contents from register 1 to the accumulator will transfer $x(4)$ to the buffer, the addition of the contents from register 2 will transfer $x(2)$ to register 1 and so on. Hence after 4 additions, $X(2)$ will be stored inside the accumulator. Exactly the same procedure (addition starting from register 1, register 2, ..., to register 4) is required for finding other $X(k)$'s by virtue of the circulant form of the data matrix. The correctness of the operation may be seen by considering

eqn. 4.11. The total number of additions for the whole transform is P^2 for a P-point MNT, if the initial loading of $x(0)$ to the accumulator is taken to be equivalent to one addition. Notice that the control of data flow is very simple and no scrambling of data is needed at any intermediate step of the transformation.

The initial setting of the registers can be done by loading registers 1,2,3 and 4 with $x(4)$, $x(2)$, $x(1)$ and $x(3)$ respectively and then perform one self read-write (load and store from the same register) operation to each register individually.

Fig. 4.2 shows a pipeline arrangement of a 5-point convolver using Mersenne Number Transform. The upper register set and adder 1 are used for Mersenne Number Transforms. The lower register set and adder 3 are used for Inverse Mersenne Number Transforms. Accumulator 2, register B and adder 2 are used for the computation of the product of $H(k)$ and $X(k)$ by a shift-add technique. After the first $X(k)$ has been obtained in accumulator 1, it is shifted down into shift register A which is used to control the addition of $H(k)$ into accumulator 2. If there is a "one" at the rightmost bit of the shift register A, $H(k)$ is added into accumulator 2 and then shifted to the left circularly. If there is a zero at the right most bit of the shift register A, accumulator 2 shifts to the left circularly without addition. These shift-add operations are done simultaneously with the MNT unit.

Similarly, immediately after the first product of $X(k)H(k)$ has been obtained the Inverse Mersenne Number Transform can be

started. The circuit for the IMNT is mainly constructed according to eqn. 4.16. Instead of rotating the input words as shown in the MNT unit, the result registers (of $y'(m)$'s) are rotated to the right after each addition. For example, when the contents of register C3 is added to register C, the sum will be entered into register C2. Since rotations are done on the result registers, the multiplications by 2^m of the input data is implemented by the multiplication by 2^{P-m} in the result registers. Similarly to the MNT, the multiplication by powers of two is done simply by hardware. The initial setting of the result registers can be done by one self read-write operation to each register individually. After the last $X(k)$ has been calculated, registers A1 to A4 are released from their original use. At the same time the IMNT unit should also be at the last cycle of its operation. The data in register C are added to the contents of registers C1 to C4, the results can then be entered into register A1 to A4 in such a way that non-rotated results are obtained.

Hence it is seen that this pipeline convolver requires approximately $P^2 T$ ns for the computation of a P-point cyclic convolution using the Mersenne Number Transform, where T is the time for one addition. Using 17-bit adders similar to those used in Ref.[44] for which the time for one addition is 21ns, the total convolution time is approximately $17^2 \times 21$ ns or $7 \mu s$ for a 17-point convolution. Notice that the whole design is very simple. It involves only a very simple control unit, 3 adders, 41 registers or accumulators, 17 words of ROM or RAM and some buffers.

4.3 Speed Optimized Convolver Using Mersenne Number Transform

In the previous section, the Mersenne Number Transform Convolver has been built on a pipeline and serial data movement basis. Compared with multipliers, adders are much less expensive and so we may use more adders to construct a parallel processor to speed up the computation. Fig. 4.3 shows a 17-point Fast Mersenne Number Convolver using parallel adders. The principle of operation is very similar to that described in the previous section. Register A holds the input data which is added into register B1 to B17 in parallel. The results of each parallel addition are stored in the right hand side registers respectively. This arrangement is equivalent to the shifting of these sum registers to line up properly for the next addition. After the last $x(n)$ has been added, the sums are shifted down to registers C1 to C17 and then sent serially to the multiplier for the multiplication of $H(m)$'s. After the first product has been obtained, it is sent to register A' to compute the Inverse Mersenne Number Transform. Comparing eqn. 4.8 with eqn. 4.16, it is clear that the IMNT and MNT have essentially the same structure. The only difference is that the directions of data flow (or equivalently α^n flow) are opposite. Hence for the IMNT, the results of each parallel addition are stored in the "left" hand side registers (not shown on the diagram) respectively.

The convolution time for this design is approximately equal to the time for 17 additions and 17 multiplications. A fast Mersenne Number multiplier can be constructed by (i) modifying a commercially

available multiplier for MNT arithmetic, or (ii) using MSI or LSI to construct a dedicated MNT multiplier [76,77], or indeed (iii) fabricating a dedicated I.C. for this purpose. Fig. 4.4 shows a possible construction of a 17-point MNT multiplier using the quarter-square multiplier,

$$X_m H_m = \frac{1}{4} [(X_m + H_m)^2 - (X_m - H_m)^2]$$

This multiplier requires 86 ns for each multiplication (no attempt has been made to optimize the speed of the design). Hence the total execution time for a 17-point convolution for using this multiplier is $17 \times (86 + 21) \approx 1.8 \mu s$. Notice that the execution time of the present design depends mainly on the speed of the multiplier and adders.

4.4 Long Convolution from Short Convolution

Possible word-lengths for the Mersenne Number Transform implementation of convolvers using the present techniques are $\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67\}$. These are also the possible sequence lengths for the MNT's. However, the sequence lengths of the MNT can be extended to $2P$ when -2 is chosen as the root of unity. In this case

$$X(k) = \left\langle \sum_{n=0}^{2P-1} x(n) \alpha^{nk} \right\rangle_M \quad (4.17)$$

where $\alpha = -2$ and

$$M = 2^P - 1$$

for $k = 0, 1, \dots, 2P-1$

Eqn. 4.17 can be mapped into a two-dimensional form [68] through the following mapping

$$n = \langle Pn_2 + 2n_1 \rangle_{2P} \quad (4.18)$$

$$k = \langle P \cdot P^{-1}k_2 + 2 \cdot 2^{-1}k_1 \rangle_{2P} \quad (4.19)$$

$$\begin{aligned} \text{where } PP^{-1} &= 1 \pmod{2} \\ &= 0 \pmod{P} \\ 2 \cdot 2^{-1} &= 1 \pmod{P} \\ &= 0 \pmod{2} \end{aligned}$$

$$\begin{aligned} \text{for } n_1, k_1 &= 0, 1, \dots, P-1 \\ n_2, k_2 &= 0, 1 \end{aligned}$$

Hence the two-dimensional form becomes

$$x_{k_2, k_1} = \sum_{n_2=0}^{2-1} \sum_{n_1=0}^{P-1} x_{n_2, n_1} \alpha_1^{n_1 k_1} \alpha_2^{n_2 k_2} \quad (4.20)$$

$$\text{where } \alpha_1 = \alpha^2 \text{ and } \alpha_2 = \alpha^P$$

$$x_{n_2, n_1} = x(\langle Pn_2 + 2n_1 \rangle_{2P})$$

$$x_{k_2, k_1} = X(\langle P \cdot P^{-1}k_2 + 2 \cdot 2^{-1}k_1 \rangle_{2P})$$

Rewrite eqn. 4.20 into vector form

$$\bar{x}_{k_2} = \sum_{n_2=0}^1 (\bar{A} \bar{x}_{n_2}) \alpha_2^{n_2 k_2} \quad (4.21)$$

for $k_2=0,1$

where

$$\bar{x}_{k_2} = \begin{bmatrix} x_{k_2,0} \\ x_{k_2,1} \\ \dots \\ x_{k_2,P-1} \end{bmatrix}, \quad \bar{x}_{n_2} = \begin{bmatrix} x_{n_2,0} \\ x_{n_2,1} \\ \dots \\ x_{n_2,P-1} \end{bmatrix}$$

$$\bar{A} = \begin{bmatrix} \alpha_1^0 & \alpha_1^0 & \dots & \alpha_1^0 \\ \alpha_1^0 & \alpha_1^1 & \dots & \alpha_1^{P-1} \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \alpha_1^0 & \alpha_1^{P-1} & \dots & \alpha_1^{(P-1)^2} \end{bmatrix}$$

We now have

$$\bar{x}_0 = \bar{A}\bar{x}_0 + \bar{A}\bar{x}_1 \quad (4.22)$$

$$\bar{x}_1 = \bar{A}\bar{x}_0 - \bar{A}\bar{x}_1 \quad (4.23)$$

$$\text{for } \alpha_2^0 = 1, \quad \alpha_2^1 = -1$$

$\bar{A}\bar{x}_0$ and $\bar{A}\bar{x}_1$ represent simply two P-point Mersenne Number Transforms. Hence the circuits discussed above are suitable for the implementation of a 2P-length convolution and the total computation time is approximately twice as long as for the case of a P-length convolution.

For longer and composite lengths, one may transform the convolution into a two-dimensional cyclic convolution form,

$$y(m_2, m_1) = \sum_{n_2=0}^{P_2-1} \sum_{n_1=0}^{P_1-1} x(n_2, n_1) h(m_2 - n_2, m_1 - n_1)$$

$$\text{where } N = P_1 P_2$$

$$m_2 = 0, 1, \dots, P_2 - 1$$

$$m_1 = 0, 1, \dots, P_1 - 1$$

and use two different Mersenne Number Transforms to carry out the computation [67]. This method needs one multiplication per point, however the price to be paid for using this approach is that arithmetic with double word-lengths has to be used on the second dimension of the convolution. On the other hand, one may also use MNT for the computation on one dimension and use Winograd's eight short convolution algorithms [23-25] for the computation in the other dimension. This method requires a total of $P_1 M_2$ multiplications for the whole computation where M_2 is the number of multiplications for a P_2 -point cyclic convolution using Winograd's short convolution algorithms.

4.5 Extension to Fermat Number Transforms and other Number Theoretic Transforms

Since $nk = \frac{1}{2} [n^2 + k^2 - (n-k)^2]$, the Mersenne Number Transform of eqn. 4.1 can be expressed in the following form

$$X(k) = \left\langle \alpha^{\frac{k^2}{2}} \sum_{n=0}^{P-1} (x(n) \alpha^{\frac{n^2}{2}}) \alpha^{\frac{-(n-k)^2}{2}} \right\rangle_M \quad (4.24)$$

For the existence of eqn. 4.24, the square root of α should exist in order to carry out the computation. For Mersenne Number Transform with $\alpha=2$, $\sqrt{2}$ always exists and is given by

$$\sqrt{2} = 2^{\frac{P+1}{2}}$$

which is in the desirable form of a power of 2.

Similarly, the Inverse Mersenne Number Transform, eqn. 4.13 can also be written as

$$y(m) = \left\langle \frac{1}{P} \alpha^{\frac{-m^2}{2}} \sum_{k=0}^{P-1} (y_k \alpha^{\frac{-k^2}{2}}) \alpha^{\frac{(m-k)^2}{2}} \right\rangle_M \quad (4.25)$$

Similar to eqns. 4.6 and 4.15, eqns. 4.24 and 4.25 represent the Mersenne Number Transform and the Inverse Mersenne Number Transform in circular convolution forms. Unlike the multiplications of $W^{\frac{n^2}{2}}$ and $W^{\frac{k^2}{2}}$ in the chirp z-transform [78,79], for which complex multiplications are required, the multiplication of $\alpha^{\frac{n^2}{2}}$ and $\alpha^{\frac{k^2}{2}}$ required in the above form can simply be implemented by shifts. It is interesting to note that for actual implementation of the MNT pair, eqns. 4.24 and 4.25 for example, the terms $\alpha^{\frac{k^2}{2}}$ and $\alpha^{\frac{-k^2}{2}}$ cancel out. This leaves the terms $\alpha^{\frac{n^2}{2}}$ and $\alpha^{\frac{-m^2}{2}}$ to be implemented. Besides the additional

shifts have to be used to implement $\alpha^{\frac{n^2}{2}}$ and $\alpha^{\frac{-m^2}{2}}$, another disadvantage of this formulation is that the sequence $\{\alpha^{\frac{-(n-k)^2}{2}} : k, n=0, 1, \dots, P-1\}$ in eqn. 4.24 is different in structure from the sequence $\{\alpha^{\frac{(m-k)^2}{2}} : k, m=0, 1, \dots, P-1\}$ in eqn. 4.25. Hence slightly different circuits have to be used for the MNT and the IMNT in the present form. This disadvantage may be removed by rearranging eqn. 4.25 as follows

$$y(m) = < \frac{1}{P} \alpha^{\frac{m^2}{2}} \sum_{k=0}^{P-1} (Y_k \alpha^{\frac{k^2}{2}}) \alpha^{\frac{-(m+k)^2}{2}} >_M \quad (4.26)$$

Hence the sequences $\{\alpha^{\frac{-(n-k)^2}{2}} : k, n=0, 1, \dots, P-1\}$ and $\{\alpha^{\frac{-(m+k)^2}{2}} : k, m=0, 1, \dots, P-1\}$ now have the same structure and can be generated by the same original sequence but rotating in the opposite directions respectively. However, the product of $\alpha^{\frac{k^2}{2}}$ in eqn. 4.24 and $\alpha^{\frac{k^2}{2}}$ in eqn. 4.26 involve no cancellation effect in this case and hence some extra-shifts have to be used to compute this part.

The most desirable and attractive feature of this formulation is that the input sequence and the output sequence are handled in

their natural order which simplifies the design of the control unit. Note also that eqns. 4.24 to 4.26 do not restrict the sequence length to be a prime number implying that this formulation is also suitable for other Number Theoretic Transforms.

For example, a Fermat Number Transform pair can be written as

$$X(k) = \left\langle \alpha^{\frac{k^2}{2}} \sum_{n=0}^{N-1} (x(n) \alpha^{\frac{n^2}{2}}) \alpha^{\frac{-(n-k)^2}{2}} \right\rangle_{F_t} \quad (4.27)$$

$$x(n) = \left\langle \frac{1}{N} \alpha^{\frac{n^2}{2}} \sum_{k=0}^{N-1} (X(k) \alpha^{\frac{k^2}{2}}) \alpha^{\frac{-(n+k)^2}{2}} \right\rangle_{F_t}$$

for $n, k=0, 1, \dots, N-1$ and

$$F_t = 2^{2^t} + 1, \text{ a Fermat number}$$

where $t=1, 2, \dots$

If α is chosen as 2, $\sqrt{2}$ can be represented by [40,28]

$$\alpha^{\frac{1}{2}} = \sqrt{2} = 2^{2^{t-2}} (2^{2^{t-1}} - 1)$$

This is not the simplest representation of α . It would be better to choose α to be 4 or a power of 4 but this will unavoidably shorten the sequence lengths available for Fermat Number Transforms. It should be further remembered that arithmetic used for a Fermat Number Transforms is more complicated than the arithmetic used in Mersenne Number Transforms. Hence special arithmetic [44,45] may have to be used in the computation of Fermat Number Transform thereby

making the FNT less attractive than the MNT.

4.6 Conclusion

Because of the lack of a FFT-type algorithm for the computation, the Mersenne Number Transform has received much less attention compared with the Fermat Number Transform over the last few years. However the hardware design presented in this chapter shows that the MNT convolvers are simpler in structure than the corresponding case for FNT convolvers. This is mainly due to the simplicity in MNT arithmetic and due also to the fact that the MNT can easily be converted into convolution form. This feature leads to our design for the very fast implementation of Mersenne Number Transforms.

The nature of shift-adds in our design has some similarities to the basic operations in distributed arithmetic [80-85]. However, in distributed arithmetic a relatively large ROM size has to be used to form tables for the calculation. In using multi-dimensional techniques to evaluate convolutions (or DFTs), different sets of impulse response coefficients (or W 's in DFTs) are used for the computation of short convolutions (or DFTs), and hence the ROM size may be too large for a practical implementation. However, for the approach presented in this chapter the ROM or RAM size required to store H 's is approximately equal to the sequence length of the convolution and this is small enough for all practical cases.

Although MSI ICs were used in this paper to illustrate the speed achievable with the present design, this new implementation

approach is recommended as a basis for architectures for VLSI Digital Signal Processors. In such a realization, no doubt, the speed of the computation can further be increased.

Mersenne Number Transforms can be used to implement digital filters and the Discrete Fourier Transform very effectively [see chapter 9]. This is due to (i) the fact that MNT provides a means for constructing Digital Signal Processors without roundoff error, and (ii) the present design which gives a fast and efficient approach to compute MNT/IMNT that allows a drastic processing word load reduction over the FFT-type approach.

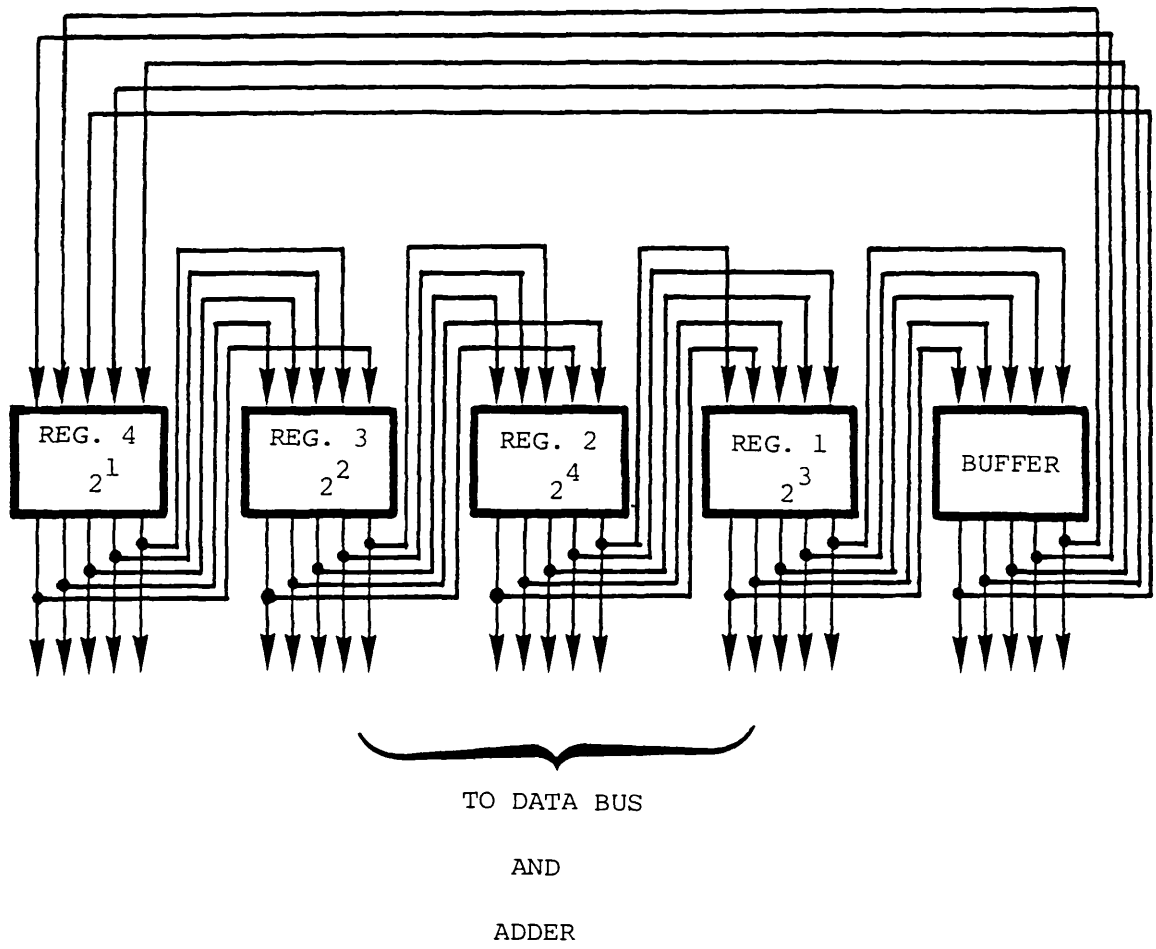


Figure 4.1 Register Arrangement for a 5-point Mersenne
Number Transform Unit

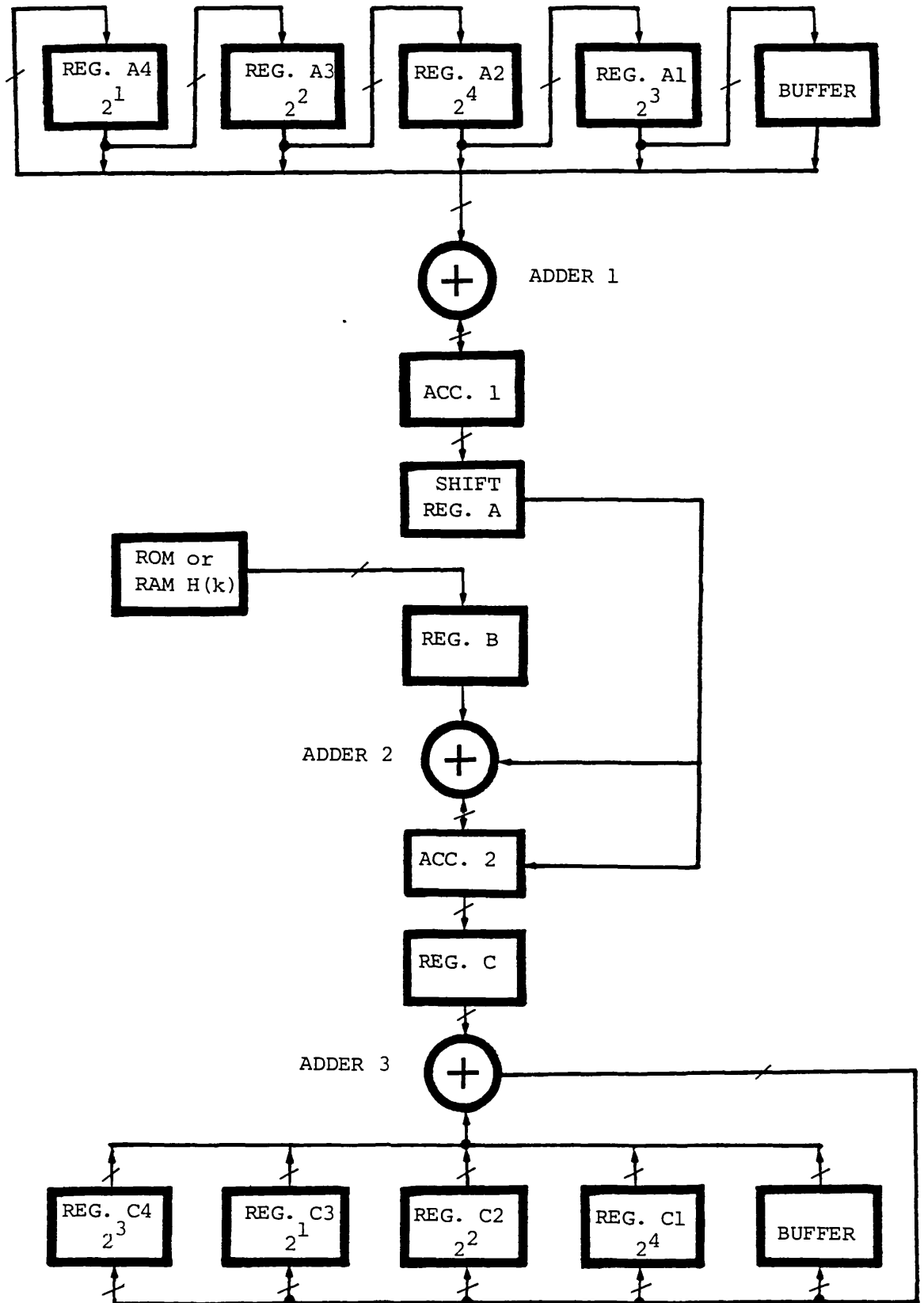


Figure 4.2 Circuit Diagram of a Convolver using Mersenne Number Transform

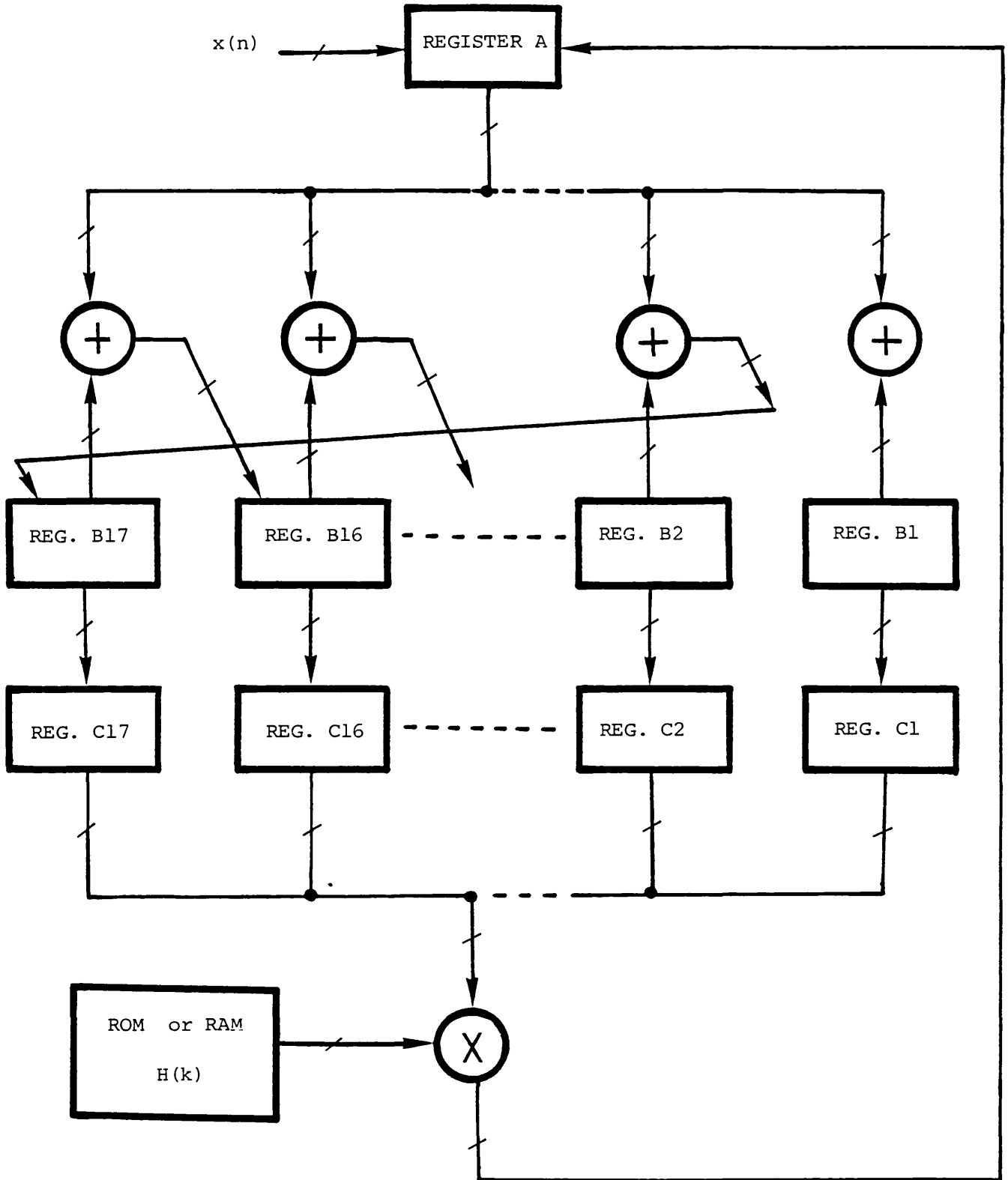


Figure 4.3 Fast Mersenne Number Convolver using Parallel Adders

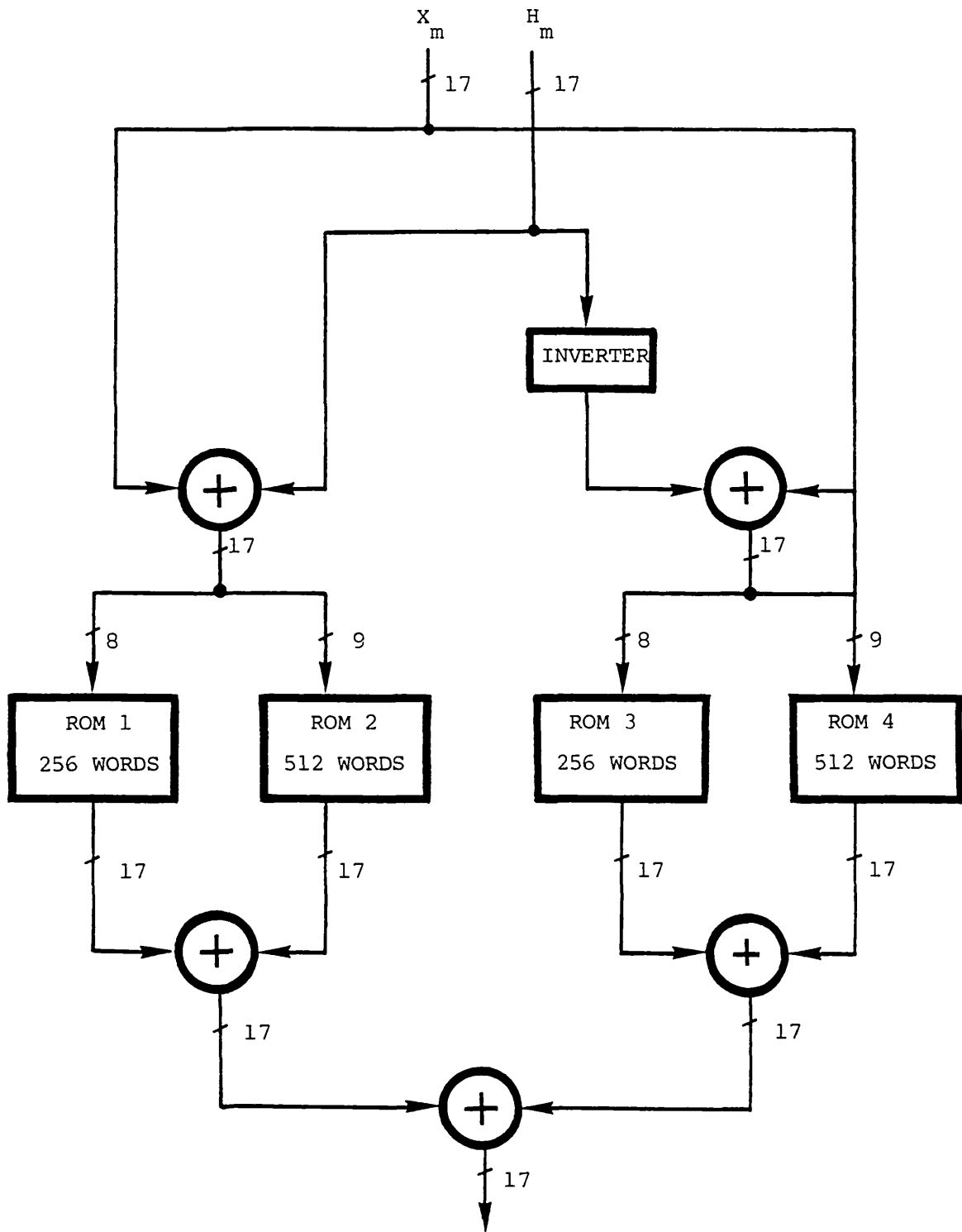


Figure 4.4 17-BIT MERSENNE NUMBER MULTIPLIER

CHAPTER FIVE EFFICIENT IMPLEMENTATION OF CYCLIC CONVOLUTIONS
BY MERSENNE NUMBER TRANSFORMS

The structure discussed in chapter 4 gives rise to simple and efficient designs for the implementation of the Mersenne Number Transform. This is not, however, the only possible way to implement MNTs effectively. Indeed in a completely different approach from the previous chapter, we convert the Mersenne-prime and the Mersenne-composite Number Transforms into recursive filter forms and propose simple hardware structures to carry out the fast implementation of circular convolutions.

Goertzel[86] proposed to realize Discrete Fourier Transforms using hardware structures of first-order digital filters. Curtis and Vickenden[87] improved Goertzel's algorithm to implement the DFT using a recursive first-order filter section requiring only one complex multiply. In this chapter, we first suggest modifying these algorithms to implement Mersenne Number Transforms and propose hardware structures for their realization. We then extend these structures to Mersenne Number Transforms with transform lengths other than P or $2P$. We also present the results of our study employing two efficient methods to compute very long circular convolutions using Mersenne Number Transforms.

A brief account of the contents of the present chapter has been published in [88] whilst an extended version is given in [89].

5.1 Hardware Designs for Mersenne Number Transforms

The Mersenne Number Transform and its inverse can be defined in a more general form than eqns. 4.1 and 4.2 as follows:

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n) \alpha^{nk} \right\rangle_M \quad (5.1)$$

$$x(n) = \left\langle \frac{1}{N} \sum_{k=0}^{N-1} X(k) \alpha^{-nk} \right\rangle_M \quad (5.2)$$

for $k, n=0, 1, \dots, N-1$

Where $M=2^P-1$, a Mersenne number

The root of unity, α , is usually chosen to be 2 or -2 but it may be other suitable values as discussed below. Eqns. 5.1 and 5.2 can be defined for both prime and composite Mersenne numbers where possible values for P are selected from

$$\{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, \dots\}.$$

Now let us consider $N=P$, then eqn. 5.1 becomes eqn. 4.1 as shown below:

$$X(k) = \left\langle \sum_{n=0}^{P-1} x(n) \alpha^{nk} \right\rangle_M \quad (5.3)$$

where $\alpha=2$ and

$$k=0, 1, \dots, P-1$$

In this case $\langle \alpha^P = 1 \rangle_M$ hence α^{nk} in eqn. 5.3 can be written as

$\alpha^{\langle nk \rangle_P}$ for $n = 0, 1, \dots, P-1$. This is because P is a prime number and so the set of integers inside $\{\langle nk \rangle_P : n=1, 2, \dots, P-1\}$ is merely a permutation of the set $\{1, 2, \dots, P-1\}$, for $k=1, 2, \dots, P-1$. This observation implies that the set of data, $\{x(n) : n=1, 2, \dots, P-1\}$ is essentially multiplied by the same set of powers of α , permuted by the expression $\langle nk \rangle_P$ for $k = 1, 2, \dots, P-1$.

Let $nk = m$ for $k = 1, 2, \dots, P-1$, from which we may write immediately

$$n = \langle mk^{-1} \rangle_P$$

Hence eqn. 5.3 can be written as:

$$X(0) = \sum_{n=0}^{P-1} x(n) \quad \text{and} \quad (5.4)$$

$$X(k) = \sum_{m=0}^{P-1} x(\langle k^{-1}m \rangle_P) \alpha^m \quad (5.5)$$

$$\text{for } k=1, 2, \dots, P-1$$

Since P is a prime number the inverse of k in eqn. 5.5 always exists. The data set $\{x(\langle k^{-1}m \rangle_P) : m=0, 1, \dots, P-1\}$ is only a permuted version of the set $\{x(m) : m=0, 1, \dots, P-1\}$ for $k = 1, 2, \dots, P-1$. This is similar to the approach of Curtis for the DFT[87]. Hence

$$\begin{aligned} X(k) = & x(0) + [\dots [x(\langle k^{-1}(P-1) \rangle_P) \alpha + x(\langle k^{-1}(P-2) \rangle_P)] \alpha + x(\langle k^{-1}(P-3) \rangle_P)] \alpha \\ & \dots + x(\langle k^{-1}.2 \rangle_P)] \alpha + x(\langle k^{-1}.1 \rangle_P)] \alpha \end{aligned} \quad (5.6)$$

Eqn. 5.6 can be implemented by using simple hardware or assembly language. The essential feature is that the multiplier is of constant value, α irrespective of the value of $X(k)$ to be evaluated. For the MNT the simplest form of the constant α for binary operation is 2. The value $x(\langle k^{-1}(P-1) \rangle_p)$ is initially loaded into the result register, the result register is then multiplied by α (just a 1-bit circular shift is necessary, if $\alpha = 2$) and then $x(\langle k^{-1}(P-2) \rangle_p)$ is added into the result register. This process is repeated and finally $x(0)$ is added into the result register to form the final result.

The above can be considered as a recursive filter having a transfer function

$$X(z^{-1}) = \frac{1}{1 - z^{-1}\alpha} \quad (5.7)$$

Fig. 5.1 shows a block diagram for the implementation of eqn. 5.7. The multiplication by 2 in MNT arithmetic is only a one bit circular shift. However, a circular shift may still be time consuming for very fast implementations of digital signal processors. The time for the circular shift could be saved completely by just rearranging (shifting circularly) the pin connections for the data transfer from the delay element to the adder in Fig. 5.2.

To exemplify the above, consider the Mersenne Number Transform of length 11. Choose $M = 2^{11} - 1$, $P = 11$ and $\alpha = 2$. Hence eqn. 5.5 can be written in matrix form:

$$\begin{bmatrix} X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \\ X(8) \\ X(9) \\ X(10) \end{bmatrix} = \begin{bmatrix} x(0) & x(1) & x(2) & x(3) & x(4) & x(5) & x(6) & x(7) & x(8) & x(9) & x(10) \\ x(0) & x(6) & x(1) & x(7) & x(2) & x(8) & x(3) & x(9) & x(4) & x(10) & x(5) \\ x(0) & x(4) & x(8) & x(1) & x(5) & x(9) & x(2) & x(6) & x(10) & x(3) & x(7) \\ x(0) & x(3) & x(6) & x(9) & x(1) & x(4) & x(7) & x(10) & x(2) & x(5) & x(8) \\ x(0) & x(9) & x(7) & x(5) & x(3) & x(1) & x(10) & x(8) & x(6) & x(4) & x(2) \\ x(0) & x(2) & x(4) & x(6) & x(8) & x(10) & x(1) & x(3) & x(5) & x(7) & x(9) \\ x(0) & x(8) & x(5) & x(2) & x(10) & x(7) & x(4) & x(1) & x(9) & x(6) & x(3) \\ x(0) & x(7) & x(3) & x(10) & x(6) & x(2) & x(9) & x(5) & x(1) & x(8) & x(4) \\ x(0) & x(5) & x(10) & x(4) & x(9) & x(3) & x(8) & x(2) & x(7) & x(1) & x(6) \\ x(0) & x(10) & x(9) & x(8) & x(7) & x(6) & x(5) & x(4) & x(3) & x(2) & x(1) \end{bmatrix} \begin{bmatrix} 2^0 \\ 2^1 \\ 2^2 \\ 2^3 \\ 2^4 \\ 2^5 \\ 2^6 \\ 2^7 \\ 2^8 \\ 2^9 \\ 2^{10} \end{bmatrix} \quad (5.8)$$

Eqn. 5.6 becomes

$$\begin{aligned}
 X(1) &= x(0) + [[\dots [x(10) \cdot 2 + x(9)] \cdot 2 + x(8)] \cdot 2 + \dots x(2)] \cdot 2 + x(1)] \cdot 2 \\
 X(2) &= x(0) + [[\dots [x(5) \cdot 2 + x(10)] \cdot 2 + x(4)] \cdot 2 + \dots x(1)] \cdot 2 + x(6)] \cdot 2 \\
 &\dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \\
 X(10) &= x(0) + [[\dots [x(1) \cdot 2 + x(2)] \cdot 2 + x(3)] \cdot 2 + \dots x(9)] \cdot 2 + x(10)] \cdot 2
 \end{aligned} \quad (5.9)$$

Both eqns. 5.8 and 5.9 are in the form of recursive filters.

From eqn. 5.8, it is clear that there is a specific rearrangement of the data vector for a specific component of the transform. The sequence of data $x(\langle k^{-1} \rangle_m)_p$ has to be generated as shown in the rectangular matrix by the expression

$$\langle mk^{-1} \rangle_p \text{ for } m, k=1, 2, \dots, P-1$$

This generation procedure is slightly complicated and time consuming. Hence it is more convenient to store the eleven $x(n)$'s in RAM (or buffered registers) and to store the sequence of the locations of $x(n)$'s to be accessed in ROM according to the rectangular matrix shown in eqn. 5.8. The RAM size is 11-words (or P-words, in general) and the ROM size is $11^2 \times 4$ -bit (or $P^2 \times \log_2 P$ bits, in general). Fig. 5.3 shows the hardware structure for the 11-point Mersenne Number Transform processor. The binary counter is used to latch the ROM to output addresses of $x(n)$'s to be added into the adder sequentially.

It is worth emphasizing that this hardware structure is very simple, the number of components used is small and the control complexity is reduced to a minimum, especially when compared with the FFT-type of algorithms.

For inverse Mersenne Number Transform (IMNT), the situation is very similar to MNT. Hence for a 11-point IMNT, we have

$$x(n) = \langle 11^{-1} \sum_{m=0}^{11-1} X(-mn^{-1}) \alpha^m \rangle_M$$

for $n=1,2,\dots,11-1$

and

$$\begin{bmatrix} x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \\ x(8) \\ x(9) \\ x(10) \end{bmatrix} = \frac{1}{11} \begin{bmatrix} X(0) & X(10) & X(9) & X(8) & X(7) & X(6) & X(5) & X(4) & X(3) & X(2) & X(1) \\ X(0) & X(5) & X(10) & X(4) & X(9) & X(3) & X(8) & X(2) & X(7) & X(1) & X(6) \\ X(0) & X(7) & X(3) & X(10) & X(6) & X(2) & X(9) & X(5) & X(1) & X(8) & X(4) \\ X(0) & X(8) & X(5) & X(2) & X(10) & X(7) & X(4) & X(1) & X(9) & X(6) & X(3) \\ X(0) & X(2) & X(4) & X(6) & X(8) & X(10) & X(1) & X(3) & X(5) & X(7) & X(9) \\ X(0) & X(9) & X(7) & X(5) & X(3) & X(1) & X(10) & X(8) & X(6) & X(4) & X(2) \\ X(0) & X(3) & X(6) & X(9) & X(1) & X(4) & X(7) & X(10) & X(2) & X(5) & X(8) \\ X(0) & X(4) & X(8) & X(1) & X(5) & X(9) & X(2) & X(6) & X(10) & X(3) & X(7) \\ X(0) & X(6) & X(1) & X(7) & X(2) & X(8) & X(3) & X(9) & X(4) & X(10) & X(5) \\ X(0) & X(1) & X(2) & X(3) & X(4) & X(5) & X(6) & X(7) & X(8) & X(9) & X(10) \end{bmatrix} \begin{bmatrix} 2^0 \\ 2^1 \\ 2^2 \\ 2^3 \\ 2^4 \\ 2^5 \\ 2^6 \\ 2^7 \\ 2^8 \\ 2^9 \\ 2^{10} \end{bmatrix}$$

(5.10)

A comparison between eqns. 5.8 and 5.10 reveals that exactly the same structure as in the forward case (the same RAM, ROM and etc.) can be used for the inverse Mersenne Number Transform. Note also that for the computation of convolutions using Mersenne Number Transform, the term 11^{-1} in eqn. 5.10 is usually absorbed inside the computation of the transform of the coefficients of the system function and so no extra-multiplication is required.

In the previous paragraphs we have proposed to realize the Mersenne Number Transform pair in a serial data movement basis. Adders are, however, much less expensive than multipliers and so we may use more adders to construct a high speed Mersenne Number Transform processor for the purpose of high speed digital processing. The structure of the present implementation allows us to make a

parallel realization by dividing the computation of $X(k)$'s into B blocks, say for example, as shown in Fig. 5.4. Each block contains one adder and computes approximately $\frac{P}{B}$ $X(k)$'s hence the speed of the implementation can be improved by B times at the expense of $(B-1)$ extra adders and some increase in RAM size.

However, in an optimized case, eqn. 5.1 can actually be rearranged directly in the form of recursive digital filters as shown below:

$$X(k) = x(0) + [\dots [x(P-1) \alpha^k + x(P-2)] \alpha^k + x(P-3)] \alpha^k + \dots x(2)] \alpha^k + x(1)] \alpha^k \quad (5.11)$$

and

$$X_k(z^{-1}) = \frac{1}{1 - z^{-1} \alpha^k} \quad (5.12)$$

$$\text{for } k=0,1,\dots,P-1$$

Eqn. 5.12 is similar to Goertzel's algorithm[87], except that a simple binary number α is used instead of the complex number W . The realization of each $X(k)$ can be considered as a simple recursive filter with a single integer pole, α^k . Now, $\alpha = 2$, the multiplication of α^k means circular shifts by k positions only. As in our previous case, these circular shifts could be saved completely by rearranging (shifting circularly) the pin connections for sending data from the registers to the adders. For computing $X(0)$, no pin rearrangement is necessary. For computing $X(1)$, the pin connection is off-set (shifted) by one position circularly; for computing $X(2)$, the

pin connection is off-set (shifted) by two positions circularly and so on. Exactly the same circuit can be used to calculate the inverse of Mersenne Number Transform, except that the results are read in the reversed order.

Now, let us consider the cyclic convolution of two sequences, $\{x(n):n=0,1,\dots,N-1\}$ and $\{h(n):n=0,1,\dots,N-1\}$,

$$y(m) = \sum_{n=0}^{N-1} x(n)h(m-n) \quad (5.13)$$

for $m=0,1,\dots,N-1$

Let $N=P$, a prime number. Because of the cyclic convolution property, the $y(m)$'s may be calculated using the Mersenne Number Transform:

$$y(m) = \left\langle \frac{1}{P} \sum_{k=0}^{P-1} X(k)H(k) \alpha^{-mk} \right\rangle_P$$

for $m=0,1,\dots,P-1$

where $\{H(k):k=0,1,\dots,P-1\}$ and $\{X(k):k=0,1,\dots,P-1\}$ are the Mersenne Number Transformed terms arising from the sequences $\{x(n):n=0,1,\dots,P-1\}$ and $\{h(n):n=0,1,\dots,P-1\}$. Hence the computation of $y(m)$'s can be implemented by our proposed hardware structure. Fig. 5.5 shows the block diagram of the proposed high speed Mersenne Number Transform convolver. Register A holds the input data which is added into registers B_0 to B_{P-1} in parallel. After $P-1$ additions, $\{X(k),k=0,1,\dots,P-1\}$ are then stored inside $C_0, C_1, \dots, C_{P-1}$. The $X(k)$'s are then sent serially to the multiplier for the multiplication by the corresponding $H(m)$'s. After the first product has been

obtained $(X(0)H(0)$ for example), it is sent to register A for computing the inverse Mersenne Number Transform. Hence the whole operation can be done in a parallel and pipeline form. After P multiplications, the final product $(X(P-1)H(P-1)$ for example) is sent into A and after one more addition the convolution sum results corresponding to $y(m)$'s are stored inside register $C_0, C_1, \dots, C_{P-1}$.

The control of data flow in the present design is extremely simple and the total convolution time is approximately equal to the time for P multiplications and P additions. Hence this computation technique requires just one addition time and one multiplication time per point for the calculation of a P-point convolution.

5.2 Sequence Lengths of the Transforms Modulo a Mersenne Number

For a Mersenne Number Transform with $M = 2^P - 1$ and $\alpha = 2$, we have $\langle 2^P = 1 \rangle_M$ hence the sequence length must be P. This implies that if we want to retain the simplicity of the MNT arithmetic and to have sequence lengths longer than P (or actually to have sequence lengths other than P), α has to be chosen to have values other than 2.

Recall eqn. 2.29

$$N | (p_i - 1) \quad (5.14)$$

where p_i 's are prime factors of M

for $i=1, 2, \dots, I$

Eqn. 5.14 is a necessary condition for N to form a proper MNT with the convolution property.

If P is an odd prime, it can be shown [27-28] that any divisor of $M=2^P-1$ is of the form $2k_iP+1$. That is all M_i 's of the Mersenne numbers are in the form of $2k_iP+1$. Hence

$$\Phi(M_i) = (P_i-1) = 2k_iP \quad (5.15)$$

Eqns. 5.14 and 5.15 show that

$$N | 2k_iP \quad (5.16)$$

for $i=1,2,\dots,I$

This expression suggests that 2, P and 2P are possible sequence lengths for Mersenne Number Transform. It is easily checked that $-1(=M-1)$, 2 and $-2(=M-2)$ form the roots of unity and satisfy the three requirements in eqn. 2.24 for MNT with sequence lengths 2, P and 2P respectively. This is true for both Mersenne primes (for $P=3,5,7,13,17,19,31,61,\dots$) and Mersenne composites (for $P=11,23,29,37,41,43,47,53,59,\dots$).

If a sequence length of 2P is used, the MNT can be written as

$$X(k) = \left\langle \sum_{n=0}^{2P-1} x(n) \alpha^{nk} \right\rangle_M \quad (5.17)$$

For the actual implementation of this 2P-MNT using our proposed hardware design, eqn. 5.17 can be simplified as shown below.

Let us define

$$X_0(\ell) = \left\langle \sum_{n=0}^{P-1} x(2n) \beta^{\ell n} \right\rangle_M \quad (5.18)$$

$$X_1(\ell) = \left\langle \sum_{n=0}^{P-1} x(2n+1) \beta^{\ell n} \right\rangle_M \quad (5.19)$$

for $\ell=0,1,\dots,P-1$

where $\beta = \alpha^2 = 4$ and

$$\langle \beta^P \rangle_M = 1 \quad .$$

Since

$$X_0(\ell) = X_0(\ell+rP) \text{ and} \quad (5.20)$$

$$X_1(\ell) = X_1(\ell+rP)$$

for any integer r it follows that $X_0(\ell)$ and $X_1(\ell)$ are periodic function of period P . Now by substituting eqns. 5.18 and 5.19 into eqn. 5.17, we obtain

$$\begin{aligned} X(k) &= \left\langle \sum_{n=0}^{P-1} x(2n) \alpha^{2nk} + \sum_{n=0}^{P-1} x(2n+1) \alpha^{(2n+1)k} \right\rangle_M \\ &= \left\langle \sum_{n=0}^{P-1} x(2n) \alpha^{2nk} + \alpha^k \sum_{n=0}^{P-1} x(2n+1) \alpha^{2nk} \right\rangle_M \\ &= \langle X_0(k) + X_1(k) \alpha^k \rangle_M \end{aligned} \quad (5.21)$$

for $k=0,1,\dots,2P-1$

Consider the terms $X_0(k)$ and $X_1(k) \alpha^k$ in eqn. 5.21 seperately.

(i) Since $X_0(\ell) = X_0(\ell+rP)$ for $\ell=0,1,\dots,P-1$ it follows that all $X_0(k)$ for $k=0,1,\dots,2P-1$ can be computed by a length-P MNT as shown in eqn. 5.18.

(ii) Let us consider $X_1(k) \alpha^k$ for $k=0,1,\dots,P-1$. Then,

$$X_1(k) \alpha^k = \left\langle \sum_{n=0}^{P-1} x(2n+1) \beta^{kn} \alpha^k \right\rangle_M$$

But $\alpha = -2 = \beta^{\frac{P+1}{2}}$, therefore

$$X_1(k) \alpha^k = \left\langle \sum_{n=0}^{P-1} x(2n+1) \beta^{(n+\frac{P+1}{2})k} \right\rangle_M$$

Let $m = (n + \frac{P+1}{2})$, so that

$$X_1(k) \alpha^k = \left\langle \sum_{m=0}^{P-1} x(2m-P) \beta^{mk} \right\rangle_M$$

$$\text{That is } X_1(k) \alpha^k = \left\langle \sum_{n=0}^{P-1} x(2n-P) \beta^{nk} \right\rangle_M \quad (5.22)$$

where m is a dummy variable

Eqn. 5.22 is merely a shifted version of a P -point Mersenne Number Transform. For $k=P, P+1, \dots, 2P-1$, we notice that $\alpha^P = -1$. Therefore

$$X_1(\ell) = -X_1(\ell+rP) \text{ for } \ell = 0, 1, \dots, P-1 \text{ and } r=\text{odd}$$

hence all $X_1(k)$ for $k=0,1,\dots,2P-1$ can be found by a length- P

MNT as shown in eqn. 5.22.

This shows that a total of two P-point Mersenne Number Transforms are sufficient to compute a length $2P$ MNT and our basic P-point MNT module is suitable for this arrangement.

For sequence lengths other than 2 , P and $2P$ we consider Mersenne-composite and Mersenne-prime Number Transforms individually.

(i) Mersenne-composites

Eqn. 5.14 states that N must divide $\Phi(P_i)$ for $i=1,2,\dots,I$, where I =the total number of prime factors of M . Table 5.1 lists the prime factorization of $\Phi(P_i)$ for some Mersenne numbers of practical lengths. From this table it is clear that besides 2 , P and $2P$ there are evidently no other sequence lengths satisfying eqn. 5.14 for most of the Mersenne-composites. The only exception in this table is that of 3 , which is a prime factor of both 222 and 616318176 for the case where $P=37$ and so for practical purpose it is possible to define a sequence length of 3 for $P=37$. With the help of some multi-dimensional techniques one may compute convolutions of length $2 \times 3 \times 37$ with one multiplication per point on the integer ring modulo $(2^{37}-1)$. However, for other Mersenne-composite Number Transforms some indirect methods have to be used to increase the sequence lengths.

(ii) Mersenne-primes

For Mersenne-prime Number Transforms, M is a prime number and hence $I=1$, or $M=M_1$. In this case all prime factors of $\Phi(M)$ are possible prime sequence lengths for the transform. In order to make

these prime sequence lengths attractive, the root of unity α has to be chosen to have a very simple binary representation, otherwise some general multiplications have to be used. If $\Phi(M) = \prod_{i=1}^I p_i^{r_i}$, there are $\Phi(p_i)$ possible roots of unity of order p_i for each possible prime transform length, p_i . For a long sequence we have, therefore, a larger number of possible roots of unity that could be chosen. With the exception of length P or $2P$ for the MNT, α can not be a power of 2. However if we use the proposed recursive form in eqn. 5.6 (see Fig. 5.1 also) to compute the Mersenne Number Transform, the implementation can still be simple.

The basic operation in eqn. 5.6 can be written as

$$(a + \alpha b).$$

Now, α may not be a power of 2. However it has a simple binary form and so may be decomposed into several words, each of which is simply a power of two (that is only one bit set in each word). Therefore we need add only a few adders to the implementation.

For example, for $M=2^{17}-1$ from table 2, a simple α for the order, $N=257$ is 16900 (=100001000000100 in binary form). Therefore $(a + \alpha b)$ can be implemented as

$$\begin{aligned} a + \alpha b &= a + 16384b + 512b + 4b \quad (\text{decimal}) \\ &= a + 1000000000000000b + 1000000000b + 100b \quad (\text{binary}) \quad (5.23) \end{aligned}$$

Fig. 5.6 shows an implementation of eqn. 5.23. Three adders are used in total and the addition time for one computation of the expression

$(a + \alpha b)$ is just equivalent to the time required for two normal additions. Simplification of α multiplication can also be done by observing the repetition in bit pattern of α . For example, $90(1011010, \text{binary})$ is a root of unity of order 3 for $P=13$. The multiplication of 90 by b can be implemented as $[1010b + 1010000b]$. After 1010 has found, 1010000 can be obtained quite simply. Hence two additions are sufficient for the multiplication of 90 by b . On the other hand, if α contains a series of consecutive ones, we may use Booth's multiplication algorithm [90] to simplify the computation.

Table 2 shows some possible simple α 's and the corresponding addition times for the computation of $(a + \alpha b)$ for different prime sequence lengths. Of course, the addition time for lengths 2 and P is just the time for one MNT addition (and the overflow bit can always be taken care of by the next addition). For other prime sequence lengths, the computation of $(a + \alpha b)$, for most cases, requires two to three MNT addition times.

5.3 Long Sequence Lengths

For the computation of convolutions of long sequences, we can use some multi-dimensional techniques. Since P and $2P$ are the most efficient (in terms of simplicity in hardware and computation time) sequence lengths for the computation of convolutions using MNT's, they can be used as the basic sequence lengths for the implementation. Usually the wordlength of a system is determined by the dynamic range required. If the wordlength is determined, the basic sequence length

which is equal to P or $2P$ is obtained as well. We can then construct the whole multi-dimensional equation for the computation of convolutions of long sequences using Mersenne Number Transforms.

In our discussion below, we consider a length- P MNT module as a basic unit. It consists basically of P adders and P accumulators built in dedicated hardware or even fabricated on a single chip. The total time for P -point MNT or IMNT is P addition times or one addition time per point.

Let the length of the convolution sequences be a composite number, N . Set $N = N_1 N_2$, where N_1 ($=P$ or $2P$) is the basic sequence length of the MNT. We can then map the one-dimensional cyclic convolution into a two-dimensional convolution form. The map could be written in a linear form

$$n = \langle K_1 n_1 + K_2 n_2 \rangle_N \quad (5.24)$$

for $n_1, n_2 = 0, 1, \dots, N_1, N_2$ respectively.

Burrus[68] showed that if N_1 and N_2 are mutually prime, the resultant sequences may be cyclic in both n_1 and n_2 dimensions. On the other hand, if N_1 and N_2 are not mutually prime, the resultant sequences can be cyclic only in n_1 or n_2 . Let us consider both of these cases.

(i) $(N_1, N_2) \neq 1$

In this case, the map cannot be cyclic in both dimensions[68]. Choose n_1 to be cyclic hence a possible choice of K_1

is N_2 and K_2 is 1. The dimension n_1 implies a cyclic convolution sum whereas the dimension n_2 implies a linear convolution sum. If $N_1=P$, the cyclic convolution sum can be computed by a P-point MNT (a basic module, as discussed before). The linear convolution sum can either be computed by the Lagrange Interpolating Formula or a second Number Theoretic Transform. Both of these methods require $(2 - \frac{1}{N_2})$ multiplications per point. Two interesting cases are:

(a) $N=4P$

In this case, we may put $N_1=2P$ and $N_2=2$. The length two linear convolution sum can be calculated by the Lagrange Interpolating formula which requires 1.5 multiplications per point, whereas the n_2 dimension can be calculated by the $2P$ MNT algorithm as discussed before.

(b) $N=P(P+1)$, for $(P+1)=4k$

In this case, we may write $N_1=2P$ and $N_2=\frac{P+1}{2}$. Hence N_2 is not relatively prime to N_1 . n_1 is arranged to be cyclic and n_2 is non-cyclic. Similar to the previous case, the n_1 -dimension can be computed by $2P$ -point MNT whereas the $\frac{P+1}{2}$ linear convolution sum can be computed by a length- P MNT (by appending zeros at the end of the $\frac{P+1}{2}$ sequence). For example, if $N=992$ we may write $N_1=2P=2 \times 31$ and $N_2=16$. Hence this 992 point cyclic convolution sum can be computed by 16 62-point MNT in the forward transform, 62 16-point linear convolution and 16 62-point inverse MNT in the reverse transform. A 16-point linear convolution can always be computed by a 31-point cyclic

convolution using the 31-point Mersenne Number Transform pair. Hence the 31-point MNT module can be used as a basic unit for the computation in both dimensions. This method requires $(2 - \frac{1}{16})$ multiplications per point for the computation of a 992-point cyclic convolution.

(ii) $(N_1, N_2) = 1$

In this case, both n_1 and n_2 are cyclic. Hence eqn. 5.13 can be written as

$$y(m_2, m_1) = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{N_1-1} x(n_2, n_1) h(m_2 - n_2, m_1 - n_1) \quad (5.25)$$

for $m_2 = 0, 1, \dots, N_2$ and

$m_1 = 0, 1, \dots, N_1$

If $N_1 = P$, the first dimension, n_1 can be computed by the P-point MNT. Eqn. 5.25 becomes

$$y(m_2, m_1) = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{P-1} x(n_2, n_1) h(m_2 - n_2, m_1 - n_1) \quad (5.26)$$

where $N = N_2 P$

$m_2 = 0, 1, \dots, N_2 - 1$

$m_1 = 0, 1, \dots, P - 1$

The second dimension, n_2 can be computed in three different ways:

(a) If N_2 is a factor of $\Phi(M) = 2^P - 2$ where M is a prime number or if N_2 is a factor of all $\Phi(M_i)$'s where $M = \prod_{i=1}^I M_i$, the N_2 -point cyclic

convolution can be computed by constructing a second Mersenne Number Transform module with $\alpha \neq 2$ as discussed in the previous section. This requires an increase in the number of adders and the number of addition times per point increases from 1 to 2 to 3 in this dimension. The major advantage of this method is that the number of multiplications per point remains as 1. For example, for a 266-point cyclic convolution, we may choose $M=2^{19}-1$ and write $N = N_3 N_2 N_1$ where $N_3=2$, $N_2=7$ and $N_1=19$. The time taken for a 19-point MNT inside a 19-point MNT module is 19 MNT addition times. The time taken for the computation of a 7-point cyclic convolution using a 7-point MNT module (parallel design, with 35 adders) is $7 \times 3 (=21)$ MNT addition times. Hence the total time for the cyclic convolution of two 266 sequences is

$$\begin{aligned} t &= 2[133 \times 2 + 2 \times 7 \times 19 + 2 \times 19 \times 21]t_a + 266t_m \\ &= 2660t_a + 266t_m \end{aligned}$$

The time for the computation of the transform of $\{h(n): n=0,1,\dots,N-1\}$ is not counted since this can be pre-calculated before the setting up of the system. Hence the time for computing one point is

$$t/\text{point} = 10t_a + t_m$$

Using adders similar to those used in Ref. [44,45], the time for one addition could be 25nS and the time for one multiplication could be 100nS. Hence the execution time per point is approximately equal to

$$(10 \times 25 + 100) \text{ nS} = 350 \text{ nS}$$

Notice also that the speed for the computation can be increased further if a pipe-line architecture is used between MNT modules.

Parallel implementation of the MNT modules has been taken as an example of the technique. However it is clear that serial or semi-serial implementation of the MNT modules may be more suitable and efficient for computing convolutions using the present design if one wants to use fewer adders. But, in this case, the computation time is unavoidably longer than the parallel implementation discussed above.

Table 5.3 lists the number of operations for various cyclic convolutions computed by parallel/serial(using one adder only) implementations of the MNT modules. Notice that, in the case of the serial implementation, the number of additions for the present technique is significantly less than the number of additions($=2(P-1)$ per point) for the cyclic convolution of a similar sequence length using normal MNT (with α being a power of two, and with the transform length being a prime number, P). For example, for a 114-point cyclic convolution, the number of additions per point is 52.8, which is much smaller than $2(114-1) = 226$. Hence this new technique gives a large reduction in the number of additions for both serial and parallel implementations.

(b) If $N_2 \leq \frac{N_1+1}{2}$, similar to the case when $(N_1, N_2) \neq 1$, we can append $(N_1 - N_2)$ zeros at the end of the N_2 -length sequences, hence the same basic P -point MNT module can be used to make the computation. The

most efficient situation for this case is that $N_2 = \frac{N_1+1}{2}$, where $N_1=P$. The total number of multiplications per point is $(2N_2-1)/N_2$ or $(2-1/N_2)$. For example, a 190-point cyclic convolution can be computed by a two-dimensional convolution each of length-19 points. The first dimension requires a total of 19 addition times for the computation of one 19-point MNT whereas the second dimension requires only 10 addition times for the computation of the MNT since no addition is required for the appended zeros. However 9 additions have to be used to form 10 results from the 19 values (results) obtained from the MNT in the second dimension. Hence the total time for the computation of a 190-point convolution is

$$(10 \times 19 \times 2 + 19 \times 19 \times 2)t_a + 19 \times 19 t_m$$

This implies that the computation time per point is approximately 1.9 multiplication and 5.8 addition times. Using the same numerical values for t_a and t_m as before, the computation time per point is approximately equal to 335ns. Table 5.4 lists some possible sequence lengths and the corresponding computation times for this technique. A disadvantage of this method is that this technique requires more than one multiplication per point. However the major advantage in this technique is that the same MNT module can be used to calculate both dimensions of the convolution and the total number of adders to be used is equal to P.

(c) We may also use MNT for the computation in one dimension and use other short convolution algorithms suggested by Winograd[25], Agarwal and Cooley[24] in the second dimension. In this case the control of

data flow is more complicated and the number of multiplications per point is also large. However, this method may give us a greater choice of sequence lengths and may require a small number of additions.

5.4 Conclusion

The Mersenne Number Transform uses the simplest arithmetic among all Number Theoretic Transforms. However due to the lack of a FFT-type of algorithm, the MNT has been considered in the past less attractive compared with the Fermat Number Transform. The approach presented in this chapter, however, shows that MNT have considerable desirable features. Indeed a very simple process control of the data flow has been shown for the proposed MNT hardware modules and it has been illustrated that the Mersenne Number Transforms are very efficient for the computation of short and long cyclic convolutions. Note also that MSI ICs were used to illustrate the speed achievable with the present design, both the speed and the design complexity could be still further improved if VLSI is used to make the implementation.

This chapter and the previous chapter both describe new design of implementation of the Mersenne Number Transform. The structures proposed in chapter four use no ROM or RAM and this means that shorter data access times for their implementation can be achieved, whereas the structure described in this chapter may be used to relax the word-length and sequence length constraint of the MNT.

In the next chapter, we introduce a new NTT which has no sequence length and word-length constraint and requires a very simple hardware structure for its implementation.

Table 5.1

P	$\Phi(P_i)$	Prime Factorization
(A) Mersenne-prime		
5	30	$2 \times 3 \times 5$
7	126	$2 \times 3^2 \times 7$
13	8190	$2 \times 3^2 \times 5 \times 7 \times 13$
17	131070	$2 \times 3 \times 5 \times 17 \times 257$
19	524286	$2 \times 3^3 \times 7 \times 19 \times 73$
31	2147483646	$2 \times 3^2 \times 7 \times 11 \times 31 \times 151 \times 331$
(B) Mersenne-composite		
11	22	2×11
	88	$2^3 \times 11$
23	46	2×23
	178480	$2^4 \times 5 \times 23 \times 97$
29	232	$2^3 \times 29$
	1102	$2 \times 19 \times 29$
	2088	$2^3 \times 3^2 \times 29$
37	222	$2 \times 3 \times 37$
	616318176	$2^5 \times 3 \times 37 \times 167 \times 1039$
41	13366	$2 \times 41 \times 163$
	164511352	$2^3 \times 41 \times 59 \times 8501$
43	430	$2 \times 5 \times 43$
	9718	$2 \times 43 \times 113$
	2099862	$2 \times 3^2 \times 43 \times 2713$
47	2350	$2 \times 5^2 \times 47$
	4512	$2^5 \times 3 \times 47$
	13264528	$2^4 \times 31 \times 47 \times 569$

Prime Factorization of $\Phi(P_i)$ for Mersenne Numbers,

$$M = (2^P - 1) = \prod_{i=1}^r P_i$$

Table 5.2

P	$\Phi(M)$	Prime Factor	Possible α	Number of Addition times for one $(a + \alpha b)$ Operation
5	30	2	-1	1
		3	5	2
		5	2	1
7	126	2	-1	1
		3	19	2
		7	2	1
13	8190	2	-1	1
		3	90	3
		5	1904	3
		7	3085	3
		13	2	1
17	131070	2	-1	1
		3	21905	3
		5	618	3
		17	2	1
		257	16900	2
19	524286	2	-1	1
		3	69492	3
		7	272928	3
		19	2	1
		73	8482	3
31	2147483646	2	-1	1
		3	634005911	4
		7	1205362885	4
		11	100973744	4
		31	2	1
		151	66176224	3
		331	1090433	3

α and Computation time for different sequence lengths of
Mersenne Number Transforms

Table 5.3

P	Possible Sequence Length N	Factors of N	Number of Multi. per point	Number of Additions per point (Serial Imple.)	Number of Addition times(t_a) per point (Parallel Imple.)
5	5	5	1.0	8.0	2
	10	2x5	1.0	10.0	4
	30	2x3x5	1.0	16.7	8
7	7	7	1.0	12.0	2
	14	2x7	1.0	14.0	4
	42	2x3x7	1.0	23.3	8
13	13	13	1.0	24.0	2
	26	2x13	1.0	26.0	4
	78	2x3x13	1.0	35.3	10
	130	2x5x13	1.0	46.8	10
	390	2x3x5x13	1.0	56.1	16
	546	2x3x7x13	1.0	78.2	16
	910	2x5x7x13	1.0	89.7	16
	2730	2x3x5x7x13	1.0	99.0	22
17	17	17	1.0	32.0	2
	34	2x17	1.0	34.0	4
	102	2x3x17	1.0	48.7	10
	170	2x5x17	1.0	67.6	10
	510	2x3x5x17	1.0	82.3	16
19	19	19	1.0	36.0	2
	38	2x19	1.0	38.0	4
	114	2x3x19	1.0	52.7	10
	266	2x7x19	1.0	91.1	10
	798	2x3x7x19	1.0	105.8	16
31	31	31	1.0	60.0	2
	62	2x31	1.0	62.0	4
	186	2x3x31	1.0	87.3	12

434	2x7x31	1.0	135.7	12
1302	2x3x7x31	1.0	161.1	20
2046	2x3x11x31	1.0	216.4	20
4774	2x7x11x31	1.0	264.8	20
14322	2x3x7x11x31	1.0	290.2	28

Number of Multiplications, Additions (serial implementation) and Addition times, t_a (parallel implementation) for the computation of Cyclic Convolutions using Multiple MNT modules.

Table 5.4

P	Possible Sequence Length(N)	Factors of N	Number of Multiplications per point	Number of Addition Times (t_a) per point
Mersenne-prime				
5	5	5	1.00	2.00
	10	2x5	1.00	4.00
	15	3x5	1.67	5.33
	20	2x2x5	1.50	8.00
	30	3x10	1.67	7.33
7	7	7	1.00	2.00
	14	2x7	1.00	4.00
	28	2x2x7	1.50	8.00
	28	4x7	1.75	5.50
	48	4x14	1.75	7.50
13	13	13	1.00	2.00
	26	2x13	1.00	4.00
	52	2x2x13	1.50	8.00
	91	7x13	1.86	5.71
	182	7x26	1.86	7.71
17	17	17	1.00	2.00
	34	2x17	1.00	4.00
	68	2x2x17	1.50	8.00
	153	9x17	1.89	5.78
	306	9x34	1.89	7.78
19	19	19	1.00	2.00
	38	2x19	1.00	4.00
	76	2x2x19	1.50	8.00
	190	10x19	1.90	5.80
	380	10x38	1.90	7.80
31	31	31	1.00	2.00
	62	2x31	1.00	4.00
	124	2x2x31	1.50	8.00
	496	16x31	1.94	5.88
	992	16x62	1.94	7.88

Table 5.4 (Continued)

Mersenne-composite				
11	11	11	1.00	2.00
	22	2x11	1.00	4.00
	44	2x2x11	1.50	8.00
	66	6x11	1.83	5.67
	132	6x22	1.83	7.67
23	23	23	1.00	2.00
	46	2x23	1.00	4.00
	92	2x2x23	1.50	8.00
	276	12x23	1.92	5.83
	552	12x46	1.92	7.83
29	29	29	1.00	2.00
	58	2x29	1.00	4.00
	116	2x2x29	1.50	8.00
	435	15x29	1.93	5.87
	870	15x58	1.93	7.87
37	37	37	1.00	2.00
	74	2x37	1.00	4.00
	148	2x2x37	1.50	8.00
	703	19x37	1.95	5.90
	1406	19x74	1.95	7.90
41	41	41	1.00	2.00
	82	2x41	1.00	4.00
	164	2x2x41	1.50	8.00
	861	21x41	1.95	5.91
	1722	21x82	1.95	7.91
43	43	43	1.00	2.00
	86	2x43	1.00	4.00
	172	2x2x43	1.50	8.00
	946	22x43	1.96	5.91
	1892	22x86	1.96	7.91
47	47	47	1.00	2.00
	94	2x47	1.00	4.00
	188	2x2x47	1.50	8.00
	1128	24x47	1.96	5.92
	2256	24x94	1.96	7.92

Number of Multiplications and Addition times for the Computation of Cyclic Convolutions using Nested MNT's (only the length-P MNT module used)

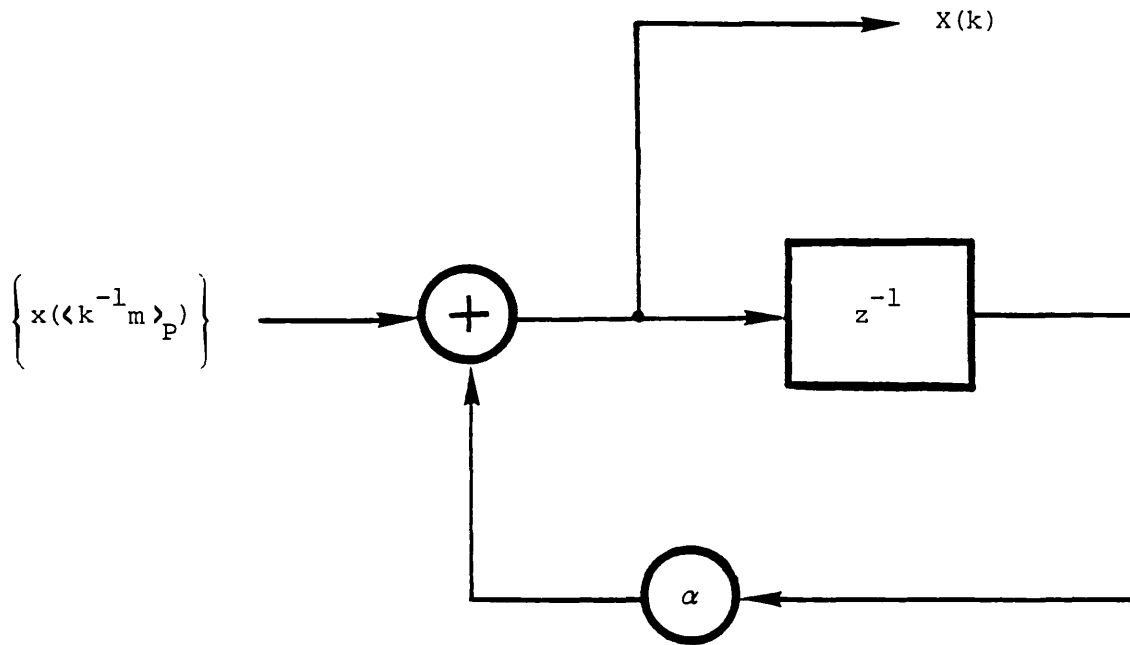


Figure 5.1 Block Diagram for the Recursive Implementation of
Mersenne Number Transform

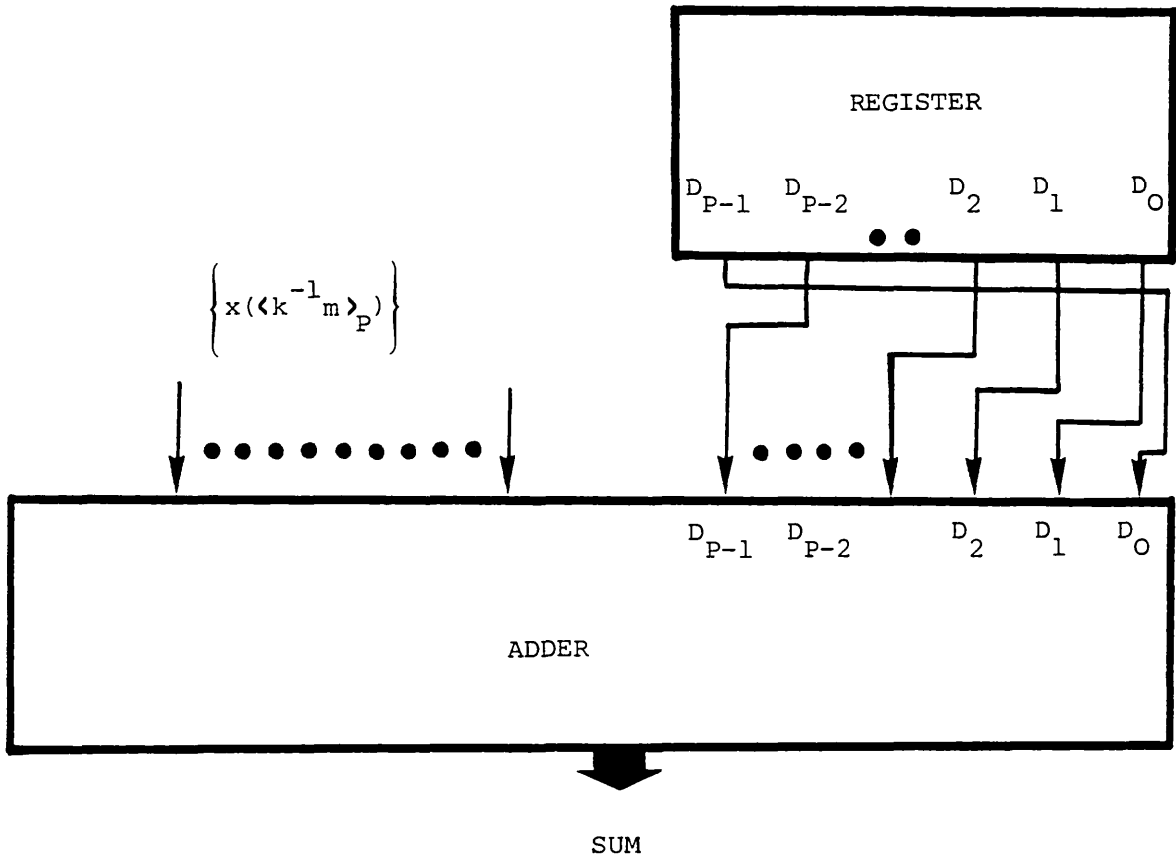


Figure 5.2 Pin Connection for the Implementation of the Multiplication by 2 for Mersenne Number Transform Arithmetic

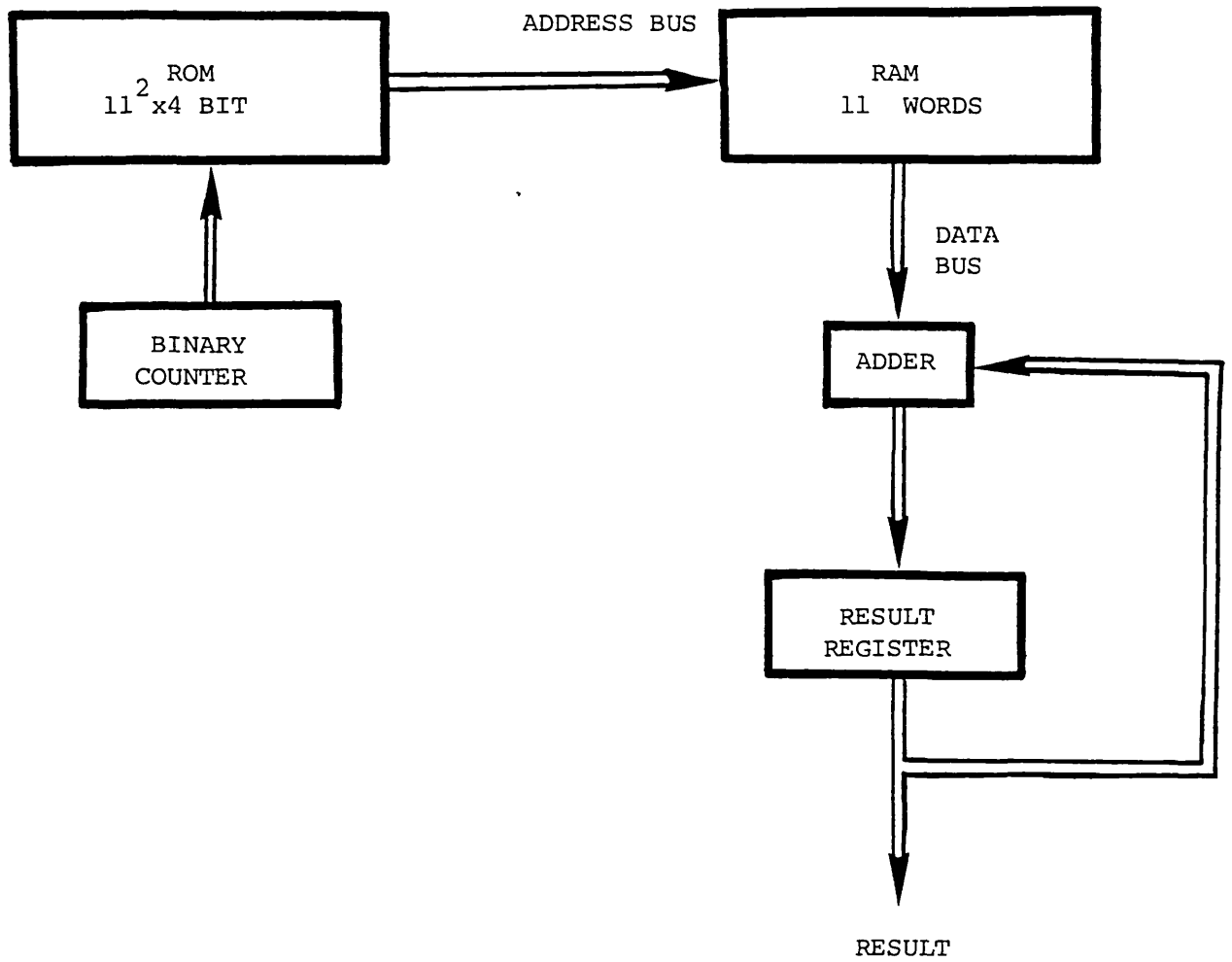


Figure 5.3 Hardware Structure for a 11-point Mersenne Number Transform Processor

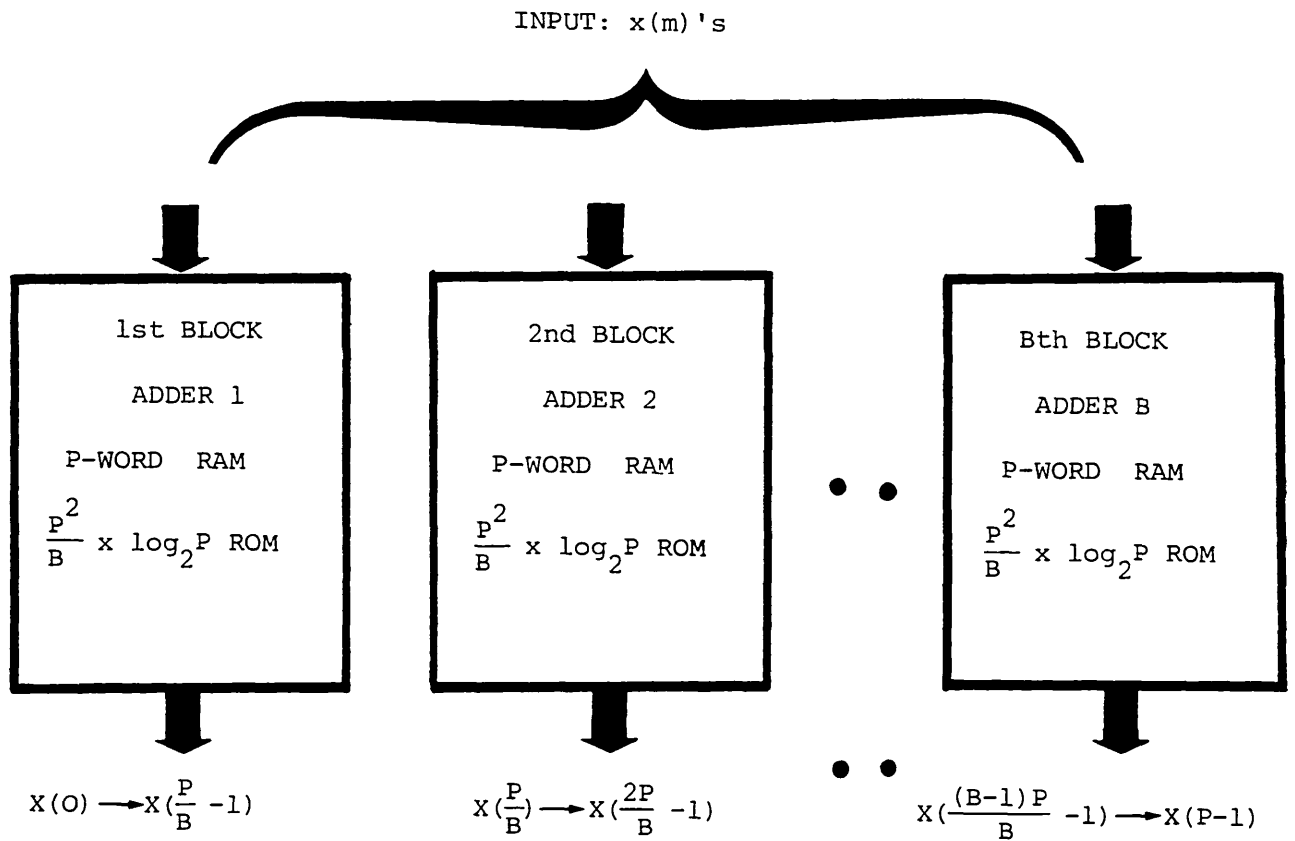


Figure 5.4 Block Diagram for Parallel Implementation of
Mersenne Number Transform

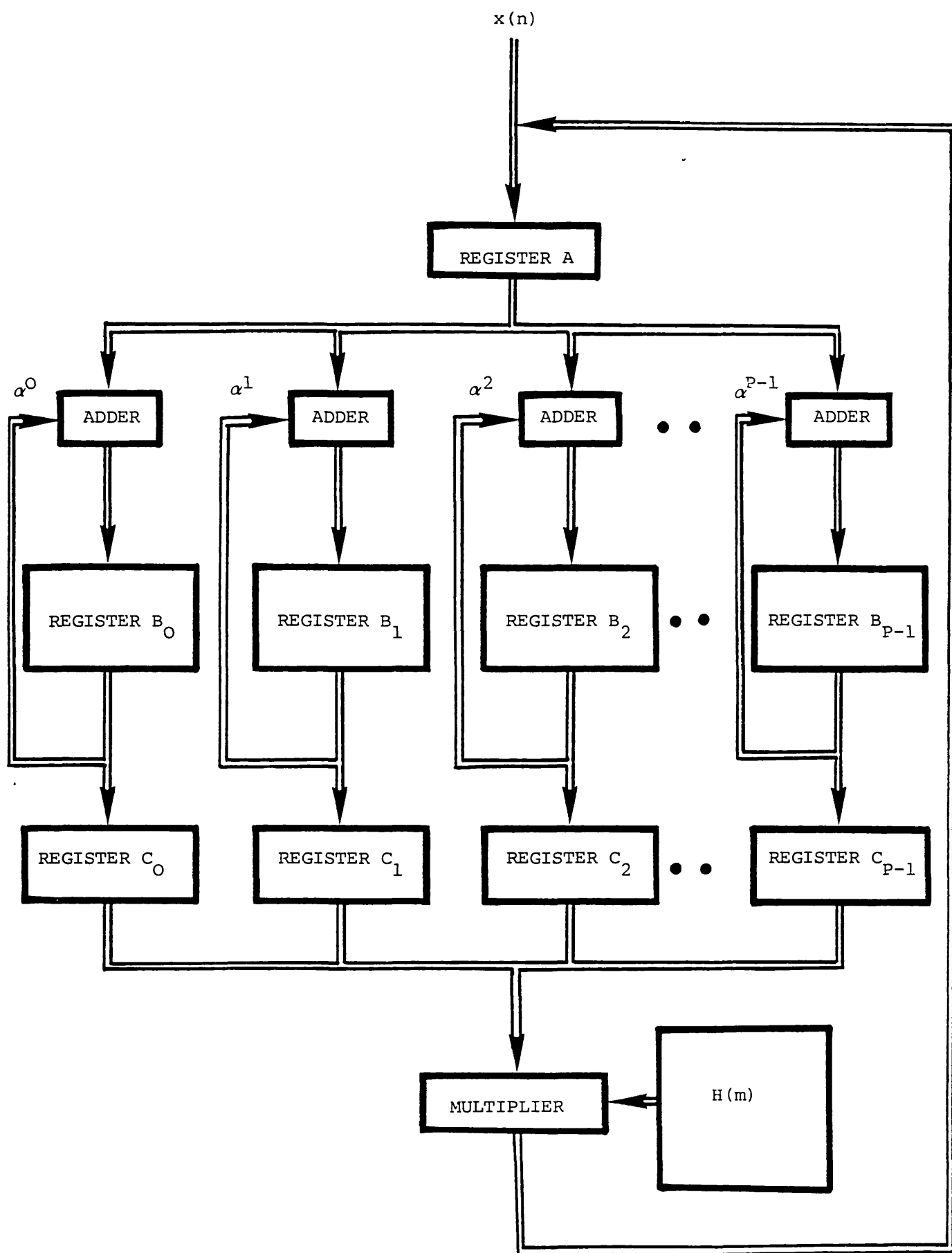


Figure 5.5 High Speed Parallel Mersenne Number Convolver

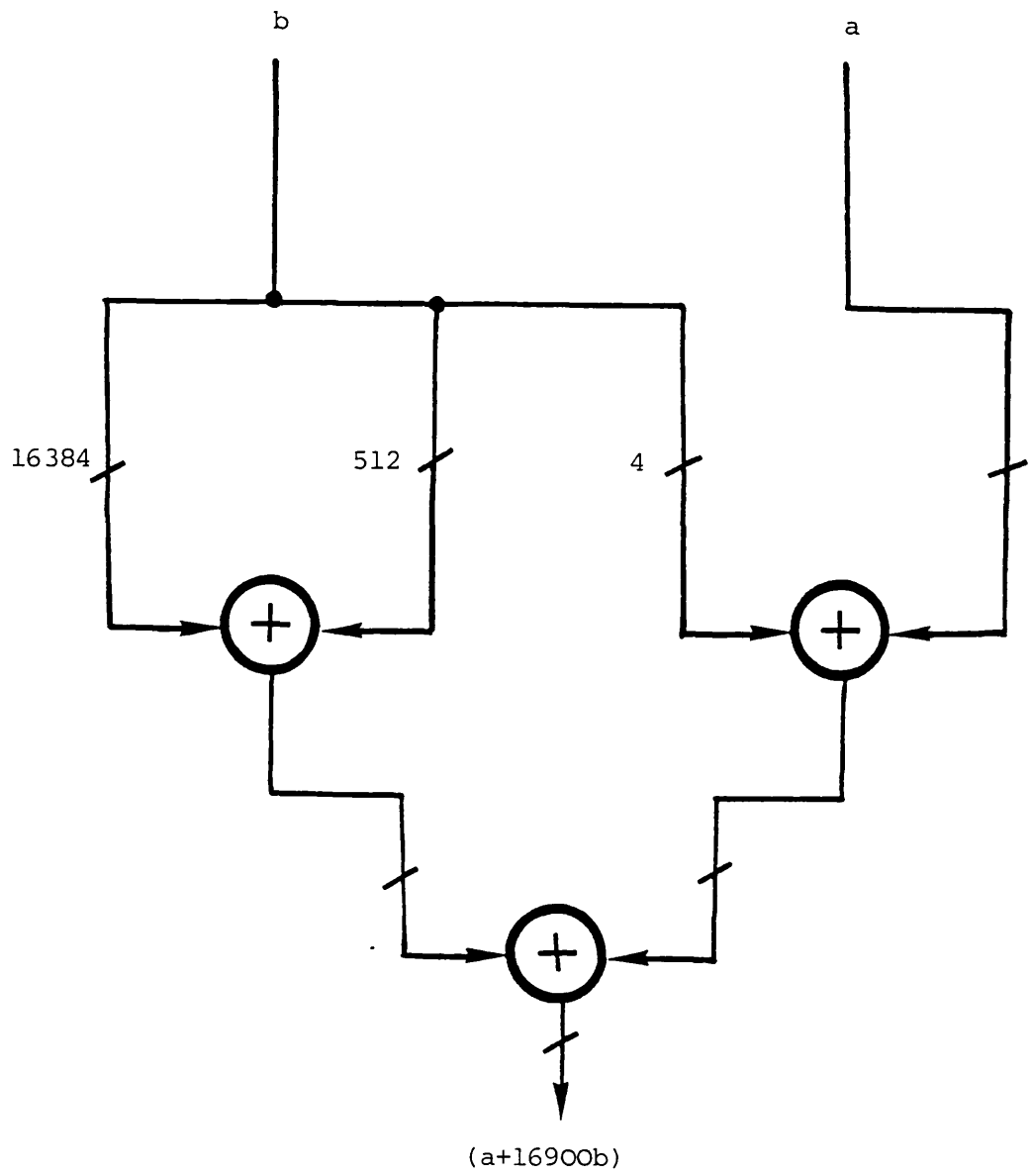


Figure 5.6 Block Diagram for the Implementation of $(a + 16900b)$

CHAPTER SIX EFFICIENT HARDWARE REALISATION OF NUMBER
THEORETIC TRANSFORM CONVOLVERS USING ROMs

Advances in fabrication technology have produced high-speed and inexpensive read-only-memories which in turn lead to further possibilities for designing new and efficient algorithms for the computation of both long and short cyclic convolutions using NTTs; this is a point originally suggested by Pollard[59]. Jullien and Miller[60,61] have also proposed to implement the NTT with a sequence length of 128 using arrays of ROMs. However, in order to achieve a reasonable dynamic range, an extension field (together with two finite fields with real roots of unity), $GF(q^2)$ with Gaussian integers as root of unity has been used. Hence full 'complex' calculation has to be realised for this modulo base. Furthermore, the ROM size required for this realisation technique is very large. Nagpal, Jullien and Miller[62] have recently modified this technique by using only moduli of the form $(4k+1)$, rather than using the second degree Galois field. However, their method still requires a very large ROM size[60-63] and a relatively complicated technique for the computation of α (the root of unity) multiplication has been used.

In this chapter we present the results of our study into using ROMs to realize NTT convolvers for fast cyclic convolutions. We first generalize eqn. 5.6 to suit any Number Theoretic Transform with prime sequence length, P , and suggest using ROM structures to make the implementation. We then propose a new type of Number Theoretic Transform for the fast implementation of cyclic convolutions. This

new realisation technique gives a very flexible choice of short and long sequence lengths of the transform. The ROM size required is reasonably small and is well within practical limits.

6.1 Prime Sequence Length Number Theoretic Transforms

The general equation for a length-N Number Theoretic Transform and its inverse defined over the field or ring of integers modulo M can be written as follows:

$$X(k) = \left\langle \sum_{n=0}^{N-1} x(n) \alpha^{nk} \right\rangle_M \quad (6.1)$$

$$x(n) = \left\langle \frac{1}{N} \sum_{k=0}^{N-1} X(k) \alpha^{-nk} \right\rangle_M \quad (6.2)$$

for $k, n=0, 1, \dots, N-1$

The transform pair of eqns. 6.1 and 6.2 has a circular convolution property if $\{\alpha, M, N\}$ are chosen on the ring or field of integers according to eqn. 2.24.

Let us consider the modulus of eqns. 6.1 and 6.2 to be a prime number, q , i.e. $M = q$. Let us also choose a prime sequence length, P . This length-P Number Theoretic Transform pair can now be written as follows:

$$X(k) = \left\langle \sum_{n=0}^{P-1} x(n) \alpha^{nk} \right\rangle_q \quad (6.3)$$

$$x(n) = \left\langle \sum_{k=0}^{P-1} x(k) \alpha^{-nk} \right\rangle_q \quad (6.4)$$

for $k, n=0, 1, \dots, P-1$

The relationship between P and q is given by [see eqn. 2.29]:

$$P \mid (q-1) \quad (6.5)$$

For example $q = 41$ and $P = 5$ form a possible selection of q and P , for $5 \mid 40$ and both 5 and 41 are prime numbers. In this case one possible root of unity is 10.

Let $nk=m$ for $k=1, 2, \dots, P-1$. We can write

$$n = \langle mk^{-1} \rangle_P$$

Hence eqn. 6.3 can be written as follows:

$$X(0) = \left\langle \sum_{n=0}^{P-1} x(n) \right\rangle_q \quad \text{and} \quad (6.6)$$

$$X(k) = \left\langle \sum_{m=0}^{P-1} x(\langle mk^{-1} \rangle_P) \alpha^m \right\rangle_q \quad (6.7)$$

for $k=1, 2, \dots, P-1$

This is the same approach suggested in chapter 5 for the hardware realisation of Mersenne Number Transforms for fast digital convolution and so eqns. 6.6 and 6.7 are the generalization of eqns. 5.4 and 5.5. Recall also eqn. 5.6:

$X(k)$

$$= x(0) + [[\dots [x(\langle k^{-1}(P-1) \rangle_p) \alpha + x(\langle k^{-1}(P-2) \rangle_p)] \alpha + x(\langle k^{-1}(P-3) \rangle_p)] \alpha \dots + x(\langle k^{-1}.2 \rangle_p)] \alpha + x(\langle k^{-1}.1 \rangle_p)] \alpha \quad (6.8)$$

In general, the value of α may not have a simple binary representation. Hence the multiplication of α in eqn. 6.8 could mean a general multiplication. However, the multiplier α is of constant value for the whole set of procedures of eqn. 6.8 and the modulo q multiplication can be carried out using a ROM lookup table. The time for one multiplication is then equivalent to one ROM access time and the size of ROM required is only $q \log_2 q$ bits.

Fig. 6.1 shows the hardware structure for the 5-point modulo 41 Number Theoretic Transform processor. The binary counter is used to latch ROM 1 to output addresses of $x(n)$'s to be multiplied by α in ROM 2.

Exactly the same circuit can be used for the computation of the forward and inverse Number Theoretic Transform, except that the binary counter counts up for the forward transform whilst for the inverse transform it counts down.

The memory size required for the computation of a P -point convolution with modulo q Number Theoretic Transform by the present technique is shown below:

ROM Size		
RAM Size	for locations	ROM Size
for $x(n)$'s	of $x(n)$'s	for α
Pxq -bit	$P^2 x \log_2 P$ -bit	$qx \log_2 q$ -bit

Usually, in our design (see below) P is chosen to be much smaller than q , and so the first and second terms of the above expression are much smaller than the third term. Furthermore, the total number of modulo P additions (and also the total number of ROM access) is equal to $2P(P-1)$ or the number of modulo P additions per point is $2(P-1)$. There are also P modulo- P multiplications (this means that there is one multiplication per point) for finding the term by term products of the transformed $x(n)$'s and the transformed system function. This multiplication can be computed by a general multiplier or a modulo P quarter square multiplier[see chapter 4].

Apparently, a practical NTT convolver can be constructed by using sufficiently large values of P and q , provided

(i) P and q are prime numbers

(ii) $q = NP + 1$,

where N is any integer

However, in actual practice, when q is sufficiently large for a practical implementation the ROM size might be too large. Furthermore, the number of modulo additions per point may also be too large for long sequence lengths, P . In order to solve these

difficulties, some multi-dimensional techniques and composite modulo bases are used to construct new NTTs for the efficient implementation of digital convolvers in the next section. Number Theoretic Transforms, with short modulo base (q), are also suggested to form the basic modules for our design.

6.2 Practical Number Theoretic Transforms

For a direct implementation of the Number Theoretic Transform, if the modulo base $M = q \approx 2^{16}$, the required ROM size is not less than 64Kx16-bit; if the modulo base $M = q \approx 2^{20}$, the required ROM size is not less than 1Mx20-bit. Both cases require a large amount of ROM, especially the latter case which represents a totally unacceptable situation. This problem can be resolved by using a composite modulo base M in which the Number Theoretic Transform is defined on the ring of integers modulo M . The arithmetic for the convolution is done mainly over the constituent fields of the ring, rather than directly over the ring of integers itself. The final results are then reconstructed by using the Chinese Remainder Theorem (CRT)[see chapter 2]. This technique provides a great reduction in ROM size. Higher computation speed can also be achieved through the use of high degree of parallelism and an absence of carry information requirements. Recall the fact that if $M = \prod_{i=1}^{\ell} q_i^{r_i}$, where q_i are prime factors of M and ℓ is the number of prime factors of M , then from eqn. 2.29

$$N \mid (q_i - 1) \quad \text{for all } q_i \quad (6.9)$$

Eqn. 6.9 implies that the transform length N must divide all $(q_i - 1)$ hence q_i must be of the form

$$q_i = Na_i + 1 \quad (6.10)$$

for $i=1,2,\dots,$

where a_i is any integer which satisfies eqn. 6.10

The choice of N can significantly affect the number of operations of the Number Theoretic Transform. Previously, the main concern of other authors has been to look for transform lengths of powers of two[40]. However, if we use look-up tables stored in ROMs to implement the multiplication by α^n , where $n = 1,2,\dots,N-1$, the ROM size becomes too large for the practical realisation in most cases. An alternative approach is to use indices and submodular decompositions to produce a manageable ROM size and hardware[60-62]. This approach is still relatively complicated and extra steps are necessary to carry out the conversion.

In our design, we use the same basic idea as are discussed in the previous section. This requires only one ROM access time for each multiplication by α^n . In order to reduce the number of operations for the computation of the transform, N is now chosen to be a composite number. Consider the case when N is equal to the product of h prime numbers, $p_1, p_2, \dots, p_h$, that is $N = \sum_{i=1}^h p_i$. Eqn. 6.1 can be converted into a multi-dimensional form[68],

$$X(k_h, \dots, k_2, k_1) = \sum_{n_h=0}^{p_h-1} \dots \sum_{n_2=0}^{p_2-1} \sum_{n_1=0}^{p_1-1} x(n_h, \dots, n_2, n_1) \alpha_1^{n_1 k_1} \alpha_2^{n_2 k_2} \dots \alpha_h^{n_h k_h} \quad (6.11)$$

for $k_1, k_2, \dots, k_h = 0, 1, \dots, (p_1-1), (p_2-1), \dots, (p_h-1)$ respectively

where $\alpha_r = \alpha^{(N/p_r)}$, for $r=1, 2, \dots, h$

and the mapping is

$$n = \left\langle \sum_{i=1}^h L_i n_i \right\rangle_N \quad (6.12)$$

$$k = \left\langle \sum_{i=1}^h L_i L_i^{-1} n_i \right\rangle_N \quad (6.13)$$

where $L_i = N/p_i$

This multi-dimensional equation can be implemented by a method similar to the Winograd Fourier Transform Algorithm(WFTA). Since the length, p_i of each dimension is a prime number, each dimension of eqn. 6.11 can be expressed in a recursive filter form and can be realised by the hardware structure suggested in the previous section. The total number of modulo M additions required by this method for the computation of N-point NTT is

$$A = \sum_{i=1}^h L_i A_i$$

where A_i is the number of Mod. M additions
for length p_i NTT

From the previous section, $A_i = p_i(p_i-1)$ and so

$$\begin{aligned} A &= \sum_{i=1}^h L_i p_i (p_i - 1) \\ &= N \sum_{i=1}^h (p_i - 1) \end{aligned}$$

Then the number of modulo M additions per point (A') for an N-point cyclic convolution (including the inverse NTT) becomes

$$A' = 2 \sum_{i=1}^h (p_i - 1) \quad (6.14)$$

If $N = P = 31$, the number of additions required per point is $2 \times (31-1) = 60$. However, if $N = p_1 p_2 = 35$, the number of additions required per point is $2(4+6) = 20$. Hence it is clear from eqn. 6.14 that the number of additions can be reduced if we choose N to be the product of a large number of small primes, p_i 's. The smallest primes are $\{2, 3, 5, 7, 11, \dots\}$. This idea is used to construct our efficient NTTs for the rest of this chapter. That is N is chosen to be of the form

$$N = r.2.3.5.7. \dots \quad (6.15)$$

where r is usually equal to 1

or a small number

Combining eqns. 6.10 and 6.15, we get a general form for the choice of the modulo bases q_i and the transform length N. That is

$$q_i = (a_i.2.3.5.7. \dots) + 1 \quad (6.16)$$

where a_i is a small number

Now let us consider $N = 210 (=2 \cdot 3 \cdot 5 \cdot 7)$. In this case $211 (=2 \cdot 3 \cdot 5 \cdot 7 + 1 = q_1)$, $421 (=2 \cdot 2 \cdot 3 \cdot 5 \cdot 7 + 1 = q_2)$ and $631 (=3 \cdot 2 \cdot 3 \cdot 5 \cdot 7 + 1 = q_3)$ are three possible choices of q_i 's. These form four possible choices of overall moduli for the NTT, that is

M	Prime factors	Possible α
88831	211×421	17
133141	211×631	48
265651	421×631	74
56052361	$211 \times 421 \times 631$	187

The above table also includes some possible choice of α 's for these NTTs. Note also that in all cases, we should check the choice of $\{\alpha, N, M\}$ against eqns. 2.24 and 6.9 for correctness.

Let us consider the NTT with 88831 as the modulo base and 210 as the transform length. 17 is a root of unity for this transform. Hence the NTT transform pair becomes,

$$X(k) = \left\langle \sum_{n=0}^{209} x(n) 17^{nk} \right\rangle_{88831}$$

$$x(n) = \left\langle 88408 \sum_{k=0}^{209} X(k) 17^{-nk} \right\rangle_{88831}$$

As shown previously, this NTT convolution pair can be implemented in parallel with arithmetic over Galois fields modulo the primes 211 and 421 respectively. That is

$$X'(k) = \langle \sum_{n=0}^{209} x(n) 17^{nk} \rangle_{211} \quad (6.17)$$

$$x'(n) = \langle \sum_{k=0}^{209} X'(k) 17^{-nk} \rangle_{211} \quad (6.18)$$

and

$$X''(k) = \langle \sum_{n=0}^{209} x(n) 17^{nk} \rangle_{421} \quad (6.19)$$

$$x''(n) = \langle \sum_{k=0}^{209} X''(k) 17^{-nk} \rangle_{421} \quad (6.20)$$

The final results can then be reconstructed by the Chinese Remainder Theorem. Eqns. 6.17 and 6.19 can be written in a multi-dimensional form as:

$$X(k_4, k_3, k_2, k_1)$$

$$= \sum_{n_4=0}^{2-1} \sum_{n_3=0}^{3-1} \sum_{n_2=0}^{5-1} \sum_{n_1=0}^{7-1} x(n_4, n_3, n_2, n_1) \alpha_1^{n_1 k_1} \alpha_2^{n_2 k_2} \alpha_3^{n_3 k_3} \alpha_4^{n_4 k_4} \quad (6.21)$$

$$\text{for } k_4=0,1$$

$$k_3=0,1,2$$

$$k_2=0,1,2,3,4$$

$$k_1=0,1,\dots,6$$

and

$\alpha_1 = 13015$	$\alpha_1 = 144$	$\alpha_1 = 385$
$\alpha_2 = 30666$	$\alpha_2 = 71$	$\alpha_2 = 354$
$\alpha_3 = 14755$	$\alpha_3 = 196$	$\alpha_3 = 20$
$\alpha_4 = -1$	$\alpha_4 = -1$	$\alpha_4 = -1$
mod 88830	mod 211	mod 421

The mapping is

$$n = \langle 105n_4 + 70n_3 + 42n_2 + 30n_1 \rangle_{210} \quad (6.22)$$

and $k = \langle 105k_4 + 70k_3 + 126k_2 + 120k_1 \rangle_{210} \quad (6.23)$

Let us now find the ROM size required for α 's for this example. The multiplication by α_4 is simply a subtraction, and so no look-up table is required. Therefore the ROM size required for this NTT becomes

$$\begin{aligned} & 3 \times 211 \times \log_2 211 + 3 \times 421 \times \log_2 421 \text{ bits} \\ & = 0.633K \times 8 + 1.263K \times 9 \text{ bits} \end{aligned}$$

This is small enough for any kind of realisation. In general, the ROM size required for α 's is

$$(h-1) \sum_{i=1}^{\ell} q_i \times \log_2 q_i \text{ bits} \quad (6.24)$$

where h = number of prime factors of N

ℓ = number of prime factors of M

Table 6.1 gives the possible cyclic convolution lengths (and other details) computed by NTT with modulus $M = 56052361 (= 211 \times 421 \times 631)$ and

in this case the required ROM size is

$$0.633K \times 8 + 1.263K \times 9 + 1.893K \times 10 \text{ bits}$$

A further remark about this table is that the number of additions per point (for convolution) refers to the number of additions per point for the overall modulo base. For a parallel implementation, all moduli require the same number of additions per point and 2 more modulo M additions per point are required for the reconstruction of the results by the CRT.

We have used $N = 210$ as an example for our discussion. This does not mean that 210 is the only transform length for the NTT modulo 88831. Actually 210 is the maximum transform length for this NTT and the hardware structure works equally well for transform lengths which divide 210, that is the transform lengths inside the set

$$\{2, 3, 5, 6, 7, 10, 14, 15, 21, 30, 35, 42, 70, 105, 210\}.$$

This gives, therefore, a large set of transform lengths for the same hardware structure.

The maps in eqns. 6.22 and 6.23 (or in general eqns. 6.12 and 6.13) seem complicated. However, for the actual implementation the mapping of eqn. 6.21 (or eqn. 6.11 in general) can be done simply in a two-dimensional form; it need not be done in a multi-dimensional form. The four-dimensional NTT in eqn. 6.21 is calculated by 30 length-7 NTTs, then 42 length-5 NTTs, then 70 length-3 NTTs and then 105 length-2 NTTs. For example, for the calculation of the first

dimension, the mapping for loading data is

$$\begin{aligned} n &= \langle 7n' + 30n_1 \rangle_{210} \\ \text{for } n' &= 0, 1, \dots, 29 \\ n_1 &= 0, 1, \dots, 6 \end{aligned}$$

Generally, we can write

$$\begin{aligned} n &= \langle N_r n' + R n_r \rangle_N & (6.25) \\ \text{for } n' &= 0, 1, \dots, (R-1) \\ n_r &= 0, 1, \dots, (N_r-1) \\ \text{where } R &= N/N_r \text{ for } r\text{th dimension} \end{aligned}$$

The computation can be done in-place and in-order simply by using eqn. 6.25 for the loading of data and using the following equation for the retrieval of data[91-92]

$$\begin{aligned} k &= \langle N_r n' + R R^{-1} n_r \rangle_N & (6.26) \\ \text{where } \langle R R^{-1} \rangle_{N_r} &= 1 \text{ for } r\text{th dimension} \end{aligned}$$

In the above example, the mapping for the retrieval of data for the computation of the first dimension becomes

$$k = \langle 7n' + 120k_1 \rangle_{210}$$

Essentially, eqn. 6.26 means that the unscrambling of data is done immediately after the computation of each dimension and so the final mapping becomes eqn. 6.13. An alternative approach to unscrambling the results is to unscramble the computed NTT at the end of the computation[91]. However, this requires some extra memory space for

the procedure.

On the other hand, if we want to have even longer sequence lengths, we may choose $N = 2310 (=2 \times 3 \times 5 \times 7 \times 11)$. In this case the set of possible transform length is

$$\{2, 3, 5, 6, 7, 10, 11, 14, 15, 21, 22, 30, 33, 35, 42, 55, 66, 70, 77, 105, \\ 110, 154, 165, 210, 231, 330, 385, 462, 770, 1155, 2310\}.$$

The possible prime modulo bases are

$$\begin{aligned} q_1 &= 2311 = 2 \times 3 \times 5 \times 7 \times 11 + 1 \\ q_2 &= 4621 = 2 \times 2 \times 3 \times 5 \times 7 \times 11 + 1 \\ q_3 &= 9241 = 4 \times 2 \times 3 \times 5 \times 7 \times 11 + 1 \end{aligned}$$

The overall modulo bases can then be 10679131 ($=2311 \times 4621$), 21355951 ($=2311 \times 9241$), 42702661 ($=4621 \times 9241$) or 98685849571 ($=2311 \times 4621 \times 9241$). These give output dynamic ranges of 2^{23} , 2^{24} , 2^{25} and 2^{36} respectively. The dynamic ranges are very large and the sequence lengths are long enough for any kind of application. However, in these cases, the memory sizes required for the α multiplications are unavoidably larger than for the previous cases.

For example, the memory required for the modulo 10679131 transform is

$$\begin{aligned} &4[2311 \times \log_2 2311 + 4621 \times \log_2 4621] \text{ bits} \\ &\approx (9.2K \times 12 + 18.5K \times 13) \text{ bits} \end{aligned}$$

This is relatively larger than the previous case, but it still falls

into the range of practical limits.

Tabling 6.2 lists some possible modulo bases and other details for this new approach to the computation of cyclic convolutions using NTTs. It can be seen that all the ROM sizes required are within practical limits and there is a large choice of modulo bases (or dynamic ranges) and sequence lengths.

6.3 Conclusion

Some basic moduli for constructing NTTs have been suggested and used as the basic word-lengths to construct large and practical word-lengths for cyclic convolutions. This list of short basic moduli is by no means exhaustive in that the user may construct other short moduli to form other long modulo bases as required. It can also be seen from this chapter that there is a large choice of sequence lengths for this proposed NTTs. Hence this new Number Theoretic Transform can relax completely the sequence length and wordlength constraints of the NTT.

From the practical point of view, this method requires a small number of additions per point and the memory required is much less than previous reported methods[60-63]. Furthermore, due to the similarities between modules and because exactly the same processes are required for different dimensions, the control of data flow is extremely simple and the whole computation can be done in-place and in-order.

It is also interesting to point out that almost the same technique can be applied to the computation of DFTs. However, in this case complex roots of unity have to be used. This causes an increase in complexity of the control design, but in this case no modulo arithmetic is required.

Table 6.1

Sequence		Inverse N^{-1}	Possible α	No. of Additions per Input	
Length	Factors			Point (for convolution)	Dynamic Range
2	2	2 802 6181	56 052 360	2	3743
3	3	3 736 8241	26 380 260	4	3056
5	5	44 841 889	332 066 54	8	2367
6	2x3	46 710 301	26 380 261	6	2161
7	7	48 044 881	15 870 091	12	2000
10	2x5	50 447 125	14 942 199	10	1674
14	2x7	52 048 621	4 159 531	14	1414
15	3x5	52 315 537	35 991 673	12	1366
21	3x7	53 383 201	11 078 529	16	1155
30	2x3x5	54 183 949	32 443 488	14	966
35	5x7	54 450 865	06 767 890	20	984
42	2x3x7	54 717 781	31 809 188	18	816
70	2x5x7	55 251 613	65 392 03	22	632
105	3x5x7	55 518 529	34 969	24	516
210	2x3x5x7	55 785 445	187	26	363

Possible Sequence Lengths for Cyclic Convolutions computed by
the NTT with modulus $M=56\,052\,361 (=211 \times 421 \times 631)$

Table 6.2

M	Factors		Factors		N^{-1}	ROM Size (for Max.N)	Output Dynamic Range
	of M	Max. N	of N	Possible α			
88831	211	210	2,3,5,7	17	88408	0.633Kx8	$2^{16.4}$
	421					1.263Kx10	
133141	211	210	2,3,5,7	48	132507	0.633Kx8	$2^{17.0}$
	631					1.893Kx10	
265651	421	210	2,3,5,7	74	264386	1.263Kx9	$2^{18.0}$
	631					1.893Kx10	
56052361	211	210	2,3,5,7	187	55785445	0.633Kx8	$2^{25.7}$
	421					1.263Kx9	
	631					1.893Kx10	
10679131	2311	2310	2,3,5,	3	10674508	9.244Kx12	$2^{23.3}$
	4621		7,11			18.484Kx13	
21355951	2311	2310	2,3,5,	75	21346706	9.244Kx12	$2^{24.3}$
	9241		7,11			36.964Kx14	
42702661	4621	2310	2,3,5,	196	42684175	18.484Kx13	$2^{25.3}$
	9241		7,11			36.964Kx14	
98685849571	2311	2310	2,3,5,	2302	98643128424	9.244Kx12	$2^{36.5}$
	4621		7,11			18.484Kx13	
	9241					36.964Kx14	

Possible Moduli for the Computation of Cyclic Convolutions using the proposed NTT.

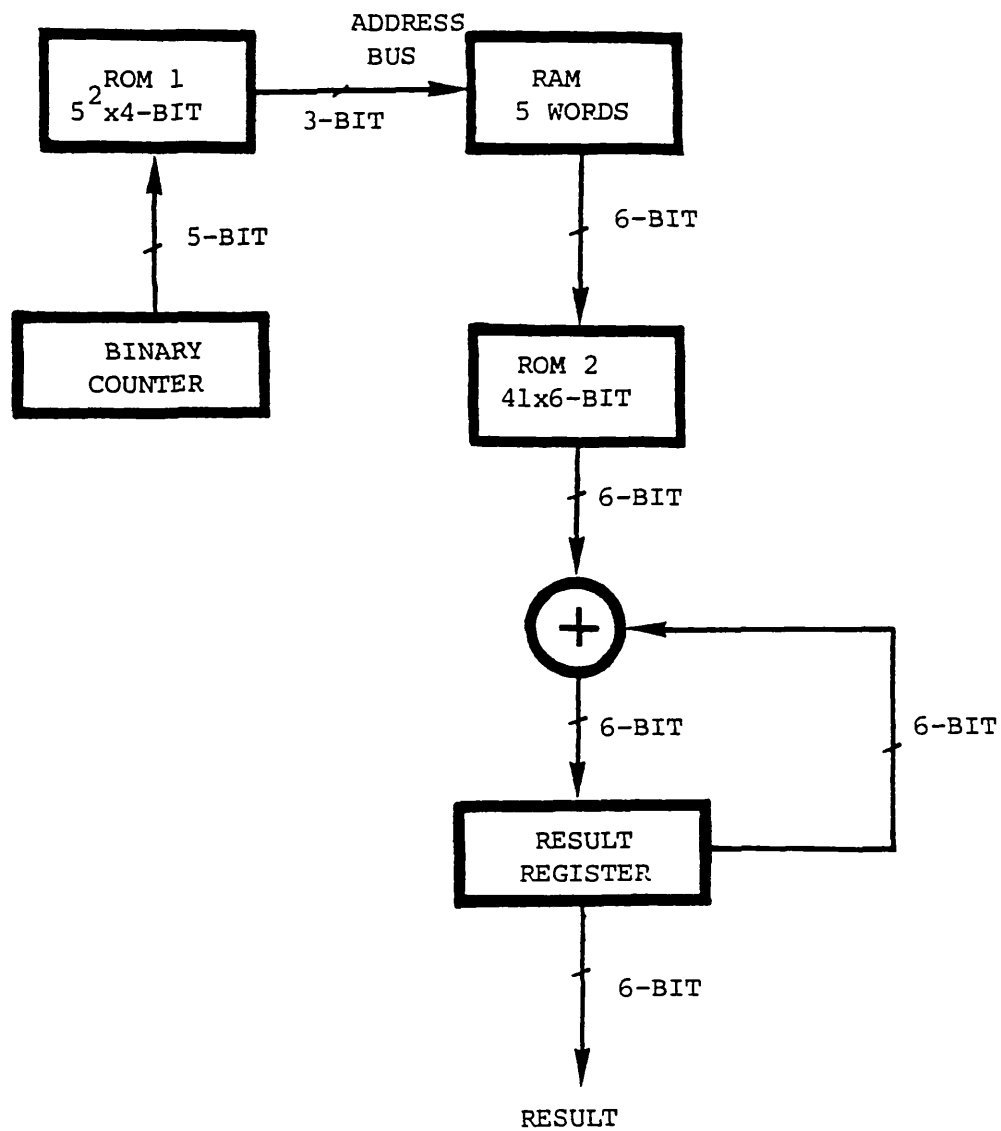


Figure 6.1 Hardware Structure of a Modulo 41, Length-5 NTT Processor

CHAPTER SEVEN VERY FAST DISCRETE FOURIER TRANSFORM USING
NUMBER THEORETIC TRANSFORM

The Discrete Fourier Transform plays a key role in Digital Signal Processing because it can be used as a mathematical tool to describe the relationship between the time domain and frequency domain representation of discrete signals. In the previous chapters we have shown that Number Theoretic Transforms can be used to implement cyclic convolutions very effectively. It is the purpose of this and the following chapters to show that the same basic hardware structures can be used to implement the Discrete Fourier Transform.

In this chapter it is shown that Number Theoretic Transforms (NTT) can be used to compute the Discrete Fourier Transform (DFT) very efficiently. By noting some simple properties of number theory and the DFT, the total number of real multiplications for a length- q DFT is reduced to $(q-1)$, where q is a prime number. This requires less than one real multiplication per point. For a proper choice of transform length and NTT, the number of shift-adds per point is approximately the same as the number of additions required for FFT algorithms. In chapters 8 and 9, we expand the ideas presented in this chapter and discuss the details of the analysis of using the Fermat Number Transform and the Mersenne Number Transforms to implement DFTs.

A brief account of the contents of the present chapter has been presented in [94] whilst a complete account is given in [95].

7.1 Theory

Let g be a primitive root that generates all nonzero elements inside the field modulo q . Consider now an N -point Discrete Fourier Transform,

$$Y(k) = \sum_{n=0}^{N-1} x(n)W_0^{nk} \quad (7.1)$$

where $k = 0, 1, \dots, N-1$ and

$$W_0 = e^{-j\frac{2\pi}{N}}$$

If N is a prime number q , then eqn. 7.1 can be reordered[18] in the following form

$$Y(0) = \sum_{n=0}^{q-1} x(n) \quad \text{and} \quad (7.2)$$

$$Y(\langle g^k \rangle_q) = x(0) + \sum_{n=1}^{q-1} x(\langle g^{-n} \rangle_q) W_0^{\langle g^{k-n} \rangle_q} \quad (7.3)$$

for $k = 1, 2, \dots, q-1$

We can write eqn. 7.3 as

$$Y(\langle g^k \rangle_q) = x(0) + X(\langle g^k \rangle_q) \quad (7.4)$$

where

$$X(\langle g^k \rangle_q) = \sum_{n=1}^{q-1} x(\langle g^{-n} \rangle_q) W_0^{\langle g^{k-n} \rangle_q} \quad (7.5)$$

for $k = 1, 2, \dots, q-1$

Eqn. 7.5 represents a backward circular convolution of length-(q-1).

That is

$$\{x(g^{-1}), x(g^{-2}), \dots, x(g^{-q+1})\} \circledast \{w_0^g, w_0^g, \dots, w_0^{g^{q-2}}\} \quad (7.6)$$

where $\circledast$ means circular convolution and the

subscripts and indices are modulo q

Let us now define

$$X_k = X(\langle g^{k+1} \rangle_q) \quad (7.7)$$

$$W_n = W_0 \langle g^n \rangle_q \quad (7.8)$$

$$x_n = x(\langle g^{-(n+1)} \rangle_q) \quad (7.9)$$

for $k = 0, 1, \dots, q-2$

$n = 0, 1, \dots, q-2$

Hence eqns. 7.5 and 7.6 become

$$X_k = \sum_{n=0}^{q-2} x_n W_{k-n} \quad (7.10)$$

for $k = 0, 1, \dots, q-2$ and

$$\{x_0, x_1, \dots, x_{q-2}\} \circledast \{w_0, w_1, \dots, w_{q-2}\} \quad (7.11)$$

The Number Theoretic Transform can now be applied to compute the cyclic convolution sum of these two sequences. Thus we can write

$$X'_m = \left\langle \sum_{n=0}^{q-2} x_n \alpha^{mn} \right\rangle_M \quad \text{and} \quad (7.12)$$

$$W'_m = \left\langle \sum_{n=0}^{q-2} w_n \alpha^{mn} \right\rangle_M \quad (7.13)$$

for $m = 0, 1, \dots, q-2$

The results can then be obtained by the inverse transform of the products, $X'_m W'_m$'s. That is

$$X_k = \left\langle \frac{1}{q-1} \sum_{m=0}^{q-2} X'_m W'_m \alpha^{-mk} \right\rangle_M \quad (7.14)$$

for $k = 0, 1, \dots, q-2$

Recall that all W'_m 's are complex numbers, hence apparently the total number of multiplications for a real sequence of length- q (to find all $X'_m W'_m$'s) for this method is $2(q-1)$. However the sequence $\{w_0^0, w_0^1, \dots, w_0^{q-2}\}$ can actually be written as

$$\{w_0^0, w_0^1, \dots, w_0^{\frac{q-1}{2}-1}, (w_0^0)^*, (w_0^1)^*, \dots, (w_0^{\frac{q-1}{2}-1})^*\} \quad (7.15)$$

where $*$ denotes complex conjugates.

Therefore the sequence $\{w_0, w_1, \dots, w_{q-2}\}$ can be written as

$$\{w_0, w_1, \dots, w_{\frac{q-1}{2}-1}, w_0^*, w_1^*, \dots, w_{\frac{q-1}{2}-1}^*\}. \quad (7.16)$$

Note also that $\text{Real}(W_n) = \text{Real}(W_n^*)$ and $\text{Imag}(W_n) = -\text{Imag}(W_n^*)$. Hence

$$\text{Real}(W_{n + \frac{q-1}{2}}) = \text{Real}(W_n) \quad (7.17)$$

$$\text{Imag}(W_{n + \frac{q-1}{2}}) = -\text{Imag}(W_n) \quad (7.18)$$

$$\text{for } n = 0, 1, \dots, (\frac{q-1}{2} - 1)$$

In view of these relationships, eqn. 7.13 can be written as

$$\begin{aligned} W'_m &= \left\langle \sum_{n=0}^{\frac{q-1}{2}-1} W_n \alpha^{mn} + \sum_{n=\frac{q-1}{2}}^{q-2} W_n \alpha^{mn} \right\rangle_M \\ &= \left\langle \sum_{n=0}^{\frac{q-1}{2}-1} W_n \alpha^{mn} + \sum_{n=0}^{\frac{q-1}{2}-1} W_{n+\frac{q-1}{2}} \alpha^{(n+\frac{q-1}{2})m} \right\rangle_M \end{aligned}$$

and since $\alpha^{\frac{q-1}{2}} = -1$, we can write

$$W'_m = \left\langle \sum_{n=0}^{\frac{q-1}{2}-1} W_n \alpha^{mn} + (-1)^m \sum_{n=0}^{\frac{q-1}{2}-1} W_{n+\frac{q-1}{2}} \alpha^{mn} \right\rangle_M \quad (7.19)$$

On combining eqns. 7.17, 7.18 and 7.19, it is clear that

$$\begin{aligned} \text{Real}(W'_m) &= \text{Real}(\left\langle \sum_{n=0}^{\frac{q-1}{2}-1} W_n \alpha^{mn} \right\rangle_M) \text{ for } m=\text{even} \\ &= 0 \quad \text{for } m=\text{odd} \end{aligned} \quad (7.20)$$

$$\begin{aligned} \text{Imag}(W'_m) &= 0 \quad \text{for } m=\text{even} \\ &= \text{Imag}(\left\langle \sum_{n=0}^{\frac{q-1}{2}-1} W_n \alpha^{mn} \right\rangle_M) \text{ for } m=\text{odd} \end{aligned} \quad (7.21)$$

Eqns. 7.20 and 7.21 are very important in practical implementations. This is not primarily due to the fact that the number of shift-adds (α multiplications if α is a simple combination of power of 2) reduces by a factor of 2 for the calculation of W'_m 's, since all W'_m 's should be pre-calculated for a hardware implementation; it is due, however, to the fact that the total number of real multiplications forming $X'_m W'_m$'s reduces from $(2q-2)$ to $(q-1)$. This gives less than one (actually $1-\frac{1}{q}$) multiplication per point for the DFT of a real sequence of length- q . The number of additions required in this technique is evaluated below.

The shift-adds required to compute W'_m 's are not counted since these quantities can be pre-calculated and stored in ROM in a hardware implementation or stored in the program for a software implementation. The total number of shift-adds required for transforming $\{x_0, x_1, \dots, x_{q-2}\}$ to $\{X'_0, X'_1, \dots, X'_{q-2}\}$ is $(q-1)(q-2)$. However, if we choose $(q-1)$ to be highly composite, an FFT-type of algorithm can be applied to effect the transformation. In particular if $(q-1)$ is a power of 2, the number of shift-adds is equal to $(q-1)\log_2(q-1)$.

Since the results are complex, two inverse transformations, one real and one imaginary, are required. As g and q are relatively prime, Euler's Theorem says

$$\langle g^{\Phi(q)} \rangle_q = 1$$

$$\text{If } \langle g^{q-1} \rangle_q = 1 \quad (7.22)$$

then

$$\langle g^{\frac{q-1}{2}} \rangle_q = \pm \sqrt{1} = \pm 1$$

$\langle g^{\frac{q-1}{2}} \rangle_q = 1$ is contradictory with g being a primitive root, because if g is a primitive root then g is of the order $\Phi(q) = q-1$, that is $q-1$ is the least integer such that $\langle g^{q-1} \rangle_q = 1$.

$$\text{Therefore} \quad \langle g^{\frac{q-1}{2}} \rangle_q = -1 \quad (7.23)$$

Since the sequence $\{x(n): n=0,1,\dots,q-1\}$ is real, recalling the symmetry of the real and the antisymmetry of the imaginary components of W_0^{nk} about the horizontal axis one may show, from eqn. 7.1 that:

$$Y(k) = Y^*(q-k) \quad (7.24)$$

where $k=0,1,\dots,\frac{q-1}{2}$

Substituting eqns. 7.22 and 7.23 into 7.5, we obtain

$$\begin{aligned} & \{X(\langle g^k \rangle_q) : k=1,2,\dots,q-1\} \\ &= \{X(\langle g^1 \rangle_q), X(\langle g^2 \rangle_q), \dots, X(\langle g^{\frac{q-1}{2}-1} \rangle_q), X(\langle g^{\frac{q-1}{2}} \rangle_q), X(\langle g^{\frac{q-1}{2}+1} \rangle_q), \\ & \quad X(\langle g^{\frac{q-1}{2}+2} \rangle_q), \dots, X(\langle g^{q-1} \rangle_q)\} \\ &= \{X(\langle g^1 \rangle_q), \{X(\langle g^2 \rangle_q), \dots, X(\langle g^{\frac{q-1}{2}-1} \rangle_q), X(-1), X(-g^1 \rangle_q), \\ & \quad X(-\langle g^2 \rangle_q), \dots, X(1)\} \} \end{aligned}$$

$$\begin{aligned}
 &= \{X(\langle g^1 \rangle_q), X(\langle g^2 \rangle_q), \dots, X(\langle g^{\frac{q-1}{2}-1} \rangle_q), X(\langle g^{\frac{q-1}{2}} \rangle_q), X(\langle g^1 \rangle_q)^*, \\
 &\quad X(\langle g^2 \rangle_q)^*, \dots, X(\langle g^{\frac{q-1}{2}} \rangle_q)^*\} \quad (7.25)
 \end{aligned}$$

Hence this equation shows that for an actual implementation, only $\{X(\langle g^1 \rangle_q), X(\langle g^2 \rangle_q), \dots, X(\langle g^{\frac{q-1}{2}} \rangle_q)\}$ need be computed, the other half of $X(\langle g^n \rangle_q)$'s can be found by taking the conjugate of these values.

Furthermore both real and imaginary parts of the sequences $\{X'_m W'_m, m=0,1,\dots,q-2\}$ are alternately zero, a length- $(q-1)$ inverse transformation can be formed by two length- $(\frac{q-1}{2})$ inverse transforms. Hence the number of shift-adds for the inverse transformation is $2(\frac{q-1}{2})(\frac{q-1}{2}-1) = (q-1)(\frac{q-1}{2}-1)$ in general, or is $2(\frac{q-1}{2})\log_2(\frac{q-1}{2}) = (q-1)[\log_2(q-1)-1]$ if $(q-1)$ is a power of 2. Eqn. 7.4 have to be used to calculate the final results. For the calculation of $Y(0)$, a total number of $(q-1)$ additions are required. However, the summation of $\{x(1), x(2), \dots, x(q-1)\}$ has been already obtained in the FNT transformed results of $\{x_n : n=0,1,\dots,q-2\}$. Hence only one extra addition (the addition of $x(0)$) is necessary for the computation of $Y(0)$. Because of the symmetry property of the DFT, $\frac{q-1}{2}$ additions are required to compute eqn. 7.4. Hence the overall number of shift-adds including the additions of $x(0)$ and the additions for $Y(0)$ becomes

$$(q-1)[2\log_2(q-1)-1] + 1 + \frac{q-1}{2} = 2(q-1)\log_2(q-1) - \frac{q-3}{2} \quad (7.26)$$

This figure of shift-adds is approximately equal to the number of real multiplications used in a FFT.

7.2 An Example

To illustrate the idea let us consider the DFT of the sequence $\{x(0), x(1), x(2), x(3), x(4)\}$, i.e. $N=q=5$. In this case the number 2 is a primitive root which is used to generate elements inside the field modulo 5. Hence the mapping for $\{x(n)\}$ in eqn. 7.5 and eqn. 7.6 is given by

$$\begin{aligned} \{x_n : n=1,2,3,4\} &= \{x(\langle g^{-n} \rangle_M)\}, :n=1,2,3,4\} \\ &= \{x(3), x(4), x(2), x(1)\} \text{ and} \end{aligned}$$

$$\{W_0^g, :n=0,1,2,3\} = \{W_0^1, W_0^2, W_0^4, W_0^3\}.$$

Hence eqn. 7.6 becomes

$$\begin{aligned} \{x(3), x(4), x(2), x(1)\} \otimes \{W_0^1, W_0^2, W_0^4, W_0^3\} \\ \{x(3), x(4), x(2), x(1)\} \otimes \{W_0^1, W_0^2, W_0^{1*}, W_0^{2*}\} \end{aligned} \quad (7.27)$$

This convolution sum can be computed by NTT. Now let us use a Fermat Number Transform (FNT) to make the calculation. Let $M=F_4=2^{16}+1$, $\alpha=2^8$ and of course $N=4$. Hence

$$\begin{bmatrix} X'_0 \\ X'_1 \\ X'_2 \\ X'_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2^8 & -1 & -2^8 \\ 1 & -1 & 1 & -1 \\ 1 & -2^8 & -1 & 2^8 \end{bmatrix} \begin{bmatrix} x(3) \\ x(4) \\ x(2) \\ x(1) \end{bmatrix} \quad (7.28)$$

$$\begin{bmatrix} W'_0 \\ W'_1 \\ W'_2 \\ W'_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2^8 & -1 & -2^8 \\ 1 & -1 & 1 & -1 \\ 1 & -2^8 & -1 & 2^8 \end{bmatrix} \begin{bmatrix} W_0^1 \\ W_0^2 \\ W_0^{1*} \\ W_0^{2*} \end{bmatrix} \quad (7.29)$$

where $W_0 = e^{-j\frac{2\pi}{5}}$

In order to use modulo arithmetic, the W_0 terms have to be normalized to integer values. Multiplying these terms by 90 and rounding off the results to integers, we obtain

$$\begin{bmatrix} W'_0 \\ W'_1 \\ W'_2 \\ W'_3 \end{bmatrix} = \begin{bmatrix} -90 + j0 \\ 0 + j38229 \\ 202 + j0 \\ 0 + j26964 \end{bmatrix}$$

This expression may be compared with eqns. 7.20 and 7.21 for agreement. Hence for the computation of $X'_m W'_m$, $m=0,1,2,3$ a total number of four real multiplications is sufficient. This is also the total number of multiplications required for a 5-point DFT. The total

number of real shift-adds required is $2(4)\log_2(4)-(5-3)/2=15$.

As we have seen, the length for the NTT is $(q-1)$, where $(q-1)$ is always an even number. Fermat Number Transforms, Pseudo Mersenne Transforms, Pseudo Fermat Transforms or any efficient transform with an even number as the transform length are suitable for the computation. However, for some very promising NTTs, the transform lengths may not be long enough or may not match this requirement. For example, an excellent choice of q is 257 which is prime and $N(=q-1)$ is highly composite and it might be possible to use an NTT to effect the convolution. The longest transform length (with $\sqrt{2}$ as the root of unity) for FNT with modulo base F_6 is 256 [see chapter 2]. Hence F_6 is a possible choice for the implementation. If one wishes to use a shorter word-length, F_5 for example, to effect the implementation the major problem is that the maximum transform length for the FNT with $M=F_5=2^{32}-1$ is 128 for $\alpha = \sqrt{2}$. However this problem may be resolved by using multi-dimensional techniques for convolutions [see chapter 3]. Since q is a prime number, it is also possible to combine Winograd's short DFTs to carry out the computation of long DFTs using multi-dimensional formulations. The major disadvantages of the method using NTT to calculate DFT are that special arithmetic (modulo arithmetic) and normally relatively large word lengths may have to be used for the major part of the calculation, a fact common to all Number Theoretic Transforms.

CHAPTER EIGHT COMPUTATION OF DISCRETE FOURIER TRANSFORM USING FERMAT NUMBER TRANSFORM

The transform length of the Fermat Number Transform is highly composite (it is in fact a power of two) and hence Fast Fourier Transform types of algorithm can be used for the computation. In this respect, it would appear that the FNT is the most prominent candidate for the computation of DFTs. However, for practical implementations, nearly all Number Theoretic Transforms are limited by sequence length constraints. This limits the application of FNTs to a small number of relatively short DFTs only, or indeed the application of Number Theoretic Transform for computing the Discrete Fourier Transform is limited as a whole.

In this chapter, we present the results of our study using Fermat Number Transform (FNT) to compute Discrete Fourier Transforms (DFT). Eight basic FNT modules are suggested and used as the basic sequence lengths to compute long DFTs. The number of multiplications per point is for most cases not more than one, whereas the number of shift-adds is for most cases also less than the number of additions in the Winograd Fourier Transform Algorithm and the Polynomial Transform. Thus the technique to be presented is very effective in computing Discrete Fourier Transforms.

The contents of this chapter have been published in [96].

8.1 Basic Fermat Number Transform Modules

A Fermat Number is defined as $F_t = 2^{2^t} + 1$, hence $F_t - 1 (= 2^{2^t})$ is highly composite (a power of 2) and if F_t is chosen as the DFT length, q , then the corresponding $(F_t - 1)$ -point cyclic convolution can be computed by an NTT (in the present case FNT) with a FFT-type algorithm.

Fermat numbers up to F_4 , namely $\{3, 5, 17, 257, 65537\}$ are all primes, and these are possible sequence lengths (that is $q = F_t$, for $t = 0, 1, 2, 3, 4$) for the computation of Discrete Fourier Transforms using NTTs.

For actual implementations, not all Fermat primes that can be used as sequence lengths are attractive for the computation of DFT. For example, $F_4 (= 65537)$ is long enough (or perhaps too long) for many kinds of applications, but there is no suitable Fermat Number Transform (with sequence length $= 65537 - 1 = 65536$) available for such a long sequence length. In the light of this situation, it would seem that the most attractive transform lengths are $F_1 (= 5)$, $F_2 (= 17)$ and $F_3 (= 257)$. Table 8.1 lists the number of real operations for four possible prime (Fermat primes) lengths and four other lengths (to be discussed below). The number of multiplications (ML) is found by

$$ML = q - 1 \quad (8.1)$$

and the number of shift-adds (with slight modification for length 257, below) is given by eqn. 7.26

$$A = 2(q - 1) \log_2(q - 1) - \frac{q - 3}{2} \quad (8.2)$$

It can be seen from table 8.1 that for a length 257 DFT we have only a single choice of base (F_6) for modulo arithmetic and in this case the root of unity is $\sqrt{2}$. Agarwal and Burrus [28,40] have shown that for Fermat Number Transform with modulo base F_t and transform length 2^{t+2} , the root of unity α can be written as

$$\alpha = \sqrt{2} = 2^{2^{t-2}} (2^{2^{t-1}} - 1)$$

In the present case, $\alpha = \sqrt{2} = 2^{48} - 2^{16}$ which requires a two "ones" representation in the wordlength. Hence the multiplication by $\sqrt{2}$ requires one extra shift-add as compared to the number of shift-adds for the multiplication by a single power of two. However, in the fast FNT procedure (FFT-type algorithm), only one stage requires multiplications by odd powers of α . This fact means that $\frac{1}{2}(q-1)$ extra shift-adds are used in the forward FNT transformation and $\frac{1}{4}(q-1)$ extra shift-adds for reverse FNT transformation. The resulting increase in number of shift-adds is of course very small. Thus F_1 , F_2 and F_3 form three very efficient basic sequence (transform) lengths for the computation of Discrete Fourier Transform using Fermat Number Transform.

Returning to the sequence length problem listed in table 8.1, only a few Fermat primes are suitable for the computation of the DFT using FNTs. However this problem may be resolved by suitable multi-dimensional techniques. Consider, the Discrete Fourier Transform length be a prime number, q as before. Then $(q-1)$ must be a composite number and let $(q-1)=N=N_1 N_2$. The mapping from the one-dimensional cyclic convolution of eqns. 7.5 and 7.6 into a

two-dimensional convolution, can be written in linear form as follows

$$n = K_1 n_1 + K_2 n_2 \quad (8.3)$$

where $n_1=0,1,\dots,N_1-1$ and

$$n_2=0,1,\dots,N_2-1$$

The map[68] is cyclic in n_i , $i = 1$ or 2 , if and only if $K_i = C \frac{N_1 N_2}{N_i}$ where C is an integer. For a unique map, C and N_i have to be relatively prime. In order to apply the NTT, or more precisely, the FNT, to effect the computation, at least one dimension of this two-dimensional mapping must be cyclic. Hence eqn. 8.3 is written as

$$n = \langle K_2 n_2 + CN_2 n_1 \rangle_N \quad (8.4)$$

where K_2 is a constant

For this case, (i) the map is unique and cyclic in n_1 and (ii) the map is unique and not necessarily cyclic in n_2 .

Theorem

The conjugate structure (see eqn. 7.15) of the array $\{W_0^g : n=0,1,\dots,q-2\}$ is preserved in n_1 if N_1 is even for a two-dimensional mapping of the array using a map $n = \langle K_2 n_2 + CN_2 n_1 \rangle_N$ where K_2 and C are integers, and $N_2 N_1 = q-1$.

Before giving the proof of this theorem let us consider, for example, a 13-point DFT. Choose $g=2$, the array $\{W_0^g : n=0,1,\dots,q-2\}$ becomes

$$\{W_0^1, W_0^2, W_0^4, W_0^8, W_0^3, W_0^6, W_0^{1*}, W_0^{2*}, W_0^{4*}, W_0^{8*}, W_0^{3*}, W_0^{6*}\}$$

To transform this cyclic array into a 3×4 two-dimensional array, let us choose the map $n = \langle 4n_2 + 3n_1 \rangle_{12}$. Hence

$$\{w_0^g : n=0,1,\dots,q-2\} \longleftrightarrow \begin{bmatrix} w_0^1 & w_0^8 & w_0^{1*} & w_0^{8*} \\ w_0^3 & w_0^{11} & w_0^{3*} & w_0^{11*} \\ w_0^9 & w_0^7 & w_0^{9*} & w_0^{7*} \end{bmatrix}$$

Each row (dimension n_1) of this resultant two-dimensional array processes the same conjugate structure as the original one-dimensional array.

Proof:

The map $n = \langle K_2 n_2 + CN_2 n_1 \rangle_N$ gives N_2 rows of N_1 -point cyclic arrays. The term $K_2 n_2$, for $n_2 = 0, 1, \dots, N_2 - 1$ gives a base or an offset value for finding sets of data for N_1 -point cyclic arrays. Thus, this term does not affect the property of the N_1 -point cyclic arrays. Since the n_1 map is cyclic, $CN_2 n_1$ can be written as $\langle CN_2 n_1 \rangle_{N_1}$ and actually the set of integers $\{\langle CN_2 n_1 \rangle_{N_1} : n_1 = 0, 1, \dots, N_2 - 1\}$ is merely a permutation of the set $\{0, N_2, 2N_2, \dots, (N_1 - 1)N_2\}_{\text{mod } N}$. However

$$\begin{aligned} & \{\langle CN_2 n_1 \rangle_{N_1} : n_1 = 0, 1, \dots, N_1 - 1\} \\ &= \{0, CN_2, \dots, C(\frac{N_1}{2} - 1)N_2, C\frac{N_1}{2}N_2, C(\frac{N_1}{2} + 1)N_2, \dots, C(N_1 - 1)N_2\}_{\text{mod } N} \\ &= \{0, CN_2, \dots, C(\frac{N_1}{2} - 1)N_2, C\frac{N}{2}, (C\frac{N}{2} + CN_2), \dots, (C\frac{N}{2} + C(\frac{N_1}{2} - 1)N_2)\}_{\text{mod } N} \end{aligned} \quad (8.5)$$

From eqns. 7.15 and 7.16, it is clear that

$$W_{\langle n \rangle_N} = W_{\langle n + \frac{N}{2} \rangle_N}^* \quad (8.6)$$

Hence it can be seen from eqns. 8.5 and 8.6 that after substituting eqn. 8.4 into the expression $\{W_0^n : n=0,1,\dots,q-2\}$ we have arrays

$$\begin{aligned} & \{W_{n_2,0}, W_{n_2,1}, \dots, W_{n_2,(\frac{N_1}{2}-1)}, W_{n_2,0}^*, W_{n_2,1}^*, \dots, W_{n_2,(\frac{N_1}{2}-1)}^*\} \\ \text{or } W_{n_2, \langle n_1 \rangle_{N_1}} &= W_{n_2, \langle n_1 + \frac{N_1}{2} \rangle_{N_1}}^* \end{aligned} \quad (8.7)$$

$$n_1 = 0, 1, \dots, N_1 - 1$$

This completes the proof of the theorem.

Corollary:

It is impossible to map $\{W_0^n : n=0,1,\dots,q-2\}$ into a two-dimensional cyclic array (cyclic in both dimensions) with the conjugate structure preserved in both dimensions.

Proof:

The proof of this corollary can be seen from the fact that for the mapping from one-dimensional cyclic array into a two-dimensional cyclic array (in both dimensions), N_1 and N_2 should be relatively prime [68]. Hence either N_1 or N_2 can be even. From the above theorem, it is evident that for a cyclic mapping of $\{W_0^n : n=0,1,\dots,q-2\}$ in both dimensions, only one dimension preserves conjugate structure. This proves the corollary.

A direct consequence of the conjugate property of the

two-dimensional mapping is that the results of both real and imaginary parts of the Fermat Number Transformed sequence of $\{W_{n_2, n_1} : n_1=0, 1, \dots, N_1-1\}$ would be alternately zero. Hence the total number of real multiplications for each N_1 -point DFT reduces from $2N_1$ to N_1 . Similarly, the conjugate structure of $X(\langle g^k \rangle_q)$ in eqn. 7.25 is also preserved for this mapping. As in the previous example, if $n = \langle 4n_2 + 3n_1 \rangle_{12}$ then k has to map to $\langle 4n_2 + 9n_1 \rangle_{12}$ [68]. Therefore we obtain

$$\{X(2), X(4), X(8), X(3), X(6), X(12), X(2)^*, X(4)^*, X(8)^*, X(3)^*, X(6)^*, X(12)^*\}$$

$$\leftrightarrow \begin{bmatrix} X(2) & X(10) & X(2)^* & X(10)^* \\ X(6) & X(4) & X(6)^* & X(4)^* \\ X(5) & X(12) & X(5)^* & X(12)^* \end{bmatrix}$$

It is clear that only $X(2), X(10), X(6), X(4), X(5)$ and $X(12)$ need to be computed, the other $X(n)$ values are merely complex conjugates of these values.

Since N_1 is even, it is most worthwhile to choose N_1 as a power of two so that a FFT-type algorithm may be used. Depending on whether the second dimension is a cyclic or a linear convolution [67], Pseudo Fermat Transforms, Pseudo Mersenne Transforms, Mersenne Number Transforms, the Lagrange Interpolation Formula or any short convolution algorithm can be used to carry out the computation.

In practical applications, the longest transform length (with 2 or $\sqrt{2}$ as the root of unity) available for Fermat Number Transform is 256, but in this case, the modulo base is $F_6 (= 2^{64} + 1)$. This

word-length is too long for some computers (especially mini- or micro-computers) and for some hardware implementations. We therefore begin our further discussion with a basic sequence length of 64. With this basic sequence length, we can use any of the Fermat Numbers in $\{F_4, F_5, F_6\}$ as the modulo base. This gives a flexible choice of wordlengths and dynamic ranges for various applications. From eqn. 7.10

$$X_k = \sum_{n=0}^{q-2} x_n W_{k-n} \quad (8.8)$$

for $k=0,1,\dots,q-2$

$$\text{where } X_k = X(\langle g^{k+1} \rangle_q) \quad (8.9)$$

Recall $N_1 N_2 = N = q-1$. Let $N_1 = 64$ and the map of $n = \langle K_2 n_2 + N_2 n_1 \rangle_N$, and so eqn. 8.8 becomes

$$X_{k_2, k_1} = \sum_{n_2=0}^{N_2-1} \sum_{n_1=0}^{64-1} x_{n_2, n_1} W_{k_2 - n_2, k_1 - n_1} \quad (8.10)$$

The map is cyclic in n_1 , hence we may take the Number Theoretic Transform of eqn. 8.10 with respect to n_1 [67]. Eqn. 8.10 becomes

$$X'_{k_2, k_1} = \sum_{n_2=0}^{N_2-1} X''_{n_2, k_1} W''_{k_2 - n_2, k_1} \quad (8.11)$$

$$\text{where } X''_{n_2, k_1} = \langle \sum_{n_1=0}^{64-1} x_{n_2, n_1} \alpha^{n_1 k_1} \rangle_{M'} \quad (8.12)$$

$$W''_{k_2 - n_2, k_1} = \langle \sum_{n_1=0}^{64-1} W_{k_2 - n_2, n_1} \alpha^{n_1 k_1} \rangle_{M'} \quad (8.13)$$

$$k_2=0,1,\dots,N_2-1 \text{ and}$$

M' =base for modulo arithmetic

If N_1 and N_2 are not mutually prime eqn. 8.11 is a linear convolution sum of the sequences $\{X''_{n_2,k_1} : n_2=0,1,\dots,N_2-1\}$ and $\{W''_{n_2,k_1} : n_2=0,1,\dots,N_2-1\}$. On the other hand, this equation can also be arranged as a cyclic convolution sum if N_1 and N_2 are mutually prime. For the case of linear convolutions, $(2-\frac{1}{N_2})$ [67] multiplications per point are required for the computation. This number of multiplications is still relatively large⁺ if we use q 's as basic transform lengths for the application of Fermat Number Transform to DFT. We now turn our attention to the case with cyclic mapping in N_2 . A very simple mapping is $n = \langle N_1 n_2 + N_2 n_1 \rangle_{q-1}$ where $N_1=64$. If n_2 is cyclic then a Number Theoretic Transform can be used again to find this convolution sum. However N_1 and N_2 are mutually prime, N_1 is even and if we restrict ourselves to using FNTs only, Number Theoretic Transforms are not suitable for this dimension.

⁺As a typical case if we take $N_2=2$ we require 1.5 multiplications to compute one point. This would still be a practical proposition, but the number of multiplications per point is slightly greater than in the cases to be discussed below.

Recall the fact that of the various short odd cyclic convolution algorithms introduced by Winograd [25] and Agarwal and Cooley [24], the length-3 cyclic convolution algorithm is the most efficient in terms of both the numbers of additions and multiplications per point needed. In fact only 4 multiplications and 11 additions are required for the computation of 3-point cyclic convolution sum. Let $N_2=3$ so that equs. 8.10 and 8.11 become

$$X_{k_2, k_1} = \sum_{n_2=0}^{3-1} \sum_{n_1=0}^{64-1} x_{n_2, n_1} w_{k_2 - n_2, k_1 - n_1} \quad (8.14)$$

$$X'_{k_2, k_1} = \sum_{n_2=0}^{3-1} x''_{n_2, k_1} w''_{k_2 - n_2, k_1} \quad (8.15)$$

where $k_1=0,1,\dots,63$

$k_2=0,1,2$

It is important to recall that the choice of N_1 and N_2 should always agree with our basic design; this means that they must be chosen such that $(q-1) = N_1 N_2$ or equivalently $(N_1 N_2 + 1)$ must be a prime number. In our case, $(N_1 N_2 + 1)$ is equal to 193 which is prime with 5 being the minimum primitive root.

Let us now work out the number of operations required to compute this 193-point DFT. There are 64 3-point cyclic convolutions hence overall there are $64 \times 4 = 256$ multiplications. Besides the 64 3-point cyclic convolutions, there are three 64-point FNT transforms (forward and reverse), hence by using eqn. 8.2, the total number of shift-adds (some are simply additions) becomes $64 \times 11 + 3 \times [2 \times 64 \log_2 64 - (64-2)/2] - 2 = 2913$. This method requires 1.33

multiplications and 16.19 shift-adds per point respectively. Similarly, length $13(=3 \times 4 + 1)$ -point DFT and length $97(=3 \times 32 + 1)$ -point DFT can be computed by using 1.23 (6.69) and 1.32 (13.04) multiplications (additions) per point respectively.

On the other hand, for systems that can have relatively large word size (64-bit words for the present case) we may use a length-256 FNT with $F_6(=2^{64}+1)$ as a modulo base to construct another basic transform length for the DFT using the FNT. In this case, the DFT transform length is $(256 \times 3 + 1) = 769$ which again is a prime number, with 11 being the minimum primitive root. The total number of multiplications is $256 \times 4 = 1024$ and the total number of shift-adds becomes $256 \times 11 + 3 \times 4 \times 161 - 2 = 15297$. The DFT with this sequence length requires 1.33 multiplications and 19.89 shift-adds per point respectively.

Length 3-, 5-, 17-, 257-, 13-, 97-, 193- and 769-point DFTs form eight basic Fermat Number Transform modules and these modules are used to construct algorithms for computing long Discrete Fourier Transforms in the next section.

8.2 Prime Factor Implementation

The basic idea in this section arises mainly from the fact that all eight basic Fermat Number Transform modules obtained so far are prime in length. Thus any two or more of these modules can form basic lengths for two- or multi-dimensional rearrangements of Discrete Fourier Transforms. Furthermore, any one or more of these modules can

also combine with other short (or long if there happen to be any) DFT algorithms to compute long Discrete Fourier Transforms.

Now let us examine the multi-dimensional mapping of the DFT equation. For the sake of simplicity, we consider a simple two-dimensional case. Let N of eqn. 7.1 be a composite number, $N_3 N_4 = N$. Further let N_3 and N_4 be relatively prime so that the "twiddle factors" [97] of the multi-dimensionally rearranged DFT equation can be avoided by the application of the Chinese Remainder Theorem (CRT) [34,68]. In this case eqn. 7.1 can be written as

$$Y_{k_4, k_3} = \sum_{n_4=0}^{N_4-1} \sum_{n_3=0}^{N_3-1} x_{n_4, n_3} W_3^{n_3 k_3} W_4^{n_4 k_4} \quad (8.16)$$

$$\text{where } W_3 = e^{-j \frac{2\pi}{N_3}}$$

$$W_4 = e^{-j \frac{2\pi}{N_4}}$$

and the mapping becomes

$$n = N_3 n_4 + N_4 n_3 \quad (8.17)$$

$$k = N_3 N_3^{-1} k_4 + N_4 N_4^{-1} k_3 \quad (8.18)$$

$$\text{where } \langle N_3 N_3^{-1} \rangle_{N_4} = 1$$

$$\langle N_4 N_4^{-1} \rangle_{N_3} = 1$$

Recall also the fact that both the Prime Factor Algorithm (PFA) [31] and the Winograd Fourier Transform Algorithm [25] essentially make use of the same multi-dimensional mapping and short DFT algorithms for their computation. The number of operations per point becomes rather large for increasing DFT sizes. This is mainly due to the nature of multi-dimensional mapping and also due to limited number of available efficient short DFTs. There are eight efficient short DFT algorithms [25,98] which are frequently used, namely length-2,3,4,5,7,8,9 and 16. Four of these are powers of two and so only one of these four can appear in the same multi-dimensional rearranged DFT. However if we add our basic FNT modules to these 8 short DFT algorithms, we have available a much larger choice of transform lengths and the number of multiplications per point remains a very small number, even for DFT lengths longer than ten thousand.

Eqn. 8.16 can now be written in the following form suitable for the Prime Factor Algorithm [31],

$$Y_{k_4, k_3} = \sum_{n_4=0}^{N_4-1} \left(\sum_{n_3=0}^{N_3-1} x_{n_4, n_3} w_3^{n_3 k_3} \right) w_4^{n_4 k_4} \quad (8.19)$$

The two-dimensional transform in eqn. 8.19 may be implemented by first calculating N_4 length- N_3 DFT's

$$Y'_{n_4, k_3} = \sum_{n_3=0}^{N_3-1} x_{n_4, n_3} w_3^{n_3 k_3} \quad (8.20)$$

then calculating N_3 length- N_4 DFT's

$$Y_{k_4, k_3} = \sum_{n_4=0}^{N_4-1} Y'_{n_4, k_3} W_4^{n_4 k_4} \quad (8.21)$$

The number of multiplications and additions become

$$ML = N_4 M_3 + N_3 M_4 \quad (8.22)$$

$$A = N_4 A_3 + N_3 A_4 \quad (8.23)$$

where $M_3(A_3)$ and $M_4(A_4)$ are
the number of multiplications
(additions) for length N_3 and
 N_4 DFTs respectively

Any one of our eight FNT modules can be used in N_3 -point DFT's and any one of the eight short DFT's can be used in N_4 -point DFT's, provided that N_3 and N_4 are relatively prime. However after a close examination of the eight short DFT algorithms, it becomes clear that the most efficient lengths are 2 and 4, as they both require only trivial multiplications, therefore in this case eqn. 8.22 becomes $ML=N_4 M_3$. This immediately doubles or quadruples the sequence lengths of the four FNT modules without increasing the number of required multiplications per point. For other transform lengths there is an unavoidable but nevertheless slight increase in the number of multiplications per point.

Eqn. 8.16 may also be described by the following form suitable for the computation using the Winograd Fourier Transform Algorithm [25],

$$\bar{Y}_{k_4} = \sum_{n_4=0}^{N_4-1} (\bar{W} \bar{X}_{n_4}) W_4^{n_4 k_4} \quad (8.24)$$

where

$$\bar{X}_{n_4} = \begin{bmatrix} x_{n_4,0} \\ x_{n_4,1} \\ \dots \\ \dots \\ x_{n_4,N_3-1} \end{bmatrix} \quad \bar{Y}_{k_4} = \begin{bmatrix} y_{k_4,0} \\ y_{k_4,1} \\ \dots \\ \dots \\ y_{k_4,N_3-1} \end{bmatrix}$$

$$\bar{W} = \begin{bmatrix} w_3^0 & w_3^0 & \dots & w_3^0 \\ w_3^0 & w_3^1 & \dots & w_3^{N_3-1} \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ w_3^0 & \dots & \dots & w_3^{(N_3-1)^2} \end{bmatrix}$$

Eqn. 8.24 is obtained by essentially considering eqn. 8.16 as a one-dimensional DFT of length N_4 with each scalar replaced by a vector of N_3 terms. The number of operations in this case become

$$ML = M_3 M_4 \quad (8.25)$$

$$A = N_3 A_4 + M_4 A_3 \quad (8.26)$$

However because of the nesting nature of this algorithm, the trivial multiplications (the multiplication of w^0) in the basic FNT modules cannot be ignored in eqn. 8.25. From eqn. 7.2 and 7.4, it is

clear that there are two trivial multiplications, namely $(\sum_{n=0}^{q-1} x(n))W_0^0$ and $x(0)W_0^0$. In order to make a slight simplification of the computation, let us make a rearrangement of eqn. 7.4

$$Y(\langle g^k \rangle_q) = x(0) + \sum_{n=1}^{q-1} x(\langle g^{-n} \rangle_q) W_0^{\langle g^{k-n} \rangle_q}$$

such that

$$Y(\langle g^k \rangle_q) = \sum_{n=1}^{q-1} [x(\langle g^{-n} \rangle_q) - x(0)] W_0^{\langle g^{k-n} \rangle_q} \quad (8.27)$$

If eqn. 8.27 is used instead of eqn. 7.4 one W_0^0 multiplication is saved. The only difference between eqn. 7.4 and eqn. 8.27 is that $x(0)$ is included before the computation of the cyclic convolution in eqn. 8.27 whereas $x(0)$ is included after the computation of the cyclic convolution in eqn. 7.4.

As shown in Table 8.2, if we include the multiplication of $(\sum_{n=0}^{q-1} x(n))W_0^0$ in eqn. 7.2, the number of multiplications required for the eight basic FNT modules increases slightly.

Let us illustrate this idea with $N = 15$. Hence $N = 15 = 3 \times 5$. Eqn. 8.17 become

$$n = \langle 5n_4 + 3n_3 \rangle_{15}$$

$$\text{where } n_3 = 0, 1, 2, 3, 4$$

$$n_4 = 0, 1, 2$$

i.e.

$$\begin{array}{lll}
 x_{0,0} = x(0) & x_{1,0} = x(5) & x_{2,0} = x(10) \\
 x_{0,1} = x(3) & x_{1,1} = x(8) & x_{2,1} = x(13) \\
 x_{0,2} = x(6) & x_{1,2} = x(11) & x_{2,2} = x(1) \\
 x_{0,3} = x(9) & x_{1,3} = x(14) & x_{2,3} = x(4) \\
 x_{0,4} = x(12) & x_{1,4} = x(2) & x_{2,4} = x(7)
 \end{array}$$

Eqn. 8.16 and 8.24 become

$$y_{k_4, k_3} = \sum_{n_4=0}^{3-1} \left(\sum_{n_3=0}^{5-1} x_{n_4, n_3} w_3^{n_3 k_3} \right) w_4^{n_4 k_4} \quad \text{and}$$

$$\bar{y}_{k_4} = \sum_{n_4=0}^2 (\bar{w} \bar{x}_{n_4}) w_4^{n_4 k_4} \quad (8.28)$$

where

$$\bar{y}_{k_4} = \begin{bmatrix} y_{k_4,0} \\ y_{k_4,1} \\ y_{k_4,2} \\ y_{k_4,3} \\ y_{k_4,4} \end{bmatrix}, \quad \bar{x}_{n_4} = \begin{bmatrix} x_{n_4,0} \\ x_{n_4,1} \\ x_{n_4,2} \\ x_{n_4,3} \\ x_{n_4,4} \end{bmatrix}$$

$$\bar{w} = \begin{bmatrix} w_3^0 & w_3^0 & w_3^0 & w_3^0 & w_3^0 \\ w_3^0 & w_3^1 & w_3^2 & w_3^3 & w_3^4 \\ w_3^0 & w_3^2 & w_3^4 & w_3^1 & w_3^5 \\ w_3^0 & w_3^3 & w_3^1 & w_3^4 & w_3^2 \\ w_3^0 & w_3^4 & w_3^3 & w_3^2 & w_3^1 \end{bmatrix}$$

and

$$w_3 = e^{-j\frac{2\pi}{5}}$$

$$\text{for } n_4 = 0,1,2$$

Now we may use the 3-point DFT algorithm (see ref. 25, say for example), to evaluate eqn. 8.28

$$\left. \begin{aligned} \bar{S}_1 &= \bar{X}_1 + \bar{X}_2 \\ \bar{S}_2 &= \bar{X}_1 - \bar{X}_2 \\ \bar{S}_0 &= \bar{X}_0 + \bar{S}_1 \\ \bar{M}_0 &= 1 \cdot \bar{W} \bar{S}_0 \\ \bar{M}_1 &= (\cos \frac{-2\pi}{3} - 1) \cdot \bar{W} \bar{S}_1 \\ \bar{M}_2 &= j \sin \frac{-2\pi}{3} \cdot \bar{W} \bar{S}_2 \\ \bar{S}_3 &= \bar{M}_0 + \bar{M}_1 \\ \bar{S}_4 &= \bar{S}_3 + \bar{M}_2 \\ \bar{S}_5 &= \bar{S}_3 - \bar{M}_2 \end{aligned} \right\} \quad (8.29)$$

and

$$\begin{aligned} \bar{Y}_0 &= \bar{M}_0 \\ \bar{Y}_1 &= \bar{S}_4 \\ \bar{Y}_2 &= \bar{S}_5 \end{aligned}$$

All three expressions in eqn. 8.29 are actually length-5 DFT's multiplied by constants and are in the following form

$$\bar{M}_k = \text{CON } \bar{W} \bar{S}_k$$

where CON is a constant and

$$k=0,1,2$$

Let

$$\bar{M}_k = \begin{bmatrix} m_{k,0} \\ m_{k,1} \\ m_{k,2} \\ m_{k,3} \\ m_{k,4} \end{bmatrix} \quad \text{and} \quad \bar{S}_k = \begin{bmatrix} s_{k,0} \\ s_{k,1} \\ s_{k,2} \\ s_{k,3} \\ s_{k,4} \end{bmatrix}$$

Using the Fermat Number Transform, we obtain

$$\begin{bmatrix} m_{k,0} \\ m_{k,2} \\ m_{k,4} \\ m_{k,3} \\ m_{k,1} \end{bmatrix} = \text{CON} \begin{bmatrix} W_3^0(s_{k,0} + s_{k,1} + s_{k,2} + s_{k,3} + s_{k,4}) \\ b_{k,2} \\ b_{k,4} \\ b_{k,3} \\ b_{k,1} \end{bmatrix} \quad (8.30)$$

where

$$\begin{bmatrix} b_{k,2} \\ b_{k,4} \\ b_{k,3} \\ b_{k,1} \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \alpha^3 & \alpha^2 & \alpha^1 \\ 1 & \alpha^2 & \alpha^0 & \alpha^2 \\ 1 & \alpha^1 & \alpha^2 & \alpha^3 \end{bmatrix} \left\{ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \alpha^1 & \alpha^2 & \alpha^3 \\ 1 & \alpha^2 & \alpha^0 & \alpha^2 \\ 1 & \alpha^3 & \alpha^2 & \alpha^1 \end{bmatrix} \begin{bmatrix} W_3^1 \\ W_3^2 \\ W_3^4 \\ W_3^3 \end{bmatrix} \chi \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \alpha^1 & \alpha^2 & \alpha^3 \\ 1 & \alpha^2 & \alpha^0 & \alpha^2 \\ 1 & \alpha^3 & \alpha^2 & \alpha^1 \end{bmatrix} \begin{bmatrix} (s_{k,3} - s_{k,0}) \\ (s_{k,4} - s_{k,0}) \\ (s_{k,2} - s_{k,0}) \\ (s_{k,1} - s_{k,0}) \end{bmatrix} \right\} \quad (8.31)$$

where χ means term by term multiplication.

Notice that, for an actual implementation all coefficients have to be scaled to integers and they have to be within the allowable dynamic range [see chapter 2]. The "CON" in eqn. 8.30 and "1/4" in eqn. 8.31 should be absorbed inside the W_3 's in which case no extra

multiplication is required, and actually this is the original idea of the Winograd Fourier Transform Algorithm.

For our discussion so far, we have combined one of our basic FNT modules with one short DFT algorithm. However in general one may choose to combine the FNT modules to form a long transform or indeed any number of short DFT or FNT modules can be combined to form long DFT transforms using the Prime Factor Algorithm or the Winograd Fourier Transform Algorithm. The only restriction is that the sequence lengths of these short DFT algorithms and FNT modules must be mutually prime.

8.3 Hybrid Implementation

It is seen that both the Prime Factor Algorithm and the Winograd Fourier Transform Algorithm can combine with eight FNT modules to compute long DFTs. The PFA is especially efficient for sequence lengths of 2 and 4. The WFTA is very efficient for length 3 and 8, therefore it is worthwhile to use a combination of these methods to compute very long DFT sequences.

Table 8.3 lists the number of real multiplications and real additions for various DFT's computed by this hybrid technique with 3-point, 5-point, 17-point, 97-point and 193-point FNT modules. For the cases with 3 or higher dimensions, we consider processing two sets of real data simultaneously, otherwise a slight increase in the number of additions is required [see section 9.2 as well]. In this table either F_4 or F_5 or F_6 can be chosen as the modulo base for the

computation, and this results in a flexible choice of word lengths for the implementation. Figs. 8.1 and 8.2 give the number of multiplications and additions for WFTA [25], Polynomial Transform (PT) [98] and for the present DFT computation technique. As it can be seen from these figures, the number of multiplications for WFTA and the Polynomial Transform is between 30 and 90 percent greater than the corresponding number of multiplications for the present method. On the other hand, the present method also requires between 10 to 20 percent fewer shift-adds than the number of additions in WFTA and Polynomial Transform for most of the sequence lengths. Furthermore, for sequence length 680 or above, the number of shift-adds for the present method is very significantly fewer than the number of additions for WFTA and PT. Hence the present method is more useful for long DFT lengths. It is worth pointing out that for dedicated hardware or microcomputer implementations, the multiplication time is of the order of ten times, or more slower than addition, notwithstanding the relative costs of such operations.

Table 8.4 lists the number of real operations required for various DFT using the 257-point FNT module. In this case, F_6 has to be used as the modulo base for the computation. It is clear from Fig. 8.1 that the number of multiplications per point remains constant and is equal to one for even extremely long DFT sequences. Hence the present method gives a very efficient way to compute long DFTs.

8.4 Conclusion

Eight basic FNT modules for computing DFT's have been presented and used as the basic transform lengths to compute long Discrete Fourier Transforms. The number of multiplications per point for most cases is not more than one and the number of shift-adds is almost equal to (for long sequence lengths, less than) the number of additions in the Winograd Fourier Transform Algorithm and the Polynomial Transform. Thus the present technique is very effective for computing Discrete Fourier Transforms.

Some possible transform lengths based on the eight basic FNT modules have been listed but this listing is no means exhaustive in that the user may combine different short DFTs to form other transform lengths as required. Hence this method gives a large class of transform lengths for computing Discrete Fourier Transforms.

The Fermat Number Transform has already been used to compute convolutions effectively. These are mainly implemented through dedicated hardware, or modules (sometimes with add-on hardware) written in assembly language for a computer/microcomputer. Hence with just some slight modifications of these modules, one may compute Discrete Fourier Transforms using FNTs on the same system. However, since most computer/microcomputer hardware is not designed to do modulo Fermat Number arithmetic, it is perhaps desirable to fabricate digital signal processor chips with FNT arithmetic which can facilitate the computation of DFTs and convolutions using FNTs. The major problem left to be solved is concerned with the design of the architecture of the digital signal processor.

Table 8.1

	Possible	Possible	Number	Number
DFT	Modulo	Root of	of real	of real
Length	Base	Unity	Multiplications	Shift-adds
			ML (per point)	A (per Point)
3	$F_1, \dots, F_6$	$2^2, \dots, 2^{64}$	2 (0.67)	4(1.33)
5	$F_1, \dots, F_6$	$2, \dots, 2^{32}$	4 (0.80)	15 (3.00)
17	F_3, F_4, F_5, F_6	$2, \dots, 2^8$	16 (0.94)	121 (7.12)
257	F_6	$\sqrt{2}$	256 (1.00)	4161 (16.19)
13	$F_1, \dots, F_6$	$2, \dots, 2^{32}$	16 (1.23)	87 (6.69)
97	F_4, F_5, F_6	$2, 2^2, 2^3, 2^4$	128 (1.32)	1265 (13.04)
193	F_5, F_6	$2, 2^2$	256 (1.33)	2913 (15.09)
769	F_6	$\sqrt{2}$	1024 (1.33)	15297 (19.89)

$$\text{where } F_t = 2^{2^t} + 1$$

Number of real multiplications and shift-adds for real data

DFT computed by FNT

Table 8.2

DFT Length N	Number of Multiplications M_3
3	3
5	5
17	17
257	257
13	17
97	129
193	257
769	1025

Number of multiplications (including the multiplications of w_0^0) for eight basic FNT modules.

Table 8.3

DFT Length	Prime Factors	Number of Multiplications per point	Number of Shift-adds per point
12	3x4	0.67	3.33
20	4x5	0.80	5.00
30	2x3x5	1.00	5.33
40	5x8	1.00	6.25
60	3x4x5	1.00	6.33
120	3x5x8	1.00	7.58
204	3x4x17	1.00	10.45
255	3x5x17	1.00	11.45
408	3x8x17	1.00	11.70
510	2x3x5x17	1.00	12.45
680	5x8x17	1.00	13.37
1020	3x4x5x17	1.00	13.45
2040	3x5x8x17	1.00	14.70
4080	3x5x16x17	1.13	16.08
14280	3x5x7x8x17	1.29	19.84
23160	3x5x8x193	1.33	22.67**
39576	3x8x17x97	1.33	24.74
65960	5x8x17x97	1.33	26.41
98940	3x4x5x17x97	1.33	26.49
197880	3x5x8x17x97	1.33	27.74
393720	3x5x8x17x193	1.33	29.79**

** An additional 0.75 shift-adds per point is required for modulo F_4

Number of operations per point for DFTs computed by FNT with modulo base F_4, F_5 or F_6

Table 8.4

DFT Length	Prime Factors	Number of Multiplications per point	Number of Shift-adds per point
257	257	1.00	16.19
514	2x257	1.00	17.19
771	3x257	1.00	17.52
1028	4x257	1.00	18.19
1542	2x3x257	1.00	18.52
3084	3x4x257	1.00	19.52
3855	3x5x257	1.00	20.52
6168	3x8x257	1.00	20.77
7710	2x3x5x257	1.00	21.52
15420	3x4x5x257	1.00	22.52
30840	3x5x8x257	1.00	23.77
52428	3x4x17x257	1.00	26.64
104856	3x8x17x257	1.00	27.89
131070	2x3x5x17x257	1.00	28.64
262140	3x4x5x17x257	1.00	29.64
524280	3x5x8x17x257	1.00	30.89

Number of operations per point for DFTs computed by FNT with
modulo base F_6 only

Figure Captions

Figure 8.1

Number of multiplications per point versus sequence length for DFTs computed by (i) FNT with modulo base F_4 , F_5 or F_6 ($\circ$), (ii) FNT with modulo base F_6 only ($\bullet$), (iii) Polynomial Transform ($\triangle$) and (iv) Winograd Fourier Transform Algorithm (∇).

Figure 8.2

Number of shift-adds (or additions) per point versus sequence length for DFTs computed by (i) FNT with modulo base F_4 , F_5 or F_6 ($\circ$), (ii) FNT with modulo base F_6 only ($\bullet$), (iii) Polynomial Transform ($\triangle$) and (iv) Winograd Fourier Transform Algorithm (∇).

NUMBER OF MULTIPLICATIONS PER POINT

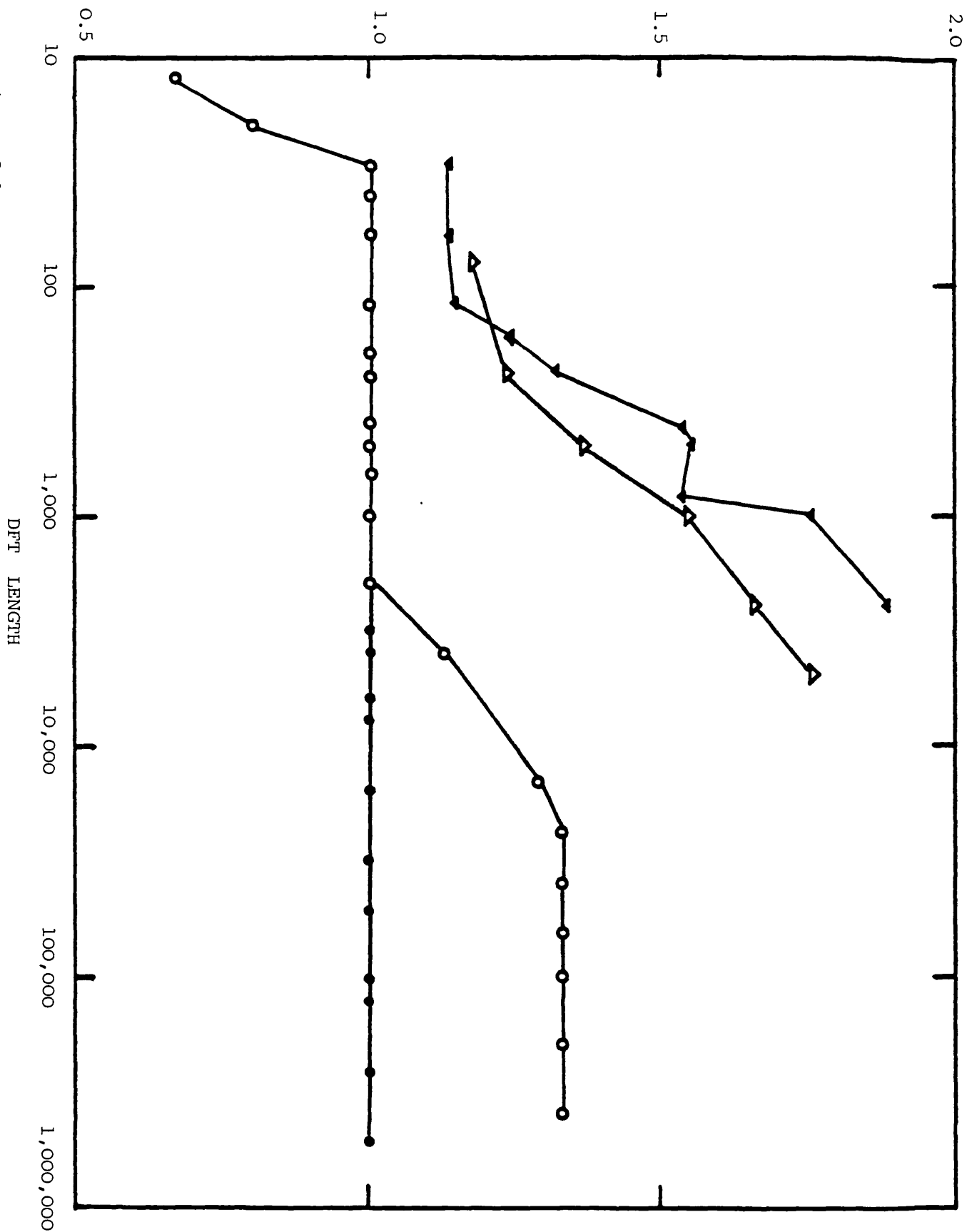


Figure 8.1

NUMBER OF ADDITIONS PER POINT

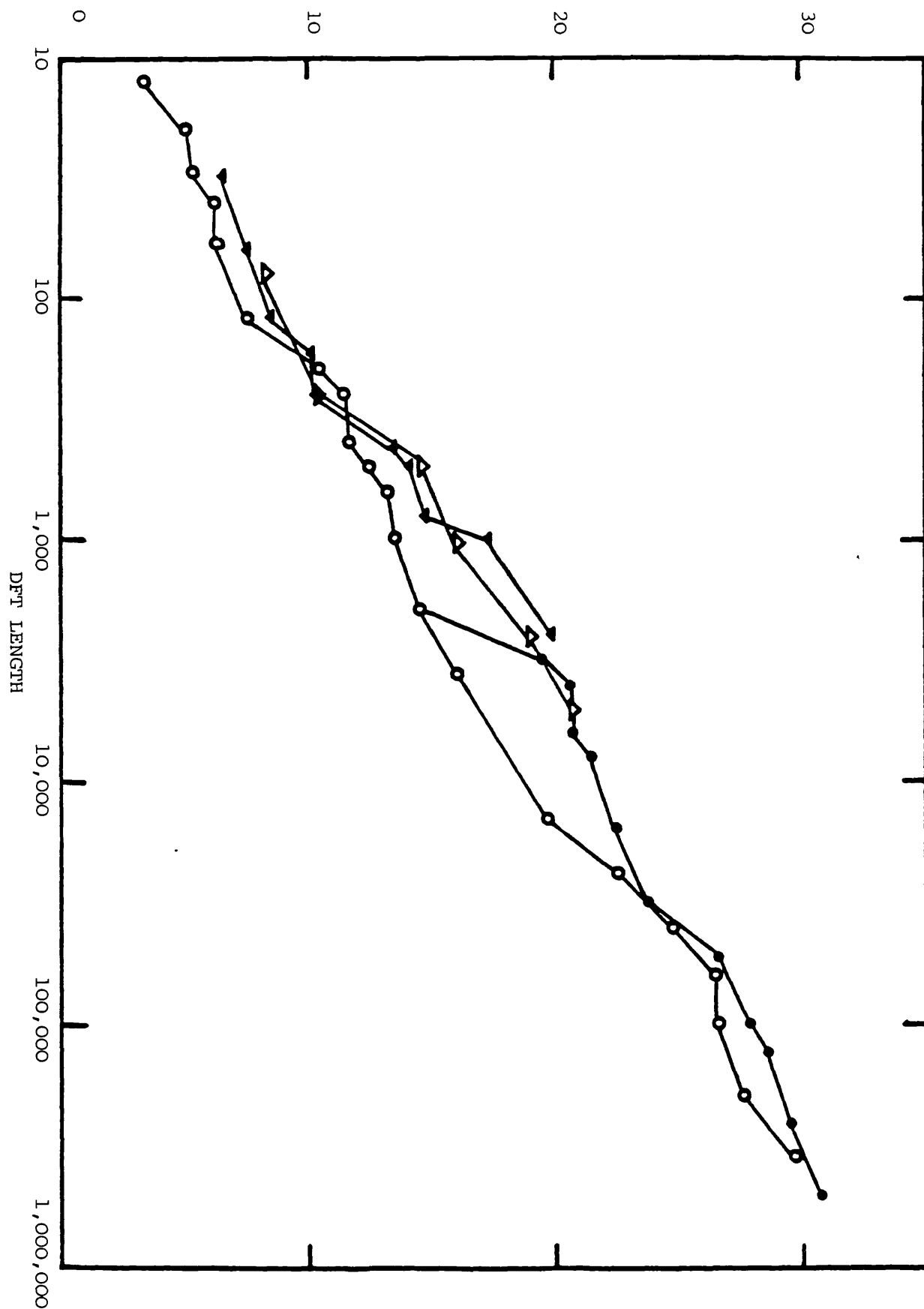


Figure 8.2

CHAPTER NINE FAST MERSENNE NUMBER TRANSFORMS FOR THE
COMPUTATION OF DISCRETE FOURIER TRANSFORM

Recall the fact that a very simple one's complement arithmetic can be used to compute the Mersenne Number Transform. We have also suggested two different realisation structures for the fast computation of convolutions using Mersenne Number Transforms in chapter 4 and chapter 5 respectively. It is the purpose of the present chapter to investigate the use of the same hardware structures for the computation of cyclic convolution using MNT to compute the Discrete Fourier Transform.

Reed, Truong[99-101] and Benjauthrit[101] have proposed hybrid algorithms for computing the Discrete Fourier Transform(DFT) of certain long sequences. In that technique, the Mersenne-prime Number Transform was used in conjunction with a Winograd-type algorithm to compute a DFT. Using this method, with a transform wordlength of 31 or 61 and with real input data, 1.33 or more multiplications per point and a relatively large number of additions are required.

In this chapter we suggest the use of some suitable Mersenne-composite Number Transforms, instead of Mersenne-prime Number Transforms as used by Reed, Truong and Benjauthrit[99-101], to compute Discrete Fourier Transforms. This new approach not only retains the simplicity of the arithmetic of the MNT, but also allows a large choice of transform lengths and wordlengths and requires not more than one multiplication per point for sequences longer than a thousand

points. The time for computing DFT's using this new approach is extremely short, especially if a special hardware structure is used for the implementation. We also show that the number of additions can be reduced significantly if Pseudo-Mersenne Number Transforms are used for the computation of DFT's.

A brief account of the contents of the present chapter has been published in [102] whilst an extended version is given in [103].

9.1 Mersenne Number Transforms and Discrete Fourier Transforms

If Mersenne Number Transforms are used to carry out the computation of eqn. 8.27, a transform length of $2P$ must be used because $(q-1)$ is an even number, where q is an odd prime. So it is clear that the necessary condition for using Mersenne Number Transforms to compute DFT's is given by the following equation:

$$q=2^P+1 \quad (9.1)$$

where both q and P are prime numbers.

It is necessary to check that only some Mersenne Number Transforms satisfy eqn. 9.1. For example, if $P=31$, then $q=2^{31}+1=63$ which is a composite number, hence $M=2^{31}-1$ is not suitable for the present implementation.

$\{2,3,5,11,23,29,41,53\}$ are possible values of P which satisfy eqn. 9.1 and the corresponding sequence lengths for the DFT are $\{5,7,11,23,47,59,83,107\}$. However, the wordlength for the first three possible P 's are too short and have no practical use. Hence

$P=11,23,29,41,53$ are more practical choices for P and are also the possible wordlengths for the corresponding Mersenne Number Transforms. It is interesting to point out that the Mersenne numbers formed by $P=11,23,29,41$ and 53 are composite numbers and that the transforms can actually be called the Mersenne-composite Number Transforms.

Now, let us write eqn. 8.27 as

$$Y_k = \sum_{n=0}^{q-2} x_n W_{k-n} \quad (9.2)$$

for $k=0,1,\dots,q-2$

$$\text{where } Y_k = Y(\langle g^{k+1} \rangle_q) \quad (9.3)$$

$$x_n = x(\langle g^{-(n+1)} \rangle_q) - x(0) \quad (9.4)$$

$$W_n = W_0 \langle g^n \rangle_q \quad (9.5)$$

for $n=0,1,\dots,q-2$

Consider the use of the Mersenne-composite Number Transform to compute eqn. 9.2, we may write

$$X_m = \langle \sum_{n=0}^{q-2} x_n \alpha^{mn} \rangle_M \quad (9.6)$$

and

$$V_m = \langle \sum_{n=0}^{q-2} W_n \alpha^{mn} \rangle_M \quad (9.7)$$

for $m=0,1,\dots,q-2$

where $\alpha = -2$, a root of unity of
order $(q-1)$ and

$$M = 2^{P-1}$$

The results can then be obtained by the inverse transform of the
products, $X_m V_m$'s. That is

$$Y_k = \left\langle \frac{1}{q-1} \sum_{m=0}^{q-2} X_m V_m \alpha^{-mk} \right\rangle_M \quad (9.8)$$

for $k=0,1,2,\dots,q-2$

Substituting eqn. 9.1 into eqns. 9.6 and 9.7 we have

$$X_m = \left\langle \sum_{n=0}^{2^{P-1}-1} x_n \alpha^{mn} \right\rangle_M \quad (9.9)$$

and

$$V_m = \left\langle \sum_{n=0}^{2^{P-1}-1} w_n \alpha^{mn} \right\rangle_M \quad (9.10)$$

Hence eqn. 9.9 can be implemented by eqns. 4.22 and 4.23. So the
total number of operations required to compute all X_m 's is P
additions, P subtractions and two P -point Mersenne Number Transforms.
Similar to the case using the FNT for the computation of DFT, the
method used to calculate the MNT of W_n 's in eqn. 9.10 is unimportant,
since this result can be calculated before the construction of the DFT
processor. However, it is interesting to point out that if a
two-dimensional formulation of eqn. 9.10 using eqns. 4.18 and 4.19 is
used, the transformed results of W_n 's corresponding to eqn. 4.22 are
purely real and the transformed results of W_n 's corresponding to

eqn. 4.23 are purely imaginary. To see the validity of this statement, let us recall the fact that: (i) all the real parts of V_m' 's in eqn. 9.10 are equal to zero for $m=\text{odd}$ and; (ii) all the imaginary parts of the V_m' 's in eqn. 9.10 are equal to zero for $m=\text{even}$. The two-dimensional formulation of eqn. 9.10 corresponding to eqns. 4.22 and 4.23 can be written as

$$V_{0,m_1} = \sum_{n_1=0}^{P-1} W_{0,n_1} \alpha_1^{m_1 n_1} + \sum_{n_1=0}^{P-1} W_{1,n_1} \alpha_1^{m_1 n_1} \quad (9.11)$$

$$V_{1,m_1} = \sum_{n_1=0}^{P-1} W_{0,n_1} \alpha_1^{m_1 n_1} - \sum_{n_1=0}^{P-1} W_{1,n_1} \alpha_1^{m_1 n_1} \quad (9.12)$$

Note that $P \cdot P^{-1}$ in eqn. 4.19 is equal to P and $2 \cdot 2^{-1} (=P+1)$ used in eqn. 4.19 is an even number. V_{0,m_1}' 's in eqn. 9.11 are the results of V_m' 's when $m=\text{even}$ in eqn. 9.10 and so all imaginary parts are zero. Similarly, V_{1,m_1}' 's in eqn. 9.12 are the results of V_m' 's when $m=\text{odd}$ in eqn. 9.10 and so all the real parts are zero. Hence the total number of multiplications required to compute all $X_m V_m'$'s (in eqn. 9.8) is $2P$. That is, the number of multiplications per point remains $(1-\frac{1}{q})$.

Similarly to the MNT, the Inverse Mersenne Number Transform (IMNT) can be computed with this two-dimensional formulation. Eqn. 9.8 can be written as

$$Y_{0,k_1} = \langle \frac{1}{q-1} (\sum_{m_1=0}^{P-1} X_{0,m_1} V_{0,m_1} \alpha_1^{-m_1 k_1} + \sum_{m_1=0}^{P-1} X_{1,m_1} V_{1,m_1} \alpha_1^{-m_1 k_1}) \rangle_M \quad (9.13)$$

$$Y_{1,k_1} = \langle \frac{1}{q-1} (\sum_{m_1=0}^{P-1} X_{0,m_1} V_{0,m_1} \alpha_1^{-m_1 k_1} - \sum_{m_1=0}^{P-1} X_{1,m_1} V_{1,m_1} \alpha_1^{-m_1 k_1}) \rangle_M \quad (9.14)$$

Since all the V_{0,m_1} 's are real and all the V_{1,m_1} 's are imaginary, the $Y_{0,k}$'s and $Y_{1,k}$'s are actually conjugate pairs. That is

$$Y_{0,k} = Y_{1,k}^*$$

Thus a total of two length-P Inverse Mersenne Number Transform are sufficient to calculate a length $(2P+1)$ DFT. Note also that the factor $(\frac{1}{q-1})$ should be absorbed inside the calculation of $V_{0,m}$'s and $V_{1,m}$'s in eqns. 9.13 and 9.14, so no extra operation is required.

Let us consider, for example, the DFT of the sequence $\{x(0), x(1), \dots, x(22)\}$, i.e. $q=23$. In this case 5 is a primitive root which can be used to generate all non-zero elements inside the field modulo 23. Hence eqn. 7.6 becomes

$$\begin{aligned} &\{x(14), x(12), x(7), x(6), x(15), x(3), x(19), x(13), x(21), x(18), x(22), x(9), x(11), \\ &x(16), x(17), x(8), x(20), x(4), x(10), x(2), x(5), x(1)\} \otimes \{w_0^1, w_0^5, w_0^2, w_0^{10}, \\ &w_0^4, w_0^{20}, w_0^8, w_0^{17}, w_0^{16}, w_0^{11}, w_0^9, w_0^{1*}, w_0^{5*}, w_0^{2*}, w_0^{10*}, w_0^{4*}, w_0^{20*}, w_0^{8*}, w_0^{17*}, w_0^{16*}, \\ &w_0^{11*}, w_0^{9*}\} \end{aligned}$$

From eqns. 4.21-4.23, we get

$$\bar{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2^2 & 2^4 & 2^6 & 2^8 & 2^{10} & 2^1 & 2^3 & 2^5 & 2^7 & 2^9 \\ 1 & 2^4 & 2^8 & 2^1 & 2^5 & 2^9 & 2^2 & 2^6 & 2^{10} & 2^3 & 2^7 \\ 1 & 2^6 & 2^1 & 2^7 & 2^2 & 2^8 & 2^3 & 2^9 & 2^4 & 2^{10} & 2^5 \\ 1 & 2^8 & 2^5 & 2^2 & 2^{10} & 2^7 & 2^4 & 2^1 & 2^9 & 2^6 & 2^3 \\ 1 & 2^{10} & 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 \\ 1 & 2^1 & 2^2 & 2^3 & 2^4 & 2^5 & 2^6 & 2^7 & 2^8 & 2^9 & 2^{10} \\ 1 & 2^3 & 2^6 & 2^9 & 2^1 & 2^4 & 2^7 & 2^{10} & 2^2 & 2^5 & 2^8 \\ 1 & 2^5 & 2^{10} & 2^4 & 2^9 & 2^3 & 2^8 & 2^2 & 2^7 & 2^1 & 2^6 \\ 1 & 2^7 & 2^3 & 2^{10} & 2^6 & 2^2 & 2^9 & 2^5 & 2^1 & 2^8 & 2^4 \\ 1 & 2^9 & 2^7 & 2^5 & 2^3 & 2^1 & 2^{10} & 2^8 & 2^6 & 2^4 & 2^2 \end{bmatrix}$$

$$\bar{x}_0 = [x(14), x(7), x(15), x(19), x(21), x(22), x(11), x(17), x(20), x(10), x(5)]^T$$

$$\bar{x}_1 = [x(9), x(16), x(8), x(4), x(2), x(1), x(12), x(6), x(3), x(13), x(18)]^T$$

and

$$\bar{x}_0 = \bar{A}\bar{x}_0 + \bar{A}\bar{x}_1$$

$$\bar{x}_1 = \bar{A}\bar{x}_0 - \bar{A}\bar{x}_1$$

Similarly

$$\bar{w}_0 = [w_0^1, w_0^2, w_0^4, w_0^8, w_0^{16}, w_0^9, w_0^{5*}, w_0^{10*}, w_0^{20*}, w_0^{17*}, w_0^{11*}]^T$$

$$\bar{w}_1 = [w_0^{1*}, w_0^{2*}, w_0^{4*}, w_0^{8*}, w_0^{16*}, w_0^9, w_0^5, w_0^{10}, w_0^{20}, w_0^{17}, w_0^{11}]^T$$

and

$$\bar{w}_0 = \bar{A}\bar{w}_0 + \bar{A}\bar{w}_1$$

$$\bar{W}_1 = \bar{A}\bar{W}_0 - \bar{A}\bar{W}_1$$

In order to use modulo arithmetic, the W_0^n terms have to be normalized to integers. Multiplying these terms by 10 and rounding off the results to integers, we obtain

$$\bar{W}_0 = \begin{bmatrix} 2039 \\ 38 \\ 990 \\ 264 \\ 1569 \\ 189 \\ 1941 \\ 2 \\ 716 \\ 571 \\ 89 \end{bmatrix} \quad \bar{W}_1 = \begin{bmatrix} j1999 \\ j1364 \\ j542 \\ j403 \\ j1500 \\ j1075 \\ j441 \\ j982 \\ j1742 \\ j952 \\ j1216 \end{bmatrix}$$

It can be seen that the vector $\bar{W}_0$ is purely real and the vector $\bar{W}_1$ is purely imaginary.

Now, let us find the computation time required to calculate a q-point [(2P+1)-point] DFT. A length-P MNT module, using a parallel implementation, forms the basic unit of our design (see chapter 5). The forward transformation requires two length-P MNTs, P additions and P subtractions. Hence the time required for this part is approximately equal to $4xPx t_a$, where t_a is the time for one Mersenne Number addition, assuming that the subtraction time is roughly equal

to the addition time. We may check that only the time required for computing two P-point MNT's is required for the inverse transformation, since the addition and subtraction in eqns. 9.13 and 9.14 respectively are not necessary to be carried out for actual computation. The total number of multiplications is $2P$ (or $q-1$). Thus the computation time per point for this method is approximately equal to

$$t = \frac{4xPxt_a + 2xPxt_a + 2Pxt_m}{2P+1}$$

$$\approx 3t_a + t_m$$

where t_m is the time for one
Mersenne Number
multiplication

Using adders similar to those used in [44,45], the computation time, t_a , for a 23-bit MNT addition is approximately 30nS. A high speed multiplier can be constructed from ROMs using the technique of quarter-square multiplication. The computation time for a 23-point MNT multiplication will then be about 120nS. Hence the total computation time for a 47-point DFT is approximately equal to $46[3 \times 30 + 120]nS \approx 10\mu S$ or $0.2\mu S$ per point. The computation time for finding $Y(0)$ in eqn. 7.2 and for forming $[x(\langle g^{-(n+1)} \rangle_q) - x_0]$ in eqn. 9.4 are not counted, since they may be found by a very slight modification of the hardware design. The increase in computation time is negligibly small. The speed of operations can be improved further by using a pipeline architecture, at the cost, however, of increasing

the complexity of the control of data flow and of the timing.

9.2 Long Sequence Lengths

Similarly to the case for the FNT, the sequence lengths (q) of the DFT's considered so far have been prime numbers. It is possible to combine the 8, efficient, short DFT algorithms[25,31], namely of length 2,3,4,5,7,8,9 and 16, to carry out the computation of Discrete Fourier Transforms using a multi-dimensional formulation[67]. The computation can then be simplified by using Winograd's Fourier Transform Algorithm (WFTA) [25] or the Prime Factor Algorithm[31].

Examples:

(i) Consider the computation of a 94-point DFT.

Let the input function be defined by [47]:

$$\begin{aligned} x(n) &= 50n, & 0 \leq n \leq 5 \\ &= 500-50n, & 6 \leq n \leq 10 \\ &= 0, & 11 \leq n \leq 93 \end{aligned}$$

The two-dimensional rearrangement of the 94-point DFT (for Prime Factor Algorithm) equation becomes:

$$\begin{aligned} Y_{k_b, k_a} &= \sum_{n_b=0}^{2-1} \sum_{n_a=0}^{47-1} x_{n_b, n_a} W_a^{n_a k_a} W_b^{n_b k_b} \\ &= \sum_{n_a=0}^{46} x_{0, n_a} W_a^{n_a k_a} + (-1)^{k_b} \sum_{n_a=0}^{46} x_{1, n_a} W_a^{n_a k_a} \end{aligned} \quad (9.15)$$

$$\text{where } W_a = e^{-j\frac{2\pi}{47}}$$
$$W_b = e^{-j\frac{2\pi}{2}} = -1$$
$$\text{for } k_a = 0, 1, \dots, 46$$
$$k_b = 0, 1$$

and the mapping is

$$n = 47n_b + 2n_a$$

$$k = 47k_b + 48k_a$$

Note that, because of the symmetry property only half of Y_{k_b, k_a} need to be computed, the other half can be obtained by merely taking the complex conjugates of the computed half number of points. Thus the computation of the 94-point DFT requires two 47-point DFT's and 47 complex additions. The total computation time required for this 94-point DFT is approximately equal to $10 \times 2 \mu\text{s} + 94 \times 30 \text{ns} \approx 23 \mu\text{s}$. Fig. 9.1 shows the results of the implementation using the present technique.

(ii) For the implementation of a 1128-point DFT, one may also convert the one-dimensional DFT into a two-dimensional form suitable for computation using Winograd's Fourier Transform Algorithm.

$$Y_{k_a, k_b} = \sum_{n_b=0}^{24-1} \sum_{n_a=0}^{47-1} x_{n_b, n_a} W_a^{n_a k_a} W_b^{n_b k_b} \quad (9.16)$$

$$\text{where } W_a = e^{-j\frac{2\pi}{47}}$$

$$W_b = e^{-j\frac{2\pi}{24}}$$

$$\text{for } k_a = 0, 1, \dots, 46$$

$$\text{for } k_b = 0, 1, \dots, 23$$

and the mapping is

$$n = 47n_b + 24n_a$$

$$k = 47 \times 23 k_b + 24 \times 2 k_a$$

Therefore, we can write eqn. 9.16 as:

$$\bar{Y}_{k_b} = \sum_{n_b=0}^{24-1} (\bar{W} \bar{X}_{n_b}) W_b^{n_b k_b} \quad (9.17)$$

where

$$\bar{X}_{n_b} = \begin{bmatrix} x_{n_b, 0} \\ x_{n_b, 1} \\ \dots \\ x_{n_b, 46} \end{bmatrix} \quad \bar{Y}_{k_b} = \begin{bmatrix} y_{k_b, 0} \\ y_{k_b, 1} \\ \dots \\ y_{k_b, 46} \end{bmatrix}$$

$$\bar{W} = \begin{bmatrix} W_a^0 & W_a^0 & \dots & \dots & W_a^0 \\ W_a^0 & W_a^1 & \dots & \dots & W_a^{46} \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ W_a^0 & W_a^{46} & \dots & \dots & W_a^{46^2} \end{bmatrix}$$

Essentially eqn. 9.17 is obtained by considering eqn. 9.16 as a one-dimensional DFT of length-24 with each scalar replaced by a vector of 47 elements. Each multiplication in the 24-point DFT refers to a 47-point DFT. The 24-point vector DFT can be further split into an 8x3 two-dimensional vector DFT. As in the previous case, the MNT is used to calculate the 47-point DFT's and Winograd's[25] short DFT algorithms (length-3 and length-8) are used to calculate the 24-point vector DFT's. For the computation of a 3-point (or 8-point) DFT, 3(or 16) pre-multiplication additions, 3(or 8) multiplications and 3(or 10) post-multiplication additions are required. With real input data all operations before the post-multiplication additions are purely real and the post multiplication additions of the 1st dimension (length-47) may be computed in a similar way to that discussed in section 9.1 However, for the post-multiplication additions of the 2nd dimension (length-3, say for example), the input data are complex, and so complex additions are required. The input to the third dimension (length-8) of post-multiplication additions is also complex, but, because of the symmetrical nature of the real input to

the DFT, only half of the results need to be calculated, the other half can be obtained from the complex conjugates of the results of the first half. So, for this dimension, the number of post-multiplication additions is the same as that required for processing real data. Therefore the computation of a 1128-point real DFT requires approximately half as many operations as are required for a complex DFT since only the post-multiplication additions of the second dimension need complex arithmetic. However, if we want to compute real data with just half the number of operations required for complex data, we can simply use the well-known technique of processing two sets of real data simultaneously. Notice also that for the WFTA the multiplications in the 24-point vector convolutions are actually absorbed inside the multiplications of the 47-point DFT's. Thus only additions are required for this part of the computation, there are no extra multiplications. The time required is $47 \times (8 \times 6 + 3 \times 26) t_a \approx 178 \mu S$. The computation time for the 24 47-point DFT's is approximately $24 \times 10 \mu S = 240 \mu S$. Hence the total computation time for a 1128-point DFT is $418 \mu S$, using adders with a 30nS addition time and a multiplier of 120nS multiplication time.

Table 9.1 lists the number of multiplications per point for some possible DFT lengths computed by this hybrid technique. It can be seen that the number of multiplications per point remains as 1 even for sequences longer than one thousand points.

Moreover, one may also make a full use of our basic MNT module by a slight modification of the multi-dimensional formulation. Eqn. 9.1 can be modified as shown below:

$$q = 2PxN+1 \quad (9.18)$$

In eqn. 9.18 both q and P remain as two prime numbers. P is the word length of the Mersenne Number Transform to be used. As in the previous section, the q -point DFT can be converted into a cyclic convolution of length $2PxN$. Eqn. 9.2 can be written as

$$Y_{k_2, k_1} = \sum_{n_2=0}^{N-1} \sum_{n_1=0}^{2P-1} x_{n_2, n_1} W_{k_2 - n_2, k_1 - n_1} \quad (9.19)$$

$$\begin{aligned} \text{where } n_1, k_1 &= 0, 1, \dots, 2P-1 \\ n_2, k_2 &= 0, 1, \dots, N-1 \end{aligned}$$

and the mapping is

$$n = \langle Kn_2 + Nn_1 \rangle_{2PN} \quad (9.20)$$

where K is an integer

The map is cyclic in n_1 , and so we may take the Mersenne Number Transform of eqn. 9.19 with respect to n_1 . Eqn. 9.19 becomes

$$Y'_{k_2, k_1} = \sum_{n_2=0}^{N-1} X''_{n_2, k_1} W''_{k_2 - n_2, k_1} \quad (9.21)$$

$$\text{where } X''_{n_2, k_1} = \left\langle \sum_{n_1=0}^{2P-1} x_{n_2, n_1} \alpha^{n_1 k_1} \right\rangle_M \quad (9.22)$$

$$w''_{k_2-n_2, k_1} = \left\langle \sum_{n_1=0}^{2P-1} w_{k_2-n_2, n_1} \alpha^{n_1 k_1} \right\rangle_M \quad (9.23)$$

$$k_2 = 0, 1, \dots, N-1$$

$$M = 2^{P-1}$$

It is important to point out that elements inside the sequence $\{Y'_{k_2, k_1} : k_2 = 0, 1, \dots, N-1\}$ for all values of k_1 are either entirely real or entirely imaginary which means that no complex convolution is required. If N is relatively prime to $2P$, eqn. 9.20 can be written as

$$n = \langle 2Pn_2 + Nn_1 \rangle_{2PxN}$$

and eqn. 9.21 becomes a cyclic convolution which can be implemented by Winograd's short convolution algorithms[99-101,25]. However, the control of data flow becomes very simple if we use the Mersenne Number Transform again. If

$$N \leq (P+1) \quad (9.24)$$

The convolution sums of eqn. 9.21 can be found by a length- $2P$ (or length- P) cyclic convolution of two length- $2P$ sequences formed by appending zeros at the ends of the sequences $\{X''_{n_2, k_1} : n_2 = 0, 1, \dots, N-1\}$ and $\{W''_{n_2, k_1} : n_2 = 0, 1, \dots, N-1\}$. This arrangement can always be done no matter whether eqn. 9.21 is a cyclic convolution or a linear convolution, but the number of multiplications per point is

unavoidably increased.

For example, a length-1013 DFT can be rearranged into a $46 \times 22 (=1013-1)$ -point two-dimensional discrete convolution sum. The 22-point linear convolution can be computed by a 46-point MNT module by appending 24 zeros at the ends of the corresponding sequences. Let us estimate the operation time required for the computation of the 1013-point DFT. Totally there are 22 46-point forward Mersenne Number Transforms, 22 46-point Inverse Mersenne Number Transforms and 46 46-point cyclic convolutions. Using the previous 23-point MNT module as a basic unit, the total computation time for this 1013-point DFT is approximately

$$\begin{aligned} & [2 \times 22 \times 46 \times 30 + 46 \times 46 (3 \times 30 + 120) + 2 \times 22 \times 46 \times 30] \text{ns} \\ & \approx 566 \mu\text{s} \end{aligned}$$

It is seen that because of the larger number of multiplications required, the computation time per point is longer than in the case of the 1128-point DFT using the hybrid algorithm, but the control of data flow is extremely simple. Hence this forms a very efficient algorithm for the computation of long DFT's. Table 9.2 shows the number of multiplications per point for some possible sequence lengths.

9.3 DFTs using Pseudo-Mersenne Number Transforms

The Pseudo-Mersenne Number Transform (Pseudo-MNT) was introduced by Nussbaumer[50] to relax the word-length and sequence

length constraints. This transform is defined on the ring of integers modulo a divisor of $M(=2^N-1)$, with N composite) and may have transform lengths other than N or $2N$. Compared with the MNT, the major disadvantage of the Pseudo-MNT is that all operations modulo (2^N-1) must be executed on word-lengths longer than that of the final result. For the application of this transform to the DFT the sequence lengths must satisfy eqn. 9.1. On a close examination of all possible sequence lengths, the Pseudo-MNT with sequence length 35, modulo base $(2^{35}-1)/(31.127)$ and $\alpha = 2$ is most suitable. In this case, the DFT lengths can either be $71(=2 \times 35 + 1)$ or $11(=2 \times 5 + 1)$. Using a two-dimensional formulation of the 35-point Pseudo-MNT (see section 9.5, Appendix) the 71-point DFT can be computed with 1506 additions (21.2 additions per point) and with 71 multiplications (1 multiplication per point). The 11-point DFT can also be computed with 96 additions (8.7 additions per point). Making use of similar techniques to those described in section 9.2 table 9.3 shows the number of operations required for different sequence lengths. It is seen that the number of multiplications per point remains as one, even for very long sequence lengths whereas the number of additions is much smaller than that required for the computation of the DFT using the MNT. However, the Pseudo-Mersenne Number Transforms use longer word-lengths than that used in MNT's for their operations. They also have small dynamic ranges and an extra step is needed to reduce from modulo base (2^N-1) to the actual modulo base for the final step of their operations.

9.4 Conclusion

We have shown that the Discrete Fourier Transform may be computed very efficiently by Mersenne-prime or Mersenne-composite Number Transforms. If Mersenne-composite Number Transforms are used the number of multiplications per point is not more than one, even for sequence lengths longer than one thousand points. The control of data flow is simple and the design of the control unit for the present method is very straightforward.

The techniques presented in this chapter also give a large choice of wordlengths for the computation of DFT's. The sequence lengths can be extended by using a hybrid of Winograd's Fourier Transform Algorithm and the Mersenne Number Transform, or simply by using a nested MNT algorithm. The former method requires the shorter operation time for finding DFT's whereas the latter method requires the simpler control unit. Furthermore, the number of additions can be reduced significantly if pseudo-Mersenne Number Transforms are used for the computation of DFTs.

9.5 Appendix

Number of Additions for DFT's computed by the MNT

The shift-adds required for computing the MNT of the W 's are not counted, since these quantities can be precalculated and stored in a ROM in a hardware implementation or stored in a program in a software implementation. As it is shown in eqns. 9.13 and 9.14, two length- P MNT's and $2P$ additions are required for the forward transformation. From eqns. 9.13 and 9.14, it is also clear that two length- P MNT's are required for the inverse transformation. Notice that the results of the individual summations in eqns. 9.13 and 9.14 are either entirely real or entirely imaginary and so the addition in eqn. 9.13 and the subtraction in eqn. 9.14 do not need to be computed. For the calculation of $Y(0)$, a total number of $(q-1)$ additions is required. However, $Y(0)$ can also be obtained from eqn. 9.9

$$Y(0) = X_0 + x(0)$$

This shows that the computation of $Y(0)$ requires only one extra addition. Lastly, P additions ($2P$ additions are required, if eqn. 9.4 is used) are required to compute the addition in eqn. 7.3. Hence the total number of operations required to compute a q -point ($2P+1$ point) DFT is

$$4 P\text{-point MNT} + (3P + 1) \text{ additions}$$

Now, let us consider the number of additions for cases using length-5 and length-35 Pseudo-Mersenne Number Transforms. The total

number of additions required for a 5-point Pseudo-MNT is $5 \times 4 = 20$. Hence the total number of additions required for a 11-point DFT is $4 \times 20 + 3 \times 5 + 1 = 96$ (or 8.7 additions per point). The implementation of a 35-point Pseudo-MNT can be simplified by arranging the transformation in a 5×7 two-dimensional form. The number of additions is given by eqn. 8.26 and is equal to

$$5 \times 7 \times 6 + 7 \times 5 \times 4 = 350$$

Therefore the number of additions for a 71-point DFT is $4 \times 350 + 3 \times 35 + 1 = 1506$ (or 21.2 additions per point).

Table 9.1

MNT Word Length	DFT Length	Factors	No. of Multi. per point
11	23	23	0.96
11	46	2x23	0.96
11	69	3x23	1.00
11	92	4x23	0.96
11	184	8x23	1.00
11	276	3x4x23	1.00
11	552	3x8x23	1.00
11	1104	3x16x23	1.13
11	1380	3x4x5x23	1.20
11	2760	3x5x8x23	1.20
11	5520	3x5x16x23	1.35
23	47	47	0.98
23	94	2x47	0.98
23	141	3x47	1.00
23	188	4x47	0.98
23	376	8x47	1.00
23	564	3x4x47	1.00
23	1128	3x8x47	1.00
23	2256	3x16x47	1.13
23	2820	3x4x5x47	1.20
23	5640	3x5x8x47	1.20
23	11280	3x5x16x47	1.35
29	59	59	0.98
29	118	2x59	0.98
29	177	3x59	1.00
29	236	4x59	0.98
29	472	8x59	1.00
29	708	3x4x59	1.00
29	1416	3x8x59	1.00
29	2832	3x16x59	1.13
29	3540	3x4x5x59	1.20
29	7080	3x5x8x59	1.20
29	14160	3x5x16x59	1.35

41	83	83	0.99
41	166	2x83	0.99
41	249	3x83	1.00
41	332	4x83	0.99
41	664	8x83	1.00
41	996	3x4x83	1.00
41	1992	3x8x83	1.00
41	3984	3x16x83	1.13
41	4980	3x4x5x83	1.20
41	9960	3x5x8x83	1.20
41	19920	3x5x16x83	1.35
53	107	107	0.99
53	214	2x107	0.99
53	321	3x107	1.00
53	428	4x107	0.99
53	856	8x107	1.00
53	1284	3x4x107	1.00
53	2568	3x8x107	1.00
53	5136	3x16x107	1.13
53	6420	3x4x5x107	1.20
53	12840	3x5x8x107	1.20
53	25680	3x5x16x107	1.35

Number of Multiplications per point for some possible DFT lengths computed by the Hybrid (MNT and WFTA) algorithm.

Table 9.2

Word Length	Sequence Length (q)	Factors (q-1)	No. of Multi. per point
5	31	3x10	1.61
	41	4x10	2.44
7	43	3x14	2.28
	71	5x14	2.76
11	89	4x22	2.72
	199	9x22	2.43
13	157	6x26	2.15
	313	12x26	2.16
17	307	9x34	1.89
	443	13x34	2.61
19	381	10x38	1.90
	647	17x38	2.23
23	461	10x46	2.30
	1013	22x46	2.09
29	513	9x58	3.28
	1567	27x58	2.15
31	683	11x62	2.81
	1861	30x62	2.07
37	1259	17x74	2.18
	2591	35x74	2.11
41	1723	21x82	1.95
	2953	36x82	2.28
43	1721	20x86	2.15
	3613	42x86	2.05
47	2069	22x94	2.14
	4231	45x94	2.09
53	2333	22x106	2.41
	5407	51x106	2.08
59	3541	30x118	1.97
	6491	55x118	2.15
61	3539	29x122	2.10
	7321	60x122	2.03

Possible sequence lengths and the number of multiplications per point for computing DFT's using Nested Mersenne Number Transforms.

Table 9.3

DFT Length	Prime Factors	No. of Multi. per point	No. of Shift-adds per point
22	2x11	1.00	10.18
33	3x11	1.00	11.18
44	4x11	1.00	11.18
66	2x3x11	1.00	12.18
88	8x11	1.00	12.43
132	3x4x11	1.00	13.18
264	3x8x11	1.00	14.43
426	2x3x71	1.00	24.70
568	8x71	1.00	24.95
852	3x4x71	1.00	25.70
1704	3x8x71	1.00	26.95
2343	3x11x71	1.00	32.88
3124	4x11x71	1.00	32.88
4686	2x3x11x71	1.00	33.88
6248	8x11x71	1.00	34.13
9374	3x4x11x71	1.00	34.88
18744	3x8x11x71	1.00	36.13

Number of operations per point for DFT's computed by
Pseudo-MNT with modulo base $M=(2^{35}-1)/(31.127)$

94-POINT DFT COMPUTED BY MNT

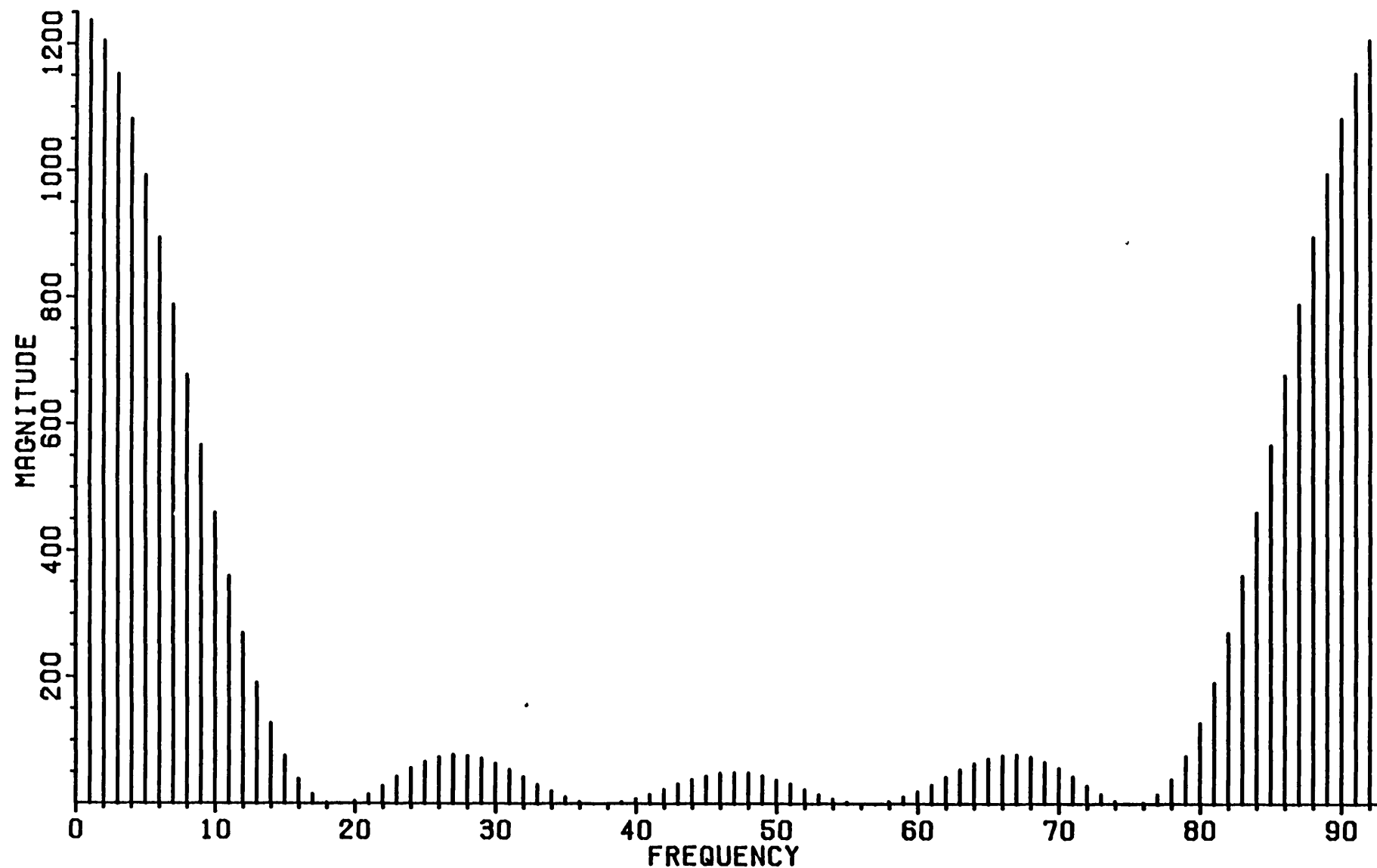


Figure 9.1 94-Point DFT Computed by MNT

CHAPTER TEN GENERAL CONCLUSIONS AND SUGGESTIONS FOR FUTURE RESEARCH

In this thesis we have presented algorithms to apply Number Theoretic Transforms and to design hardware structures for the computation of cyclic convolutions and Discrete Fourier Transforms. It is well-known that a relatively larger word-length than the word-length required for ordinary arithmetic is needed for Number Theoretic Transforms. However, in return this allows us an exact computation of the cyclic convolution.

Chapter 1 gives an introduction and a historical background of the present research area. This chapter reflects the importance and shows the rapid development of this research area.

An over-view of the Number Theoretic Transform was given in chapter 2. This includes a systematic development of the Number Theoretic Transform leading to the Fermat Number Transform, the Mersenne Number Transform, the pseudo-Number Theoretic Transform and the complex Number Theoretic Transform. We have also discussed on the problems of the dynamic range, word-length and sequence length constraints. It is interesting to point out that whilst most of the contents of this chapter can be found in the literature, there are still some points or comments mentioned which have never been published or mentioned before. For example, the derivation of the general equations[2.62-2.65] for the computation of complex convolutions by real NTT it seems not to have been published before.

This is also the first time that the use of Mersenne-composite Number Transforms [pages 66-67] for the computation of complex convolutions has been discussed. This chapter is meant to be a reasonable overview of the state-of-the-art in Number Theoretic Transforms.

Polynomial formulations are used in chapter 3 to provide a systematic derivation of the equations for the computation of long cyclic convolutions using the Number Theoretic Transform. These new formulations show very clearly the structures of the equations for the computation of cyclic convolutions rearranged in multi-dimensional form. The number of multiplications per point is shown to be always less than 2 when N_1 and N_2 are not mutually prime. This figure is reduced to one multiplication per point when N_1 and N_2 are mutually prime and a nested NTT is used. For the convolution of two-dimensional arrays, the number of multiplications per point turns out to be always less than 4 when N_1 and N_2 are not mutually prime whilst the number of multiplications remains as one per point when N_1 and N_2 are mutually prime. The methods presented in this chapter are systematic and general, as opposed to the derivations given by Kraats and Venetsanopoulos[72].

Because of the lack of FFT-types of algorithm for its implementation the Mersenne Number Transform has received much less attention than the Fermat Number Transform. Among all NTTs, the Mersenne Number Transform uses the simplest arithmetic for its implementation. In chapter 4 Mersenne Number Transforms are converted into cyclic convolutions in which form they are amenable to a simple

hardware interpretation. This structure is especially useful for systems requiring the minimum number of components. We have also suggested a possible way to extend this implementation to other Number Theoretic Transforms, in particular to the Fermat Number Transform.

Another major disadvantage of the MNT results from the strong relationship between the length of the sequences that can be convolved and the ring in which the transform is defined. In chapter 5 we have proposed a second hardware implementation technique which is fundamentally different from the method discussed in chapter 4. We have converted the Mersenne-prime and Mersenne-composite Number Transforms into recursive filter form and suggested simple hardware structures to carry out the fast implementation of circular convolutions. The structures proposed in chapter 4 use no ROM or RAM, all the work is done in registers and so the data access time is greatly reduced; whereas the structures in chapter 5 may be used to compute very long cyclic convolutions with simply one multiplication per point. Very fast hardware structures for the parallel implementation of MNTs have also been proposed and this may form a basic unit for the computation of long cyclic convolvers and the Discrete Fourier Transform.

Because of advances in fabrication technology, it is now possible to design architectures which are memory intensive. This means that our study can benefit from the semiconductor revolution. In chapter 6 we have generalized the recursive formulation of the Mersenne Number Transform to any Number Theoretic Transform with prime

sequence length, P , and with modulo base, q , where q is also a prime number. In this case an architecture based on a ROM look-up table is suitable for practical implementations. We then propose a new type of Number Theoretic Transforms which are not restricted by sequence length and word-length constraints. The efficiency of this structure depends mainly on the speed of the RAM and ROM for its implementation, and it also depends on the degree of parallelism used.

In chapter 7, we have derived the general equations for the computation of the Discrete Fourier Transform using the Number Theoretic Transform. The results mainly come from the observation of the properties of the symmetry of the NTT and the DFT. This leads to the minimum number of multiplications that has ever been reported for the computation of the Discrete Fourier Transform. This idea forms the basis of our investigation in chapters 8 and 9.

Eight basic FNT modules have been suggested in chapter 8. These form the basic units for the computation of long DFTs using Fermat Number Transforms. We have also suggested the use of hybrid algorithms using the FNT, the WFTA and the Prime Factor DFT for the computation of DFTs of long sequences. In most cases, this technique requires fewer number of operations than the number of operations required for the Polynomial Transform and Winograd's Fourier Transform Algorithm. It is interesting to point out that in parallel to this development, Bhattacharya and Agarwal[104] have also suggested a hybrid algorithm using the NTT and other short convolution algorithms to compute long cyclic convolutions.

In chapter 9, we have discussed in detail the computation of Discrete Fourier Transforms using Mersenne Number Transforms. It is shown that in the case of Mersenne-composite Number Transforms the number of multiplications per point for real input data is never more than one, even for sequence lengths exceeding one thousand points. This new approach allows a large choice of word-lengths, and in addition the control of data flow is extremely simple. We have also shown that the number of additions can be reduced significantly if the pseudo-Mersenne Number Transform is used for the computation of the DFT.

Both chapters 8 and 9 rely on the assumption that a relatively high efficiency FNT or MNT cyclic convolver exists for the implementation. This is possible if the hardware structures suggested in chapters 4 and 5 or by others[40,43-45] are used for the computation of cyclic convolutions.

It is also interesting to point out that the cyclic convolvers suggested in this thesis may be used as a primary module for constructing the primitives that were suggested by Wu[105], Constantinides, Curtis and Wu[106] for the ALPS (Alternative Low-level Primitive Structures).

A close examination of the algorithms developed during the course of this research and the implementation techniques suggested in consequence reveals that most of our suggested algorithms aim at reducing the number of operations, particularly the number of multiplications, and simplifying the control of data flow in practical

implementations. The state-of-the-art of the fabrication technology allows us to have very fast multipliers available. However, discrete multipliers are still very expensive compared with the price of adders. Furthermore, for VLSI implementations, a large space is always required for a high speed multiplier hence it is very expensive and difficult to fabricate digital signal processor chips with an array of multipliers. On the other hand, the number of gates and space required for the adder is relatively small, hence it is always possible to implement an array of adders [for structures in chapters 4 and 5, say for examples] in a digital signal processor.

The Number Theoretic Transform makes use of the Residue Number System(RNS)[107]. One of the fundamental qualities of the RNS is its high speed computation which arises from its parallel processing of arithmetic operations. An ideal technique for residue arithmetic is a table-lookup operation, and so new memory technologies can be expected to improve the speed and price of NTT-based digital signal processing systems. For example, Phillips announced a GaAs 256-bit RAM that works at 1.5nS. With such units real-time NTT-based system could achieve very high throughput rates indeed.

We have seen that the Number Theoretic Transform offers a fast computational tool for the calculation of cyclic convolutions and Discrete Fourier Transforms. Much research remains to be done on different aspects of the NTT and the related areas.

We have emphasised the importance of Mersenne-composite Number Transforms in this thesis although in previous years

Mersenne-composite Transforms have received almost no attention at all. However, there are still a large number of properties of this transform left to be discovered. For example, for the modulus $M=2^{29}-1$ Mersenne-composite Number Transform, the modulus, M , can actually be factorized into three reasonably short prime numbers ($233 \times 1103 \times 2089$) and so can be implemented very effectively in parallel. Furthermore, since $\langle 233 \rangle_4 = 1$, $\langle 1103 \rangle_4 = 3$ and $\langle 2089 \rangle_4 = 1$, only one constituent modulo base requires complex arithmetic for the computation of cyclic convolutions using a complex composite-Mersenne Number Transform with this modulo base. Hence this might form an efficient and practical way of computing cyclic convolutions.

Another interesting feature of the Mersenne Number Transform is that it may be implemented by ones-complement arithmetic. Making use of this feature as the base of the transform and using distributed arithmetic[80-85] for the calculation, this might form a new and even simpler method for the efficient computation of the Mersenne Number Transform.

The residue arithmetic used in the Number Theoretic Transform is not usually available in general computers and there is also no VLSI digital signal processors dedicated to the computation of residue number arithmetic. This shows that there is an urgent need to investigate the possibilities of making a digital signal processor for this application. This includes the design of hardware structures and instruction sets for the purpose of: (i) parallel processing of data; (ii) making in-place and in-order maps and (iii) inputting and

outputting data.

It would also be interesting to investigate the results of applying the algorithms presented in this thesis to practical applications. These include one-dimensional and multi-dimensional signal processing. It is expected that the saving in computational effort could be more significant in multi-dimensional processing than in the one-dimensional case.

A less certain, but more interesting problem is to investigate the possibilities of proposing a new transform which requires only one multiplication per point without using residue number arithmetic for the computation of cyclic convolutions.

In conclusion, the research presented in this thesis indicates that the proposed algorithms from the Number Theoretic Transforms for the computation of cyclic convolutions are versatile and efficient techniques for digital signal processing. These provide a fresh insight into the problem of the design of high efficiency digital signal processors.

REFERENCES

- 1 CAPPELLINI,V., and CONSTANTINIDES,A.G., and EMILIANI,P.: 'Digital Filters and their Applications', (Academic Press, 1978).
- 2 BOGNER,R.E., and CONSTANTINIDES,A.G.: 'Introduction to Digital Filtering', (John Wiley & Sons, 1975).
- 3 CAPPELLINI,V., and CONSTANTINIDES,A.G.: 'Digital Signal Processing', (Academic Press, 1980).
- 4 OPPENHEIM,A.V., and SCHAFER,R.W.: 'Digital Signal Processing', (Prentice-Hall Inc., 1975).
- 5 CHEN.C.T.: 'One-dimensional Digital Signal Processing', (Dekker, N.Y., 1979).
- 6 STANLEY,W.D.: 'Digital Signal Processing', (Prentice-Hall Co., 1975).
- 7 GOLD,B., and RADER,C.M.: 'Digital Processing of Signals', (McGraw Hill Book Co., 1969).
- 8 CHILDERS,D., and DURLING,A.: 'Digital Filtering and Signal Processing', (West Publ., St. Paul, Minnesota, 1975).
- 9 RABINER,L.R., and GOLD,B: 'Theory and Application of Digital Signal Processing', (Prentice-Hall, Inc., 1975).
- 10 OPPENHEIM,A.V.: 'Applications of Digital Signal Processing', (Prentice-Hall, Inc., 1978).

- 11 GONZALEZ,R.C., and WINTZ,P.: 'Digital Image Processing', (Addison-Wesley Publ. Co., 1977).
- 12 PRATT,W.K.: 'Digital Image Processing', (John Wiley & Sons, 1978).
- 13 RABINER,L.R., and SCHAFER,R.W.: 'Digital Processing of Speech Signal', (Prentice Hall, Inc., 1978).
- 14 PELED,A., and LIU,B.: 'Digital Signal Processing, Theory, Design, and Implementation', (John Wiley & Sons, 1975).
- 15 BOWEN,B.A., and BROWN,W.R.: 'VLSI Systems Design for Digital Signal Processing', Vol.1, (Prentice Hall, Inc., 1982).
- 16 COOLEY,J.W., and TUKEY,J.W.: 'An Algorithm for the Machine Calculation of Complex Fourier Series', Math. Comput., 1965, Vol.19, pp.297-301.
- 17 BRIGHAM,E.O.: 'The Fast Fourier Transform', (Prentice Hall, Inc., 1974).
- 18 RADER,C.M.: 'Discrete Fourier Transforms when the Number of Data Samples is Prime', IEEE Proc., 1968, Vol.56, pp.1107-1108.
- 19 McCLELLAN,J.H., and RADER,C.M.: 'Number Theory in Digital Signal Processing', (Prentice Hall, Inc., 1979).
- 20 NUSSBAUMER,H.J.: 'Fast Fourier Transform and Convolution Algorithms', (Springer-Verlag, 1981).
- 21 ELLIOTT,D.F., and RAO,K.R.: 'Fast Transforms Algorithms, Analyses, Applications', (Academic Press, 1982).

- 22 SIU,W.C., and CONSTANTINIDES,A.G.: 'Number Theoretic Transform and Convolutions', Paper presented at the IEE Seminar on Digital Filters, London, 7th-8th April, 1983, pp.1-29.
- 23 WINOGRAD,S.: 'Some Bilinear Forms whose Multiplicative Complexity depends on the Field of Constants', IBM J., Watson Research Centre, N.Y., U.S.A., IBM Rev. Rep., RC5669, 1975.
- 24 AGARWAL,R.C., and COOLEY,J.W.: 'New Algorithms for Digital Convolution', IEEE Trans., 1977, Vol.ASSP-25, pp.392-410.
- 25 WINOGRAD,S.: 'On Computing the Discrete Fourier Transform', Math. Comput., 1978, Vol.32, pp.175-199.
- 26 AGARWAL,R.C., and BURRUS,C.S.: 'Fast One-dimensional Digital Convolution by Multi-dimensional Techniques', IEEE Trans., 1974, Vol.ASSP-22, pp.1-10.
- 27 BURTON,D.M.: 'Elementary Number Theory', (Allyn and Bacon, Inc., 1980).
- 28 DICKSON,L.E.: 'History of the Theory of Numbers', (Carnegie Institute, Washington, 1919).
- 29 WINOGRAD,S: 'On Computing the Discrete Fourier Transform', Proc. Nat. Acad. Sci., U.S.A., Vol.73, pp.1005-1006.
- 30 SILVERMAN,H.F.: 'An Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA)', IEEE Trans., 1977, Vol.ASSP-25, pp.152-165.
- 31 KOLBA,D.P., and PARKS,T.W.: 'A Prime Factor FFT Algorithm using High-speed Convolution', IEEE Trans., 1977, Vol.ASSP-25, pp.281-294.

- 32 STEEN,A.J.: 'Timing of Some Fast DFT-Algorithms', Signal Processing, Vol.5, 1983, pp.541-546.
- 33 GOOD,I.J.: 'The Interaction Algorithm and practical Fourier Analysis', J. Royal Statist. Soc. B, Vol.20, 1958, pp.361-372,; "Addendum", Vol.22, 1960, pp.372-375.
- 34 GOOD,I.J.: 'The Relationship between two Fast Fourier Transforms', IEEE Trans., 1971, Vol.C-20, pp.310-317.
- 35 NUSSBAUMER,H.J.: 'Digital Filtering using Polynomial Transforms', Electronic Letters, 1977, Vol.13(13), pp.386-387.
- 36 NUSSBAUMER,H.J.: 'New Polynomial Transform Algorithms for Multi-dimensional DFTs and Convolutions', IEEE Trans., 1981, Vol.ASSP-29, pp.74-83.
- 37 NUSSBAUMER,H.J., and QUANDALLE,P.: 'Computation of Convolution and Discrete Fourier Transforms by Polynomial Transforms', IBM J. Rev. & Dev., 1978, Vol-22, pp.134-144.
- 38 POLLARD,J.M.: 'The Fast Fourier Transform in a Finite Field', Math. Comput., 1971, Vol.25, pp.365-374.
- 39 RADER,C.M.: 'Discrete Convolutions via Mersenne Transforms', IEEE Trans., 1972, Vol.C-21, pp.1269-1273.
- 40 AGARWALL,R.C., and BURRUS,C.S.: 'Fast Convolution using Fermat Number Transform with Applications to Digital Filtering', IEEE Trans., 1974, Vol.ASSP-22, pp.87-97.
- 41 RAYNER,P.J.W.: 'Number Theoretic Transforms', Proc. NATO Advanced Study Institute, Sept. 1976, Italy, pp.333-351.

- 42 SIU,W.C.: 'Number Theoretic Transform and its Applications to Digital Signal Processing', Proc. of IERE H.K. Section Workshop on Adv. Micro. and D.S.P., Hong Kong, Sept. 1982, pp.76-101.
- 43 AGARWAL,R.C., and BURRUS,C.S.: 'Number Theoretic Transforms to Implement Fast Digital Convolution', IEEE Proc., Vol.63, 1975, pp.550-560.
- 44 McCLELLAN,J.H.: 'Hardware Realisation of a Fermat Number Transform', IEEE Trans., 1976, Vol.ASSP-24, pp.216-225.
- 45 LEIBOWITZ,L.M.: 'A Simplified Binary Arithmetic for Fermat Number Transform', IEEE Trans., 1976, Vol.ASSP-24, pp.356-359.
- 46 REED,I.S., and TRUONG,T.K.: 'The Use of Finite Fields to Compute Convolutions', IEEE Trans., 1975, Vol.IT-21, pp.208-213.
- 47 REED,I.S., and TRUONG,T.K.: 'Complex Integer Convolution Over a Direct Sum of Galois Fields', IEEE Trans., 1975, IT-21, pp.657-661.
- 48 REED,I.S., TRUONG,T.K., BENJAUTHRIT,B., and WU,C.: 'A Fast Algorithm for Computing a Complex Number Theoretic Transform for Long Sequences', IEEE Trans., 1981, Vol.ASSP-29, pp.122-124.
- 49 NEVIN,R.L.: 'Application of Rader-Brenner FFT Algorithm to Number Theoretic Transforms', IEEE Trans., 1977, Vol.ASSP-25, pp.196-198.
- 50 NUSSBAUMER,H.J.: 'Digital Filtering using Complex Mersenne Transforms', IBM J. Res. Dev., 1976, pp.498-504.

- 51 NUSSBAUMER,H.J.: 'Digital Filtering using Pseudo-Fermat Number Transform', IEEE Trans., 1977, Vol.ASSP-25, pp.79-83.
- 52 DUBOIS,E., and VENETSANOPOULOS,A.N.: 'Number Theoretic Transforms with Modulus $2^{2q}-2^q+1$ ', IEEE Proc., ICASSP, 1978, pp.624-627.
- 53 DUHAMEL,P., and HOLLMANN,H.: 'Number Theoretic Transforms with 2 as a Root of Unity', Electronics Letters, Vol.18, No.22, 1982, pp.978-980.
- 54 HOLLMANN,H., and DUHAMEL,P.: 'Longer NTT's with 2 as a Root of Unity', IEEE, Proc. ICASSP, 1983, pp.159-162.
- 55 DUHAMEL,P., and HOLLMANN,H.: 'A Decomposition of the Arithmetic for NTT's with 2 as a Root of Unity', IEEE Proc., ICASSP, 1984, pp.43.8.1-4.
- 56 THOMAS,J.J., LARSOM,G.N., and KELLER,J.M.: 'Number Theoretic Transforms with Independent Length and Moduli', IEEE Trans., 1983, Vol.ASSP-31, pp.215-217.
- 57 BALLA,P.C., and ANTONIOU,A.: 'Number Theoretic Transform Based on Ternary Arithmetic', IEEE Proc., ICASSP, 1983, pp.155-158.
- 58 REED,I.S., and TRUONG,T.K.: 'Fast Mersenne-prime Transforms for Digital Filtering', Proc. IEE, Vol.125, 1978, pp.433-440.
- 59 POLLARD,J.M.: 'Implemenation of Number Theoretic Transforms', Electronics Letters, Vol.12, No.15, 1976, pp.378-379.
- 60 JULLIEN,G.A., and MILLER,W.C.: 'A Hardware Realisation of an NTT Convolver using ROM arrays', IEEE Proc., ICASSP, 1980, pp.788-791.

- 61 JULLIEN,G.A., and MILLER,W.C.: 'Very High Speed Convolution using Number Theoretic Techniques', IERE, the Third Int. Conf. on Digital Signal Processing of Signals in Communications, Loughborough, England, 1981, pp.49-57.
- 62 NAGPAL,H.K., JULLIEN,G.A., and MILLER,W.C.: 'Processor Architectures for Two-dimensional Convolvers using a Simple Multiplexed Computational Element with Finite Field Arithmetic', IEEE Trans., Vol.C-32, 1983, pp.989-1001.
- 63 RICE,B: 'Some Good Fields and Rings for Computing Number Theoretic Transforms', IEEE Trans., 1979, ASSP-27, pp.432-433.
- 64 SINGLETON,R.C.: 'An Algorithm for Computing the Mixed Radix Fast Fourier Transform', IEEE Trans., 1969, Vol.AU-17, pp.93-103.
- 65 NUSSBAUMER,H.J.: 'Complex Convolutions via Fermat Number Transforms', IBM J. Rev. Develop., 1976, Vol.20, pp.282-284.
- 66 VEGH,E., and LEIBOWITZ,L.M.: 'Fast Complex Convolution in Finite Rings', IEEE Trans., 1976, Vol.ASSP-24, pp.343-344.
- 67 SIU,W.C., and CONSTANTINIDES,A.G.: 'Cyclic Convolution of Long Sequences using Number Theoretic Transform', IEE Proc., Vol.131, Pt.G, Electronic Circuits and Systems, June 1984, pp.119-126.
- 68 BURRUS,C.S.: 'Index Mapping for Multi-dimensional Formulation of the DFT and Convolution', IEEE Trans., 1977, Vol.ASSP-25, pp.239-242.
- 69 KNUTH,D.E.: 'Seminumerical Algorithms', (The Art of Computer Programming, 1981, Vol.2, Reading, MA:Addison-Wesley).

- 70 DUDGEON,D.E., and MERSEREAU,R.M.: 'Multi-dimensional Digital Signal Processing', (Prentice-Hall Inc., 1984).
- 71 RADER,C.M.: 'On the Application of Number Theoretic Methods of High Speed Convolution to Two-dimensional Filtering', IEEE Trans., 1975, Vol.CAS-22, p.575.
- 72 KRAATS,R.H.V., and VENETSANOPOULOS,A.N.: 'Hardware for Two-dimensional Digital Filtering Using Fermat Number Transforms', IEEE Trans., 1982, Vol.ASSP-30, pp.155-162.
- 73 BEAUCHAMP,K.G.: 'Walsh Function and Its Applications', (Academic Press, 1975).
- 74 NUSSBAUMER,H.J.: 'Relative Evaluation of Various Number Theoretic Transforms for Digital Filtering Applications', IEEE Trans., Vol.ASSP-26, 1978, pp.88-93.
- 75 SIU,W.C., and CONSTANTINIDES,A.G.: 'Approach to the Hardware Implementation of Digital Signal Processors using Mersenne Number Transforms', IEE Proc., Vol.131, Pt.E, Computers and Digital Techniques, Jan. 1984, pp.10-18.
- 76 TAYLOR,F.J.: 'Large Moduli Multipliers for Signal Processing', IEEE Trans., 1981, Vol.CAS-28, pp.731-736.
- 77 JULLIEN,G.A.: 'Implementation of Multiplication, Modulo a Prime Number, with Applications to Number Theoretic Transforms', IEEE Trans., 1980, Vol.C-29, pp.899-905.
- 78 BLUESTEIN,L.I.: 'A Linear Filtering Approach to the Computation of Discrete Fourier Transform', IEEE Trans., Vol.AU-18, 1970, pp.451-455.

- 79 RABINER,L.R., SCHAFFER,R.W., and RADER,C.M.: 'The Chirp z-transform Algorithm', IEEE Trans., Vol.AU-17, 1969, pp.86-92.
- 80 PELED,A., and LIU,B.: 'A New Hardware Realisation of Digital Filter', IEEE Trans., Vol.ASSP-22, 1974, pp.456-462.
- 81 BURRUS,C.S.: 'Digital Filter Structures Described by Distributed Arithmetic', IEEE Trans., Vol.CAS-24, 1977, pp.674-680.
- 82 SIU,W.C.: 'Microprocessor-based Implementation of Digital Signal Processors using Distributed Arithmetic', Proc. IERE H.K. Section Workshop on Adv. Micro. & D.S.P., Hong Kong, Sept., 1982, pp.148-157.
- 83 SIU,W.C., and CHEN,C.F.: 'New Realisation Technique of High-Speed Discrete Fourier Transform Described by Distributed Arithmetic', IEE Proc., Vol.130, Pt.E, Computers and Digital Techniques, Nov. 1983, pp.177-182.
- 84 TAN,B.S., and HAWKINS,G.J.: 'Speed-Optimised Microprocessor Implementation of a Digital Signal Filter', IEE Proc., Vol.128, Pt.E, May 1981, pp.85-93.
- 85 CHU.S., and BURRUS,C.S.: 'A Prime Factor FFT Algorithm using Distributed Arithmetic', IEEE Trans., Vol.ASSP-30, 1982, pp.217-226.
- 86 GOERTZEL,G.: 'An Algorithm for the Evaluation of Finite Trigonometric Series', Am. Math., Monthly, 1958, Vol.65, pp.34-35.
- 87 CURTIS,T.E., and VICKENDEN,J.T.: 'Hardware-base Fourier

- Transforms: Algorithms and Architectures', IEE Proc., Vol.130, Pt.F, 1983, pp.423-432.
- 88 SIU,W.C., and CONSTANTINIDES,A.G.: 'Hardware Realisation of Mersenne Number Transforms for Fast Digital Convolution', Proc. IEEE, ICASSP, 1984, San Diego, pp.41A.2.1-4
- 89 SIU,W.C., and CONSTANTINIDES,A.G.: 'Efficient Implementation of Cyclic Convolutions by Mersenne Number Transforms', (Submitted to IEE Proc., Part F, for possible publication).
- 90 KWANG,K.: 'Computer Arithmetic: Principles, Architecture, and Design', (John Wiley & Sons, 1979).
- 91 BURRUS,C.S., and ESCHENBACHER,P.W.: 'An In-place, In-order Prime Factor FFT Algorithm', IEEE Trans., Vol.ASSP-29, 1981, pp.806-817.
- 92 ARAMBEPOLA,B.: 'Discrete Fourier Transform Processor based on the Prime-factor Algorithm', IEE Proc., Vol.130, Pt.G, No.4, 1983, pp.138-144.
- 93 SIU,W.C., and CONSTANTINIDES,A.G.: 'Efficient Hardware Realisation of Number Theoretic Transform Convolvers using ROMs', in preparation.
- 94 SIU,W.C., and CONSTANTINIDES,A.G.: 'Very Fast Discrete Fourier Transforms', A paper presented at the IEE Seminar on Digital Filter, London, 7-8th April, 1983, pp.1-29.
- 95 SIU,W.C., and CONSTANTINIDES,A.G.: 'Very Fast Discrete Fourier Transform using Number Theoretic Transform', IEE Proc., Pt.G, Electronic Circuits and Systems, 1983, Vol.130, pp.201-204.

- 96 SIU,W.C., and CONSTANTINIDES,A.G.: 'On the Computation of Discrete Fourier Transform using Fermat Number Transform', IEE Proc., Vol.131, Pt.F, Communications, Radar and Signal Processing, No.1, Feb. 1984, pp.7-14.
- 97 GENTLEMAN,W.M., and SANDE,G.: 'Fast Fourier Transforms for Fun and Profit', Fall Joint Comput. Conf., AFIPS Proc., Vol.29, Washington, DC:Sparton, 1966, pp.563-578.
- 98 NUSSBAUMER,H.J., and QUANDALLE,P: 'Fast Computation of Discrete Fourier Transforms using Polynomial Transforms', IEEE Trans., 1979, ASSP-27, pp.169-181.
- 99 REED,I.S., and TRUONG,T.K.: 'A New Hybrid Algorithm for Computing a Fast Discrete Fourier Transform', IEEE Trans., 1979, Vol.C-28, pp.487-492.
- 100 REED,I.S., and TRUONG,T.K.: 'A Fast DFT Algorithm using Complex Integer Transforms', Electronics Letters, 1978, Vol.14(6), pp.191-193.
- 101 REED,I.S., TRUONG,T.K., and BENJAUTHRIT,B: 'A Fast Hybrid Algorithm for Computing a Discrete Fourier Transform', Proc. IEEE, ICCS, 1980, pp.186-187.
- 102 SIU,W.C., and CONSTANTINIDES,A.G.: 'Some New Results on the Computation of Discrete Fourier Transforms using Mersenne Number Transforms', Proc. IEEE, International Symposium on Circuits and Systems, pp.53-56, Montreal, Canada, May 1984.
- 103 SIU,W.C., and CONSTANTINIDES,A.G.: 'Fast Mersenne Number Transforms for the Computation of Discrete Fourier Transforms', To be published in Signal Processing, European Association for Signal Processing.

- 104 BHATTACHARYA,M., and AGARWAL,R.C.: 'Number Theoretic Techniques for Computation of Discrete Convolution', IEEE Trans., Vol.ASSP-32, 1984, pp.507-511.
- 105 WU,Y.S.: 'A Common Operational Software (ACOS), approach to a Signal Processing Development System', Proc., IEEE, ICASSP, Boston, 1983, pp.1172-1175.
- 106 WU,Y.S., CONSTANTINIDES,A.G., CURTIS,T.E., and WU,L.J.: 'Architectural Approach to Alternate Low-level Primitive Structures (ALPS) for Acoustic Signal Processing', IEE Proc., Vol.131, Part F, No.3, 1984, pp.327-333.
- 107 TAYLOR,F.J.: 'Residue Arithmetic: A Tutorial with Examples', IEEE Computer, Vol.17, No.5, May 1984, pp.51-62

LIST OF PUBLICATIONS OF THE AUTHOR ON WHICH
THIS THESIS IS BASED

1. SIU,W.C., and CONSTANTINIDES,A.G.: 'Very Fast Discrete Fourier Transform', Paper presented at the IEE Seminar on Digital Filters, London, 7th-8th April, 1983, pp.1-29.
2. SIU,W.C., and CONSTANTINIDES,A.G.: 'Very Fast Discrete Fourier Transform using Number Theoretic Transform', IEE Proc., Vol.130, Pt.G, Electronic Circuits and Systems, No.5, Oct. 1983, pp.201-204.
3. SIU,W.C., and CONSTANTINIDES,A.G.: 'Approach to the Hardware Implementation of Digital Signal Processors using Mersenne Number Transforms', IEE Proc., Vol.131, Pt.E, Computers and Digital Techniques, No.1, Jan. 1984, pp.10-18.
4. SIU,W.C., and CONSTANTINIDES,A.G.: 'On the Computation of Discrete Fourier Transform using Fermat Number Transform', IEE Proc., Vol.131, Pt.F, Communications, Radar and Signal Processing, No.1, Feb. 1984, pp.7-14.
5. SIU,W.C., and CONSTANTINIDES,A.G.: 'Hardware Realisation of Mersenne Number Transforms for Fast Digital Convolution', Proc., IEEE International Conference on Acoustics, Speech and Signal Processing, San Diego, U.S.A., March 1984, pp.41A2.1-4.
6. SIU,W.C., and CONSTANTINIDES,A.G.: 'Some New Results on the Computation of Discrete Fourier Transforms using Mersenne Number Transforms', Proc., IEEE International Symposium on Circuits and Systems, Montreal, Canada, May 1984, pp.53-56.
7. SIU,W.C., and CONSTANTINIDES,A.G.: 'Cyclic Convolution of Long

Sequences using Number Theoretic Transform', IEE Proc., Vol.131, Pt.G, Electronic Circuits and Systems, No.3, June 1984, pp.119-126.

8. SIU,W.C., and CONSTANTINIDES,A.G.: 'Efficient Implementation of Cyclic Convolutions by Mersenne Number Transforms', Submitted to IEE Proc., Pt.F, Communications, Radar and Signal Processing.
9. SIU,W.C., and CONSTANTINIDES,A.G.: 'Fast Mersenne Number Transforms for the Computation of Discrete Fourier Transform', To be published in Signal Processing, European Association for Signal Processing.
10. SIU,W.C., and CONSTANTINIDES,A.G.: 'Efficient Hardware Realisation of Number Theoretic Convolvers using ROMs' in preparation.

LIST OF PRESENTATIONS CONDUCTED BY THE AUTHOR
WITHIN THIS RESEARCH PERIOD

1. , Speaker
IEE Seminar on Digital Filters, London England, 7th-8th April, 1983.
2. Speaker
IEEE International Conference on Acoustic, Speech, and Signal Processing, San Diego U.S.A., 19th-21st March, 1984.
3. Speaker
IEEE Seminar on 'A Preview of Some U.K. Contributions to the International Symposium at Montreal, 1984', U.K. and Republic of Ireland Section, IEEE Circuits and Systems, London England, 20th April, 1984.
4. Speaker
IEEE International Symposium on Circuits and Systems, Montreal Canada, 7th-10th May, 1984.