1979

Mini-computer Fermat number transform and application to digital signal processing.

Ah-Hin. Kok
University of Windsor

MINI-COMPUTER ORIENTED FERMAT NUMBER TRANSFORM AND APPLICATION TO DIGITAL SIGNAL PROCESSING

by

© Ah-Hin Kok

A Thesis
submitted to the Faculty of Graduate Studies through
the Department of Electrical Engineering in partial
fulfillment of the requirements for the
Degree of Master of Applied Science at
The University of Windsor

Windsor, Ontario, Canada

1979

ABSTRACT

This thesis investigates a Fermat Number Transform suitable for implementation with a mini-computer, its fast transform algorithms and applications to digital signal processing.

The transform considered is the FNT with modulus equals to the 4th Fermat number. From the investigation of its properties and the characteristics of the FFT-type fast transform algorithms, an efficient algorithm is developed. This algorithm makes possible a more efficient evaluation of the butterfly computations which are the major computations in the fast transform algorithms.

The realization of this transform is made on a 16-bit mini-computer. An efficient scheme is also developed to incorporate the diminished-1 coding technique to avoid ambiguity in number representation.

Application of this FNT to both one-dimensional and two-dimensional digital signal processing is investigated. The efficiency of the developed algorithm is tested with examples on digitized speech signal low pass filtering and Image enhancement.

ACKNOWLEDGEMENT

The auther wishes to express his appreciation to Dr. J. J. Soltis for his advice and assistance in this work. The auther also expresses his gratitude to Dr. G. A. Jullien and Dr. W. C. Miller for their suggestion and encouragement. Thanks also goes to other members of the department who assisted the auther.

TABLE OF CONTENTS

		PAGE
ABSTRACT		I
ACKNOWLEDGEMENTS		II
TABLE OF CONTENTS		III
CHAPTER 1:	INTRODUCTION	1
1-1	Discrete-time system	2
1-2	Linear, Time-invariant system	2
1-3	Linear Convolution	3
1-4	Periodic Convolution	3
1-5	Linear Convolution via Periodic Convolution	4
1-6	Discrete Fourier Transform	4
1-7	Cyclic Convolution Property of DFT	5
1-8	The Fast Fourier Transform	5
1-8-1	Decimation-in-time Algorithm	6
1-8-2	Decimation-in-frequency Algorithm	8
1-9	Efficiency of Convolution via FFT	11
1-10	The Number Theoretic Transform	11
1-11	Cyclic Convolution Property of NTT	12
1-12	Outline of this Work	13
CHAPTER 2:	THE FERMAT NUMBER TRANSFORM	15
2-1	Introduction	15
2-2	The Fermat Number Transform	17
2-3	Properties of FNT	19
2-4	Choice of a Suitable FNT	21
2-5	Fast Transform Algorithms	23
2-5-1	DIT-type Algorithm	23
2-5-2	DIF-type Algorithm	26
2-6	Modified Fast Algorithm (the Shift-Diminished Algorithm)	29
CHAPTER 3:	REPRESENTATION OF NUMBER AND IMPLEMENTATION	33
3-1	Number Representation	33
3-2	The Diminished-1 Number Representation	35
3-3	Binary Arithmetic Operation	35
3-3-1	Negation	35
3-3-2	Addition	36
3-3-3	Subtraction	37

3-3-4	Multiplication by Power of 2	37
3-3-5	General Multiplication	38
3-4	Implementation with NOVA mini-computer	39
3-4-1	The Arithmetic Unit	40
3-4-2	Code Translation and Mapped Memory Block for MSB	45
3-4-3	Consideration on Possible Error due to Code Translation	46
3-4-4	Computational Modules for Multiplication by Powers of 2	48
3-4-5	Algorithms for Forward and Inverse Transforms	51
CHAPTER 4:	APPLICATION TO DIGITAL SIGNAL PROCESSING...	53
4-1	Convolution using the FNT	53
4-2	Some Techniques for Weakening the Restrictions	57
4-2-1	One-dimensional Convolution by Multi-dimensional Method	57
4-2-2	Wordlength Sectioning	59
4-2-3	Impulse Response Sectioning	60
4-3	One-dimensional Signal Filtering	61
4-3-1	Example of 1-D Signal Filtering	63
4-4	Correlation via FNT	65
4-5	Two-dimensional Signal Filtering	69
4-5-1	Two-dimensional Fermat Number Transform ...	69
4-5-2	Two-dimensional Convolution via 2-D FNT ...	71
4-6	Picture Processing	73
4-7	Examples of Picture Processing Application	74
4-7-1	Edge Enhancement	78
CHAPTER 5:	SUMMARY AND CONCLUSION	81
APPENDIX	....	87
REFERENCES	....	107

CHAPTER I

INTRODUCTION

Finite digital convolution has many useful application in signal processing, such as digital signal filtering, the calculation of auto- and cross correlation. The direct computation of digital convolution is a very time consuming operation due to large number of multiplications required. However, by means of the fast transform techniques it can be implemented with much higher efficiency.

The Discrete Fourier Transform (DFT) is a well known transform that can be used to perform digital convolution efficiently via the Fast Fourier Transform (FFT) algorithms. one of the other transforms that can be utilized for efficient computation of digital convolution is the Number Theoretic Transforms (NTT). The NTT is a class of transforms unveiled fairly recently. It is an analogy to the DFT, but whose structure is based on the ring of integers modulo a certain integer.

This work investigates the implementation of a particular NTT, called the Fermat Number Transform (FNT), using a general purpose mini-computer. An algorithm that improves the efficiency of the transform in software implementation is developed. For ease of distinguish, this algorithm is called Fermat Number Transform Shift Diminished Algorithm (FNTSD). The efficiency of this algorithm is tested with its applications in both one- and two-dimensional signal filtering.

In this chapter, some of the definitions and theories with regard to fast implementation of finite digital convolution will be reviewed. The outline of this work will then be described.

1-1 Discrete-time System

A discrete-time system is essentially a transformation, $T(.)$, that maps a number sequence $x(n)$, representing the input discrete-time signal, into another number sequence $y(n)$, representing the output discrete-time signal. This is usually denoted as $y(n)=T(\ x(n)\ )$ and depicted as in FIG. 1-1.

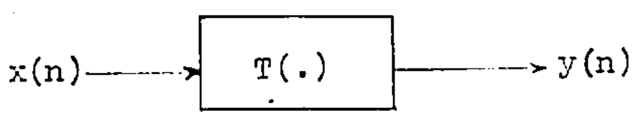

FIG.1-1

1-2 Linear, Time-invariant System

A linear, time-invariant system (LTI system) is characterized by the properties that

(a) if $T(\ x_1(n)\ ) = y_1(n)$

$T(\ x_2(n)\ ) = y_2(n)$

then $T(ax_1(n)+bx_2(n)\ ) = aT(\ x_1(n)\ )+bT(x_2(n)\ )$

$$= ay_1(n)+by_2(n)$$

(b) if $T(x(n)) = y(n)$

then $T(x(n-m)) = y(n-m)$.

where a and b are arbitrary constant, and m an integer.

The input and output of an LTI system has a convolutional relation.

1-3 Linear Convolution

Let h(n) and x(n) be two sequences with some values defined within the finite durations of N_1 and N_2 respectively, and zero otherwise. The linear convolution of h(n) and x(n) to give another sequence y(n) is defined as

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

$$= \sum_{m=0}^{N-1} h(n)x(n-m) \qquad , N = N_1+N_2 \quad , n = 0,1,2,\ldots$$

....(1.3. 1)

1-4 Periodic Convolution (Cyclic Convolution)

If h(n) represents one period section of the periodic sequence $h_p(n)$, and x(n) represents that of the periodic sequence $x_p(n)$ of both period N samples, Then the periodic convolution of h(n) and x(n) to give the sequence y(n) is defined as (where ((.)) denotes (.)mod N .)

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

or $$y(n) = \sum_{m=0}^{N-1} h_p(m)x_p(n) \qquad \text{for } n = 0,1,2,\ldots N-1$$

....(1.4.1)

1-5 Linear Convolution via Periodic Convolution

Let the sequences h(n) and x(n) be of duration M and L respectively, If h(n) and x(n) are appended with zero-valued samples to the length of N=M+L, Then the linear convolution of h(n) and x(n) can be obtained from the periodic convolution of h(n) and x(n), i.e.,

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

$$= \sum_{m=0}^{N-1} h(n-m)x(m) \qquad \ldots(1.5.1)$$

1-6 Discrete Fourier Transform (DFT)

The Discrete Fourier Transform of a sequence x(n) of N samples is defined as

$$x(k) = \sum_{n=0}^{N-1} x(n)W^{nk} \qquad k = 0,1,2,\ldots,N-1 \qquad \ldots(1.6.1)$$

and the inverse transform (IDFT) is defined as

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X(k)W^{-nk} \qquad n = 0,1,2,\ldots,N-1 \qquad \ldots(1.6.2)$$

where $W = \exp(-j2\pi/N)$

1-7 Cyclic Convolution Property of DFT

Let the sequences H(k) and X(k) be the DFT of the sequences h(n) and x(n) respectively, and both are of length N samples. The IDFT of the product H(k)X(k) is equal to the cyclic convolution of h(n) and x(n), i.e.,

$$\text{if} \quad X(k) = \sum_{n=0}^{N-1} x(n)W^{nk}$$

$$\text{and} \quad H(k) = \sum_{n=0}^{N-1} h(n)W^{nk}, \qquad k = 0,1,2,\ldots,N-1$$

$$\text{then} \quad y(n) = \sum_{k=0}^{N-1} H(k)X(k)W^{-kn}, \qquad n = 0,1,2,\ldots,N-1$$

$$= \sum_{m=0}^{N-1} h((n-m))x((m)) \qquad \ldots.(1.7.1)$$

where y(n) is the cyclic convolution of h(n) and x(n).

1-8 The Fast Fourier Transform (FFT)

The Fast Fourier Transform is a class of algorithms that make use of the symmetry and periodicity of the exponential weight $W=\exp(-j2\pi/N)$ to decompose a long DFT computation into smaller length DFT computations. In this way a significant reduce in the number of arithmetic operations can be obtained. Basically there are two types of FFT algorithms, call the decimation-in-time algorithm and the decimation-in-frequency algorithm.

1-8-1 Decimation-in-Time (DIT) Algorithm

Algorithms in which the decomposition is based on dividing the input sequence (time domain sequence) into successively smaller sequences for processing is called the decimation-in-time algorithms. The procedure is illustrated for an N-point sequence x(n), where N is an integer power of two.

By definition,

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

$$W_N = \exp(-j2\pi/N)$$

Seperating the time sequence x(n) into two sequences, $x_1(n)$ and $x_2(n)$, composed of the even- and odd-sample points respectively, we have,

$$X(k) = \sum_{r=0}^{N/2-1} x(2r) W_N^{2rk} + \sum_{r=0}^{N/2-1} x(2r+1) W_N^{(2r+1)k}$$

$$= \sum_{r=0}^{N/2-1} x(2r) (W_N^2)^{rk} + \sum_{r=0}^{N/2-1} x(2r+1) (W_N^2)^{rk} W_N^k$$

By $W_N^2 = \exp(2(-j2\pi/N))$

$= \exp(-j2\pi/(N/2))$

$= W_{N/2}$

$$X(k) = \sum_{n=0}^{N/2-1} x_1(n) W_{N/2}^{nk} + W_N^K \sum_{n=0}^{N/2-1} x_2(n) W_{N/2}^{nk}$$

$$X(k) = X_1(k) + W_N^k X_2(k)$$

$X_1(k)$ and $X_2(k)$ are DFT's of $x_1(n)$ and $x_2(n)$ respectively, and both are periodic of period N/2 samples. They have the relation,

$$X_1(k) = X_1(k-N/2) \qquad \text{for} \quad N/2 \leq k \leq N-1$$

$$X_2(k) = X_1(k-N/2) \qquad \text{"} \qquad \text{"} \qquad \text{"}$$

also $W_N^k = - W_N^{k-N/2}$.

Therefore

$$X(k) = X_1(k) + W_N^k X_2(k) \qquad \text{for} \quad 0 \leq k \leq N/2-1$$

----(1.8.1)

$$= X_1(k-N/2) + W_N^k X_2(k-N/2) \qquad \text{for} \qquad N/2 \leq k \leq N-1$$

The procedure is repeatedly applied to each of the successive subsequences, until only two-point DFT's are left to be evaluated. (1.8.1) forms the basic computation units and is usually called the 'butterflies'. For this illustration they have the flow graph of FIG. 1-8-1, and 1-8-2.

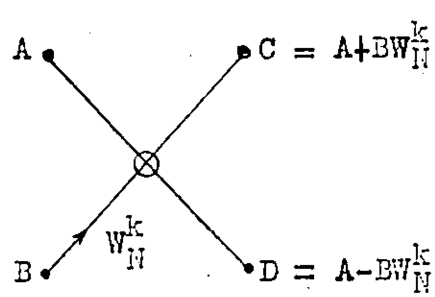

FIG 1-8-1

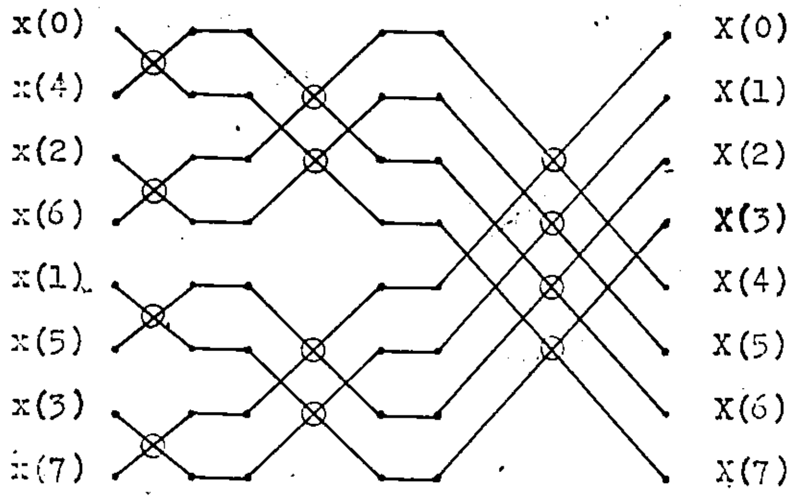

Fig. 1-8-2 Flow Graph for DIT Algorthm

1-8-2 Decimation-in-Frequency (DIF) Algorithm

DIF algorithms are based on appropriate combination of input points such that the output sequence can be successively divided into smaller subsequences for processing. This is illustrated for a sequence $x(n)$ of length N equals to an integer power of two.

If $x(n)$ is partitioned into two sequences $x_1(n)$ and $x_2(n)$ such that

$$x_1(n) = x(n)$$

and $\quad x_2(n) = x(n + N/2) \quad , \qquad n = 0,1,2,\ldots,N/2-1$

then the DFT of $x(n)$ is

$$X(k) = \sum_{n=0}^{N/2-1} x_1(n) W_N^{nk} + \sum_{n=0}^{N/2-1} x_2(n) W_N^{nk+Nk/2}$$

$$X(k) = \sum_{n=0}^{N/2-1} \left(x_1(n) + e^{-j\pi k} x_2(n) \right) W_N^{nk} \quad , \quad k = 0,1,2,\ldots N-1$$

Decompose X(k) into even- and odd-sample sequences, then we have

$$X(2k) = \sum_{n=0}^{N/2-1} \left(x_1(n) + x_2(n) \right) W_N^{2nk}$$

$$= \sum_{n=0}^{N/2-1} \left(x_1(n) + x_2(n) \right) W_{N/2}^{nk} \quad \ldots(1.8.3)$$

$$= \sum_{n=0}^{N/2-1} f(n)\, W_{N/2}^{nk} \quad , \quad k = 0,1,2,\ldots,N/2-1$$

and

$$X(2k+1) = \sum_{n=0}^{N/2-1} (x_1(n) - x_2(n))\, W_N^{n(2k+1)}$$

$$= \sum_{n=0}^{N/2-1} \left((x_1(n) - x_2(n)) W_{N/2}^{nk} \right) \quad \ldots(1.8.4)$$

$$= \sum_{n=0}^{N/2-1} g(n)\, W_{N/2}^{nk}, \quad k = 0,1,2,\ldots N/2-1$$

(1.8.3) and (1.8.4) are equivalent to two N/2-point DFT's. The procedure is repeatedly applied to each of the even- and odd-sample output subsequences until finally only two-point DFT's are left to be evaluated. (1.8.3) and (1.8.4) indicate the basic computational units (butterflies) for the DIF algorithms. FIG. 1-8-3 and FIG. 1-8-4 show the flow graphs of a butterfly and the overall algorithm for N equals to eight.

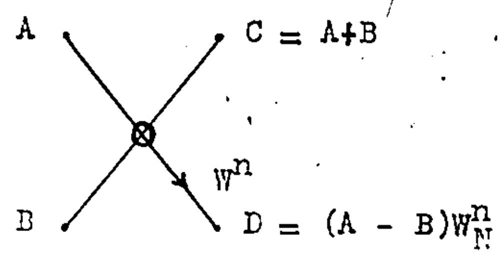

FIG.1-8-3 Butterfly for DIF Algorithm

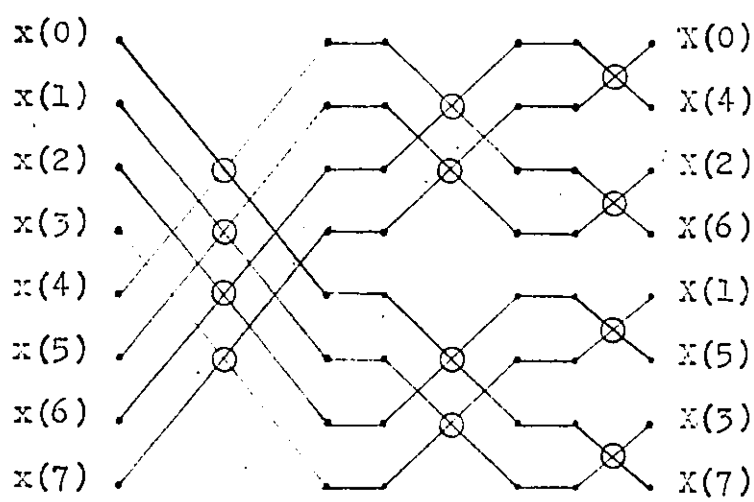

FIG.1-8-4 Flow Graph for DIF Aligorithm

1-9 Efficiency of Convolution via FFT.

In both the Dit and DIF algorithms illustreted for sequence length equals to an integer power of two, the number of complex multiplications and additions required for one transform is reduced to $(N/2)\log_2 N$ and $N \log_2 N$ respectively. while a direct DFT computation requires N^2 complex multications and additions.

If a sequence of duration N samples is to be linearly convolved with another sequence of the same duration and assume its DFT already known, then by using the method mentioned in section 1-5, the number of complex multiplications is approximately equal to $2N \log_2 N + 4N$ and that of additions equals to $4N \log_2 N + 4N$. In contrast to the direct convolution, N^2 real multiplications and additions are required.

Assume that 1 complex number multiplication takes as much time as 4 real number multiplications and 3 real number additions, and that 1 complex number addition that takes as much time as 2 real number additions, gain in efficiency can be obtained for convolution via DFT for sequence length greater than approximately 64 points.

1-10 The Number Theoretic Transform (NTT)

The general structure of NTT and its inverse transform is given by the transform pair,

$$X(k) = \left(\sum_{n=0}^{N-1} x(n)a^{nk}\right) \bmod M, \qquad k = 0,1,2,\ldots,N-1$$

$$x(n) = \left(Q \sum_{k=0}^{N-1} X(k)a^{-nk}\right) \bmod M, \qquad n = 0,1,2,\ldots,N-1 \qquad \ldots.(1.10.1)$$

Where,

a is a root of unity of order N in the ring of integer modulo an integer, M.

a^{-nk} is the multiplicative inverse of a^{nk} mod M, and Q is the multiplicative inverse of N mod M. Q is often denoted as N^{-1}.

1-11 Cyclic Convolution Property of NTT

Let the sequences H(k) and X(k) be the NTT of the N- point sequences h(n) and x(n) respectively, and that magnitude of the cyclic convolution of h(n) and x(n) does not exceed $|M/2|$, where M is the modulus of the NTT. The inverse NTT of the product H(k) X(k) mod M is equivalent to the cyclic convolution of h(n) and x(n). In notation, that is

$$X(k) = \left(\sum_{n=0}^{N-1} x(n)a^{nk}\right) \bmod M$$

$$H(k) = \left(\sum_{n=0}^{N-1} h(n)a^{nk}\right) \bmod M$$

If $\quad Y(k) = (H(k)\ X(k)) \bmod M,$

$$y(n) = \left(Q\sum_{k=0}^{N-1} Y(k)a^{-nk}\right) \bmod M$$

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

Where $-M/2 \leq y(n) \leq M/2$,

$(a^N) \bmod M = 1$

and $((\cdot))$ indicate $(\cdot) \bmod N$

If there exists an NTT such that 'a' has a simple binary representation such as 2, and N is a highly composite integer, also that the modulo M operation can be easily done, then such an NTT can be implemented efficiently using the algorithms similar to that of FFT. The arithmetic operations required for such an NTT can be only additions and binary bit-shifts. It can be used for fast computation of cyclic convolution.

1-12 Outline of This Work

This work investigates the efficient software implementation of a particular NTT called the Fermat Number Transform defined in [1] and [2] by Rader, and Agarwal and Burrus. It is found that the symmetric and periodic properties of the basis function, a^k, can be utilized to reduce the number of bit-shifts required when multiplying the data points with higher power of a. This reduces the power of 'a' in various butterfly computations from the range of (0 to N/2-1) to the range of (0 to N/4). This enables considerable decrease in computation time required in software implementation. Efficient algorithm is developed. It is applied to the implementation of an FNT with modulus $M = 2^{16} + 1$ and $a = \sqrt{2} \bmod M$.

The computer used is the NOVA 840 mini-computer. It is a 16-bit machine with 32K main memory and 80K extended memory.

The FNT of modulus $M = 2^{16} + 1$ requires 17 bits to represent all the possible date quantities. The diminished-1 (D-1) code, proposed by Leibowitz [4] is utilized for unambiguous arithmetic operations with the 16-bit word computer. An efficient scheme is developed to accomplish this coding technique in the software realization of the FNT for error free processing.

Subroutine programs are written in assembly language for the FNT of transform length of 64 points and for multiplication of data sequences in the transformed domain. The efficiency of the improved fast FNT algorithm is compared to that of the unmodified FNT and FFT algorithms.

The applicability of the FNT in signal processing is investigated. Particularly, its practical application in digital image filtering is examined. FORTRAN programs are written for two dimentional FNT and two dimensional non-recursive filter using overlap-save technique.

A field of integers mod M --

A set of integers, Sm, each of whose elements is congruent mod M to some integer in the set $\{0,1,2, \ldots, M-1\}$. Sm is closed under addition, subtraction, multiplication as well as division, i.e., each element in Sm has a multiplicative inverse in Sm. Thus a field is also a ring.

Rules and Theorems for operations in a Ring of Integer mod M

With '≡' denotes congruent mod M, the following statement statements hold [6]

-- $a + b \equiv b + a$; $ab \equiv ba$ ... commutative

-- $(a + b) + c \equiv a + (b + c)$; $(ab)c \equiv a(bc)$ ... associative

-- $a(b + c) \equiv ab + ac$ ... distributive

-- $a + 0 \equiv a$; $a(1) \equiv a$

-- For every a there is an element, (-a), such that $a + (-a) \equiv 0$

-- $a \equiv b \Rightarrow ca \equiv cb$

-- $a \equiv b \Rightarrow a - b \equiv 0$; $b - a \equiv 0$

-- $a \equiv b \Rightarrow a^n \equiv b^n$

-- $a \equiv b,\ c \equiv d \Rightarrow a + c \equiv b + d$

-- $a \equiv b,\ c \equiv d \Rightarrow ar + cs \equiv br + ds$

-- $a \equiv b,\ c \equiv d \Rightarrow ac \equiv bd$

-- A quantity, p, has a multiplicative inverse, q, mod M if p and M are mutually prime.

CHAPTER 2

THE FERMAT NUMBER TRANSFORM

2-1 Introduction

Fairly recently, the transforms having the structure similar to that of the DFT have been defined on finite rings or fields of integers with arithmetic operations performed modulo an integer [1] , [2] . This class of transforms is generally referred to as Number Theoretic Transform (NTT). The Fermat Number Transform (FNT) is a subclass of NTT defined with the modulus equals to a Fermat number.

In the following, some definitions and theorems relevant to the later descriptions are given.

Definitions

Congruence - -.

Two integers, a and b, are said to be congruent modulo M, denoted as $a \equiv b \pmod M$, if $a = b + kM$, where k is some integer and M is the modulus.

A ring of integers mod M - -

A set of integers, Sm, each of whose elements is congruent to some integer in the set $\{0,1,2,\ldots,M-1\}$, where M is the modulus. Sm is closed under addition, subtraction and multiplication, i.e., the sum, difference or product of Sm's elements is in Sm.

2-2 The Fermat Number Transform

For a sequence x(n) of length N samples, a general transform pair of the form given by,

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

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X(k)\, a^{-nk}\ , \quad n = 0,1,2,\ldots,N-1 \qquad \ldots(2.2.1)$$

is said to have the DFT structure.

In [2], [3], it is shown that if N^{-1}, the multiplicative inverse of N, exists, and if a is a root of unity of order N, i.e., for the least positive integer, N, such that

$$a^N = 1$$

then the transform pair (2.2.1) possess the cyclic convolution properties.

If the sequence x(n) is of integer values and (2.2.1) is defined on the ring of integers mod some integer, M, with the imposed conditions hold, i.e., if N is the least positive integer, and

$$X(k) = \left(\sum_{n=0}^{N-1} x(n)\, a^{nk} \right) \bmod M\ , \quad k = 0,1,2,\ldots,N-1$$

$$x(n) = \left(Q \sum_{k=0}^{N-1} X(k)\, a^{-nk} \right) \bmod M\ , \quad n = 0,1,2,\ldots,N-1$$

$$QN = 1 \bmod M\ ; \quad a^N = 1 \bmod M \qquad \ldots.(2.2.2)$$

it is the general form of the Number Theoretic Transform.

Rader has shown that [1], [5], if the modulus is a Fermat number, i.e.,

$$M = F_t = 2^b + 1 \ , \quad b = 2^t \ , \quad t = 1,2,3,\ldots$$

then for $a = 2$, $N = 2b$ is the least positive integer such that

$$a^N = 1 \bmod M \ ,$$

and Q, the multiplicative inverse of N, exists.

This transform with modulus equals to a Fermat number is called the Fermat Number Transform (FNT). For this case the transform sequence length, N, is equal to 2b. Since this transform has the DFT structure with N equals to an integer power of two, it can be computed with the FFT type fast algorithms. Multiplication by the basis function, the power of two, can be performed by simple binary bit shifting.

In [2], Agarwal and Burrus defined an FNT using

$$a = 2^{b/4}(2^{b/2}-1).$$

It is shown that this value of a is of order 4b for $t \geq 2$. The sequence length for this transform is $N = 4b$.

Since $a^2 = 2 \bmod F_t$, this value of a is denoted as

$$a = \sqrt{2} \bmod F_t.$$

2-3 Properties of FNT

Some important properties of FNT are listed below (2), where the sequence, $X(k)$, is the FNT of the sequence, $x(n)$.

(a). Periodicity

$x(n)$ and $X(k)$ can be periodically extended,

$$x(n+N) = x(n)$$

$$X(k+n) = X(k)$$

(b). Symmetry

If $x(n)$ is symmetric, i.e.,

$$x(n) = x(-n) = x(N-n)$$

then, $X(k) = X(-k) = X(N-k)$

If $x(n)$ is antisymmetric, i.e.,

$$x(n) = -x(-n) = -x(N-n)$$

then, $X(k) = -X(-k) = -X(N-k)$

(c). Symmetry of the transform pair

$$FNT\{FNT[x(n)]\} = Nx(-n)$$

(d). Shift theorem

If $FNT[x(n)] = X(k)$

then, $FNT[x(n+m)] = X(k)a^{-mk}$

(e). Fast computation algorithm

If N can be factored as

$$N = R_1 \cdot R_2 \cdot R_3 \cdots R_m$$

then a fast computational algorithm similar to that of FFT exists.

(f) Cyclic convolution property

If $X(k) = FNT[x(n)]$

$H(k) = FNT[h(n)]$

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

then $y(n) = IFNT[Y(k)]$

$= x(n) \; (*) \; h(n)$

Where (*) denotes cyclic convolution.

2-4 Choice of a suitable FNT

The following factors affect the choice of a suitable FNT for convolution application.

(a). The computer word length

If the computer is of b-bit word length, where $b = 2^t$, $t = 1,2,\ldots,$ then the most suitable modulus would be $F_t = 2^b+1$. This enable the efficient arithmetic operations modulo F_t with the b-bit Arithmetic-Logical Unit.

(b). The transform length

This affects the appropriate selection of the weight, a. The transform length, N, and the weight, a, are restricted by the conditions imposed on the transform pair (2.2.2.). So, N can be chosen to be equal to or greater than the desired transform length, such that N is highly composite and a has a simple binary representation. Such N and a enable the use of the FFT-type fast algorithms.

(c). Dynamic range of the sequence magnitude

This factor has less control over the choice of the FNT, as the existence of the FNT is governed by the existence of the appropriate F_t, a, N and N^{-1}. Since the magnitude representable by the ring of integers mod F_t ranges from $-F_t/2$ to $F_t/2$, the transform is valid if the magnitude of the convolved sequence is less than $|F_t/2|$.

For thr purpose of convolution application using a 16-bit mini-computer, the fourth Fermat number, $F_4 = 2^{16}+1$, is most suitable to be considered as the modulus. Taking

$$a = 2^{12} - 2^4 = \sqrt{2} \bmod F_t = 4080$$

$$F_t = 2^{16} + 1$$

the sequence length for transform is $N = 64$.
This should be the optimal FNT with regard to the word length of the mini-computer and the practical value of the transform with the associated possible transform length and dynamic range. The available digital signal processing facility for this work is configured around a 16-bit mini-computer. The investigation thus concentrates on the efficient implementation of the FNT using this modulus and basis function.

In the following sections, the FFT-type fast algorithms will be illustrated. An efficient algorithm, resulted from studying the symmetric and periodic properties of the basis function, is then developed.

2-5 Fast Transform Algorithms

FNT is similar in structure to DFT. The way of decimating a long DFT computation into shorter DFT computations can be applied to FNT. This is illustrated for the FNT of basis function, a, and transform length, N, equal the integer powers of two.

2-5-1 DIT-type Algorithm

By definition, the FNT of a sequence x(n) of length, N, is given by,

$$X(k) = \left(\sum_{n=0}^{N-1} x(n) a^{nk} \right) \bmod F_t \,, \qquad k = 0,1,2,\ldots,N-1$$

where N is the least positive integer such that

$$a^N = 1 \bmod F_t \quad ,$$

and $F_t = 2^b+1 \quad , \quad b = 2^t \quad , \quad t = 1,2,3,\ldots$

Let $x_e(n)$ and $x_0(n)$ be two sequences of length, N/2, composed of the even- and odd-samples of x(n) respectively. Then,

$$X(k) = \left(\sum_{r=0}^{N/2-1} x(2r) a^{2rk} + \sum_{r=0}^{N/2-1} x(2r+1) a^{(2r+1)k} \right) \bmod F_t$$

$$= \left(\sum_{n=0}^{N/2-1} x_e(n) (a^2)^{nk} + a^k \sum_{n=0}^{N/2-1} x_o(n) (a^2)^{nk} \right) \bmod F_t \qquad \ldots.(2.5.1)$$

The right side of (2.5.1) is the sum of two N/2-point FNT's with the second one weighted with a^k, i.e.,

$$X(k) = X_e(k) + a^k X_o(k)$$

$X_e(k)$ and $X_o(k)$ are the FNT of $x_e(n)$ and $x_o(n)$ respectively, and are periodic of period N/2. They have the relation,

$$X_e(k) = X_e(k-N/2) \qquad \text{for} \quad N/2 \leq k \leq N-1$$

$$X_o(k) = X_o(k-N/2) \qquad \text{for} \quad N/2 \leq k \leq N-1$$

Also $a^k = -(a^{k-N/2}) \bmod F_t$

Therefore

$$X(k) = (X_e(k) + a^k X_o(k)) \bmod F_t \qquad \text{for} \quad 0 \leq k \leq N-1$$

$$= (X_e(k-N/2) - a^k X_o(k-N/2)) \bmod F_t \qquad \text{for} \quad N/2 \leq k \leq N-1$$

The procedure is repeatedly applied to each of the successive subsequences, until only two-point FNT's are left to be evaluated. The FNT computational unit, or the 'butterfly', has the flow graph shown in FIG.2-5-1. The overall flow graph for the DIT-type FNT algorithm of $F_t = 2^4+1$, $a = 2$ and $N = 8$ is shown in FIG. 2-5-2.

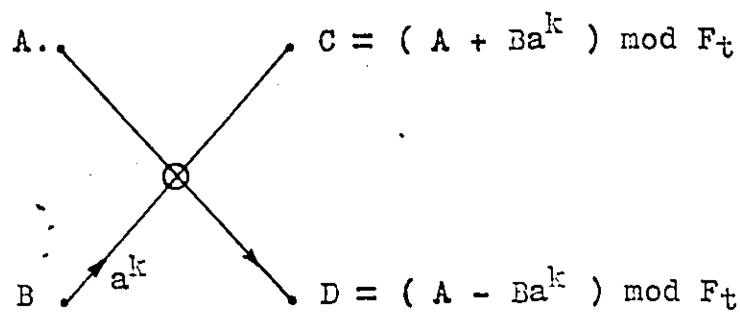

FIG. 2-5-1 The Computational Unit, 'Butterfly', for DIT-type FNT Algorithm.

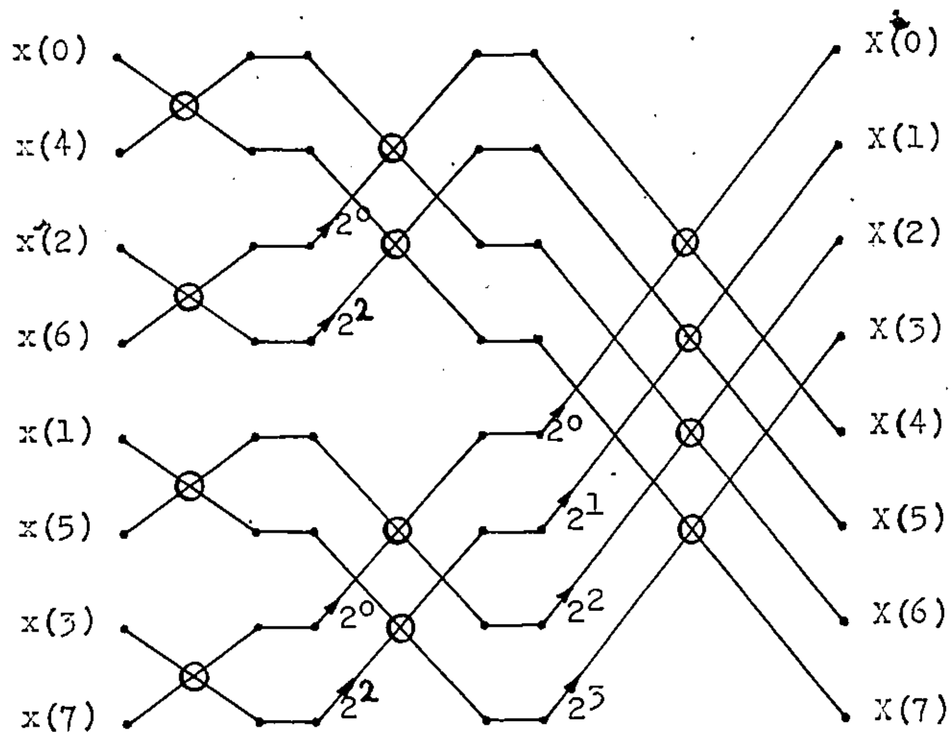

FIG. 2-5-2 Flow graph of the DIT-type 8-point FNT Algorithm. (Computations are carried out mod 17).

2-5-2 DIF-type Algorithm

Let $x(n)$ be partitioned into two sequences, $x_1(n)$ and $x_2(n)$, such that,

$$x_1(n) = x(n)$$

$$x_2(n) = x(n + N/2) \qquad \text{for} \quad n = 0,1,2,\ldots,N/2-1$$

The FNT of $x(n)$ is reduced to

$$X(k) = \left(\sum_{n=0}^{N/2-1} x_1(n)\, a^{nk} \right) \bmod F_t + \left(\sum_{n=0}^{N/2-1} x_2(n)\, a^{(n+N/2)k} \right) \bmod F_t$$

$$= \left(\sum_{n=0}^{N/2-1} x_1(n)\, a^{nk} \right) \bmod F_t + \left(a^{Nk/2} \sum_{n=0}^{N/2-1} x_2(n)\, a^{nk} \right) \bmod F_t$$

where $k = 0,1,2,\ldots,N-1$

The two summations at the right of equal sign can be combined before modulo F_t. Decomposing $X(k)$ into the even- and odd-sequences, we have for even $X(k)$,

$$X(2k) = \left(\sum_{n=0}^{N/2-1} (x_1(n) + a^{Nk}x_2(n))(a^2)^{nk} \right) \bmod F_t.$$

Since $a^{Nk} = (1 \bmod F_t)^k = 1 \bmod F_t$,

$$\therefore \quad X(2k) = \left(\sum_{n=0}^{N/2-1} (x_1(n)+x_2(n)(a^2)^{nk} \right) \bmod F_t \quad \ldots(2.5.2)$$

For odd-X(k),

$$X(2k+1) = \left(\sum_{n=0}^{N/2-1} (x_1(n)+a^{Nk+N/2}x_2(n))a^{n(2k+1)} \right) \bmod F_t$$

Since $a^{Nk+N/2} = -1 \bmod F_t$

$$\therefore \quad X(2k+1) = \left(\sum_{n=0}^{N/2-1} ((x_1(n)-x_2(n))a^n) \ (a^2)^{nk} \right) \bmod F_t \quad \ldots.(2.5.3)$$

(2.5.2) and (2.5.3) indicate that the even- and odd-samples of the FNT of x(n) can be obtained from the FNT's of $(x_1(n) + x_2(n))$ and $(x_1(n) - x_2(n))a^n$ respectively. This procedure is applied repeatedly until finally two-point FNT's are left to be evaluated.

(2.5.2) and (2.5.3) form the basic computational unit, or the 'butterfly', of the DIF-type algorithms. Its flow graph representation is shown in FIG. 2-5-3. The overall flow graph of the DIF-type algorithm of $F_t = 2^4+1$, $a = 2$ and $N = 8$ is shown in FIG. 2-5-4.

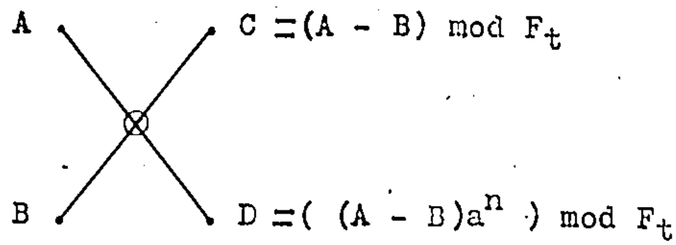

FIG. 2-5-3 Butterfly for DIF-type FNT Algorithm

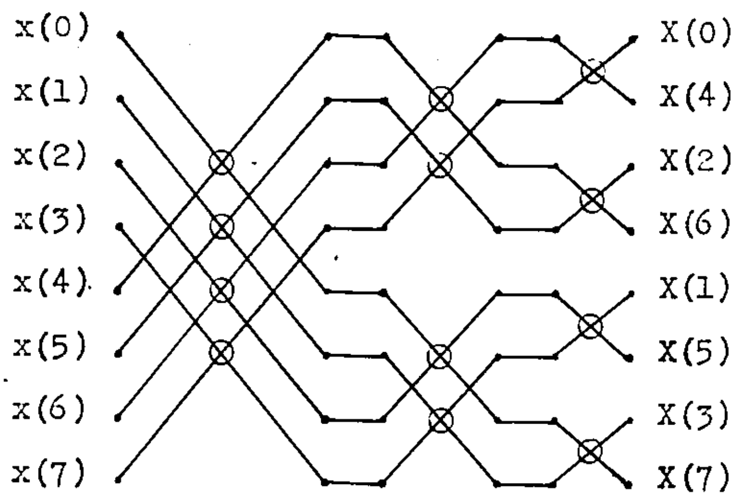

FIG. 2-5-4 Flow Graph for DIF-type 8-point FNT Algorithm. (Computation for each data point is carried out mod 17).

2-6 Modified Fast Algorithm (the Shift-Diminished Algorithm)

In both the DIT- and DIF-type of FNT algorithms illustrated, each butterfly requires multiplication of data by the basis function, a^k. The values of k varies from 0 to N/2-1.

In their software implementation of FNT using $a = 2$ and $a = \sqrt{2} \bmod F_t$, (2), Agarwal and Burrus suggested that Multiplication by 2^k can be performed by shifting the binary bit pattern of the data by k bit positions, instead of multiplication of the data by the value of 2^k. This results in considerable saving in butterfly computation time.

Usually a general-purpose digital computer does not have the instruction for performing a single multi-bit shift; multiplication by 2^k has to be done by k repertitions of one-bit shift instruction. Although bit-shifting is a simple operation, it still needs considerable amount of time when k is large. This situation can be improved if we make use of the periodic and symmetric properties of the basis function.

Consider the fact that,

$$a^{N/2} = -1 \bmod F_t ,$$

then a^k can be written as

$$a^k = (a^{N/2} \bmod F_t)(a^{k-N/2} \bmod F_t)$$

$$= (a^{N/2} a^{k-N/2}) \bmod F_t$$

$$a^k = -a^{-(N/2-k)} \bmod F_t \quad \text{. . . . (2.6.1)}$$

Thus the DIT-type butterfly computation units can be modified to

$$X(k) = (X_e(k) + a^k X_0(k)) \bmod F_t \quad \text{for} \quad 0 \leq k \leq N/4$$

$$= (X_e(k) - a^{-(N/2-k)} X_0(k) \bmod F_t$$

$$\text{for} \quad N/4 < k \leq N/2-1$$

and

$$X(k+N/2) = (X_e(k) - a^k X_0(k) \bmod F_t$$

$$\text{for} \quad 0 \leq k \leq N/4$$

$$= (X_e(k) + a^{-(N/2-k)} X_0(k)) \bmod F_t$$

$$\text{for} \quad N/4 < k \leq N/2-1$$

While in the DIF-tyoe algorithm, the computations can be modified to;

$$X(2k) = \sum_{n=0}^{N/2-1} (x_1(n)+x_2(n))\ (a^2)^{nk})\ \mathrm{mod}\ F_t$$

for $0 \leq n \leq N/2-1$

and

$$X(2K+1) = \sum_{n=0}^{N/2-1} ((x_1(n)-x_2(n))a^n)\ (a^2)^{nk})\ \mathrm{mod}\ F_t$$

for $0 \leq n \leq N/4$

$$= \sum_{n=0}^{N/2-1} (-(x_1(n)-x_2(n))\ a^{-(N/2-n)})\ (a^2)^{nk})\ \mathrm{mod}\ F_t$$

for $N/4 < n \leq N/2-1$

For $k \geq N/4$, $(N/2-k) \leq N/4$.

Hence, multiplication by a^k with k greater than N/4 can be replaced by multiplication by $-a^{-(N/2-k)}$. This modification enables the number of bit-shifts in various butterfly computations to be reduced to the range of (0 to N/4) instead of 0 to (N/2-1) as required in the algorithm given in [1] and [2].

Take for example, the FNT of modulus $F_t = 2^{16} + 1$ with a = 2 and N = 32 , the computation of $A.2^{14} \bmod F_t$ can be done by,

$$A.2^{14} = (-A).2^{-(16-14)} \bmod F_t$$

$$= (-A).2^{-2} \bmod F_t \quad ;$$

two binary bit shifts are required rather than 14 bit shifts.

A general-purpose computer usually has the instruction for performing half-word swap. Further reduction in computation time is possible with this instruction to compute multiplication of data by 2^k when $k = b/2$, where b is the computer wordlength.

The details of implementation by these two techniques will be described in chapter 3.

CHAPTER 3

REPRESENTATION OF NUMBER AND IMPLEMENTATION

To implement an FNT of modulus $F_t=2^b+1$ with a b-bit computer, the following are desirable.

-- Efficient operation of residue reduction modulo F_t.

-- Efficient operation of multiplication by the basis function.

-- Error free operation.

-- Arithmetic operations on b+1-bit numbers in the ring of integers modulo F_t should be avioded to enable efficient computation.

3-1 Number Representation

Arithmetic operations in the computation of FNT are carried out modulo F_t. This restricts the numbers involved in the computation to be within the set of integers, $\{0,1,2,\ldots,2^b\}$. Zero and positive values are represented by 0,1,2, $\ldots,2^b/2$, and negative values by $(2^b/2)+1,(2^b/2)+2,\ldots,2^b$ each of which is equivalent to adding 2^b+1 to each respective negative number. The mágnitude of the transform sequence should be less than or equal to $|2^b/2|$, to avoid aliasing in magnitude when it is carried out modulo F_t.

To represent all the possible number in binary, b+1 bits are required. But the computer used is of b-bit (b = 16 in this work) wordlength. Agarwal and Burrus[4], in their software realization of FNT with a b-bit ($b = 32$) computer, simplified the computation to b-bit arithmetics which enables efficient residue reduction. This involves using b bits to represent numbers from 0 to 2^b-1 in the way described in the previous paragraph. All number whose values do not exceed $|2^b/2|$ can be represented exactly except the value (-1). Since $-1 = 2^b \bmod F_t$, the number 2^b required b+1 bits. It can not be represented with a b-bit computer word. So, if it is encountered in the input data, it can be rounded to 0 or -2. This is equivalent to introducing some quantization error. But if 2^b appears as a result of some arithmetic operations, it cannot be represented. Erroneous output may result for that block.

In order to have error free computation, consideration has to be made on appropriate representation of numbers. Some special coding schemes have been introduced by several authors for special hardware FNT implementations,(4),(9),(10) Among these coding schemes. the "Deminished-1 number representation" proposed by Leibowitz (4) is most simple and applicable to this work. Arithmetics with this diminished-1 number representation will be described. Feasible technique is developed to accomplish this coding scheme in the software implementation of the FNT.

More specifically, the advantage of using the Diminished-1 number representation are: (a) the problem of ambiguity in number representation can be overcome and (b), operation on 17 bits numbers can be avoid ed, enabling efficient computation with 16 bits CPU.

3-2 The Diminished-1 Number Representation

In the set of integers, $Sm = \{1,2,3,\ldots,2^b+1\}$, the numbers required b+1 bits binary representation are 2^b and 2^b+1. 2^b is congruent to -1 mod F_t and 2^b+1 congruent to 0 mod F_t. If every element in Sm is subtracted by 1, a new set, $Sd = \{0,1,2,\ldots,2^b\}$ is obtained. The elements in Sm and Sd are one to one correspondence, with 2^b in Sd corresponds to $2^b+1 = 0$ mod F_t in S_m, 0 in S_d corresponds to 1 in Sm, and so on. Sd can be used to represent all the possible integers allowed in FNT computation, and is called the "Diminished-1 number representation" [4].

The only number in Sd with a 1 in the $b+1_{th}$ bit (or the most significant bit, msb) is 2^b, which represents $2^b+1 = 0$ mod F_t in Sm. Zero can be exampted from calculation. Thus using this number representation, arithmetic operations on b+1-bit numbers can be avoided. Table 3-4-1 illustrates the correspondence between the normal and diminished-1 number representations for $b = 4$.

3-3 Binary Arithmetic Operations

The following illustrates the arithmetic operations necessary in computing FNT using the diminished-1 number representation as given in [4].

3-3-1 Negation

A negative integer, -A, is represented by (-A)-1 It is obtained by complementing the b-lsb (b-least-significant-bits) of its diminished-1 positive counter-part, A-1

Denoting the b-lsb complement by an overbar, this is shown as,

$$\overline{A - 1} = 2^b - 1 - (A-1)$$

$$= 2^b + 1 - A - 1$$

$$= (-A) - 1 \bmod F_t$$

A 1 at the msb indicates the number is the representation of zero, and negation is inhibited.

Example

$$-4 = 13 \bmod F_t$$

$$0\ \overline{0011} = 0\ 1100$$

3-3-2 Addition

The sum of two dimished-1 integers, A-1 and B-1, is

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

The required sum in diminished-1 representation is (A+B)-1, thus

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

The above indicates a 1 should be added to the addition to get the correct diminished-1 representation of the sum.

A carry generated from the b-lsb addition indicates the result is greater than or equal to the modulus, thus a residue reduction mod F_t requires the subtraction of 1 from the result, because $2^b \equiv -1 \bmod F_t$. Therefore no corrective addition is necessary.

If the msb of either addend is 1, the addition is inhibited, and the remaining addend is the sum.

Example 1

```
   8               0 0111
 +14              +0 1101
  22 = 5 mod 17    1 0100
                     + 0
                   0 0100 -> 5 mod 17
```

Example 2

```
   0               1 0000
  +5              -0 0100
   5 mod 17        0 0100 -> 5 mod 17
```

3-3-3 Subtraction

Subtraction is performed by negating the subtrahend and adding it to the minuend according to the previous description.

3-3-4 Multiplication by power of two

The result of multiplying a diminished-1 number, A-1, by 2 is

$$(A-1).2 = (2A-1) - 1$$

The desired diminished-1 representation of the product is (2A-1), thus,

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

This is equivalent to left-shifting (A-1) by one bit position and a corrective addition of 1. If the bit shifted out of the b^{th} bit is a zero, a corrective 1 should be added to the result. If the bit shifted out is a 1, a residue reduction mod F_t requires the subtraction of 1 from the result, which cancels out the corrective addition, and the shifted b-lsb is the desired product.

Multiplication by a higher power of 2 is equivalent to a repetition of left-shifts and corrective additions.

If the multiplicant represents zero as indicated by a 1 at the msb, the multiplication is inhibited. The product is still zero.

Example

11	0 1010		
$11 \times 2 = 22 = 5 \bmod 17$	0 0100	→	5
$11 \times 2^2 = 44 = 10 \bmod 17$	0 1001	→	10
$11 \times 2^3 = 88 = 3 \bmod 17$	0 0011	→	3

3-3-5 General Multiplication

The product of two diminished-1 number, (A-1) and (B-1), is

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

The desired product in diminished-1 representation is (AB-1), therefore,

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

This is obtained by adding the diminished-1 sum of (A-1) and (B-1) to the product of (A-1) and (B-1), then perform the residue reduction by diminished-1 subtraction of the b-msb of the result from the b-lsb.

If the msb of either (A-1) or (B-1) is a 1, which indicates zero representation, multiplication is inhibited and the result is set to diminished-1 representation of zero.

Example

```
   14                     0 1101        0 1101
 x  8                   x 0 0111      + 0 0111
  112 = 10 mod 17       0101 1011       0 0100
                         + 0100 <-------'
                        0101 1111
                         + 1010
                          0 1001   --> 10 mod 17
```

3-4 Implementation with NOVA mini-computer

The underlying motivation in the implementation of FNT is its efficiency for computing discrete convolution; and this should be competible with the more versatile FFT. The most important consideration is whether fast operation is possible for the butterfly computational units. This requires an understanding of the capability of the computer used, particularly that of the arithmetic-and-logical unit (ALU). The following gives a brief description of the NOVA-840's arithmetic unit and instructions, on which the FNT software program is developed.

3-4-1 The Arithmetic Unit

The logical organization of the arithmetic unit is illustrated in FIG. 3-4-1.

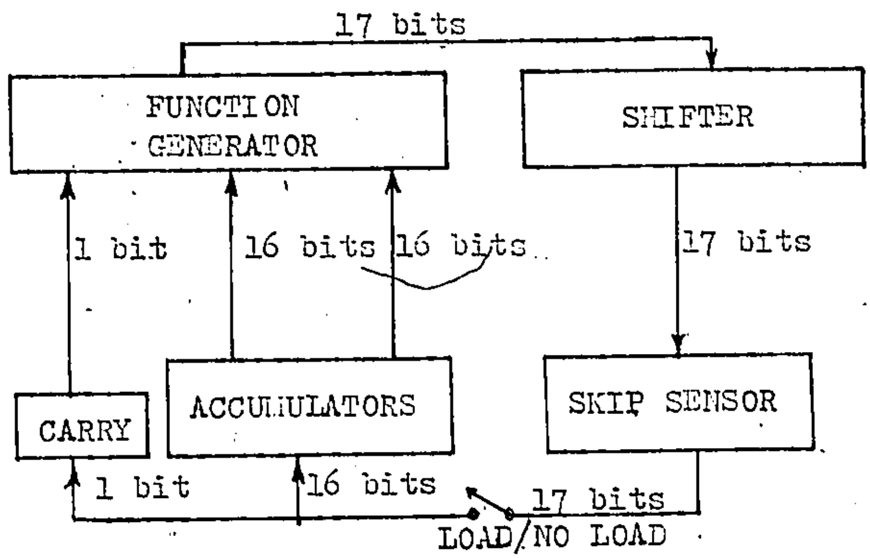

FIG. 3-4-1

The Accumulators

There are four 16-bit accumulators, AC0 to AC3, act as working and scratch-pad registers. Any one or two of them can be specified by an instruction to supply operands to the function generator. Data can be moved in either direction between any accumulator and memory location. AC2 or AC3 can also be used as an index register for addressing memory. These features are very useful for efficient butterfly computation. The pair of index registers make convenient the addressing of the data blocks. The other two accumulators do the computations and temporary store the intermediate result.

The Cary

It is a flag indicates the occurance of a carry bit out of the 16th bit in an arithmetic operation. It can be used to indicate an overflow that requires residue reduction modulo ($2^{16}+1$)..

The Shifter

The 17-bit result after the operation of the function generator can be shifted one bit position left or right. The left and right halves of the 16 lsb can also be swapped in the shifter. These are illustrated in FIG. 3-4-2. The combination of these two instructions enables fast operation of multiplications by various powers of 2.

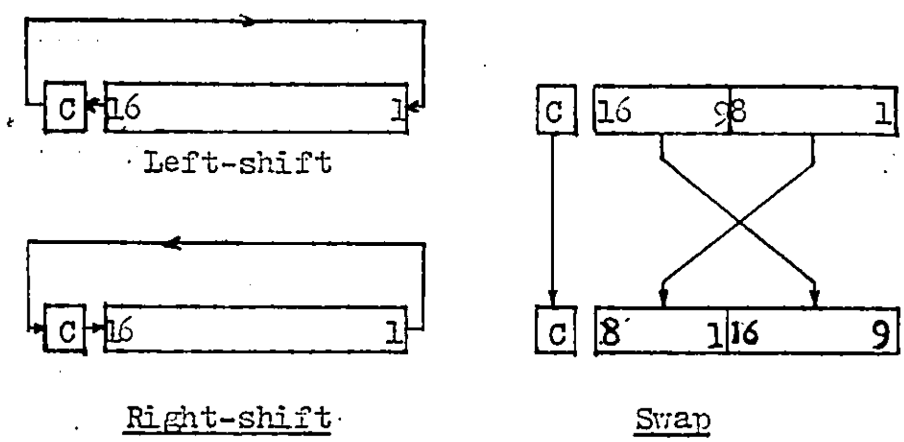

FIG. 3-4-2

The Skip Sensor

The content of the 17th bit and/or the other 16 bits can be tested for a skip over the next sequencial instruction. This function is required, or in associated with the Load/No load swith, for test for overflow that requires a residue reduction, test for zero or sign of numbers in code translation and branching of execution to different computational modules.

<u>The Function Generator</u>

The function generator performs the functions sepecified by the instruction. The arithmetic and logical instructions available are,

COM (complement)
NEG (negate)
MOV (move)
INC (increment)
ADC (add complement)
SUB (subtract)
ADD (add)
AND (and)

In each of the above instructions, the left or right shift, swap, skip conditions, base value of Cary and load/no load can be specified.

Other instructions available and required in programming the FNT are the memory reference instructions,

LDA (load accumulator from memory)
STA (store accumulator to memory)
ISZ (increment content of memory & skip if zero)
DSZ (decrement content of memory & skip if zero)
JMP (jump)
JSR (jump to subroutine)

3-4-2 Code Translation and Mapped Memory Block for MSB

In NOVA system, integer number are represented using 16 bits binary. The integer values which can be represented without ambiguity are $-2^{15},\ldots,-2,-1,0,1,2,\ldots,2^{15}-1$. Negative values are represented using two's complement representation, i.e., if A is a 16-digit binary number within $(1,2^{15})$, its two's complement is equivalent to $2^{16}-A$ which represents $-A$. The 16th bit stands for sign, 0 for positive and 1 for negative.

As can be seen from TABLE 3-4-1, the diminished-1 representation of negative values is exactly the same as the two's complement representation. Thus to translate the numbers in two's complement to diminished-1 representation, the negative numbers are left unchanged; only the positive numbers are to be subtracted by 1.

One problem arises in the representation of the diminished-1 numbers, zero and one, with 16-bit-word memory. The 16-lsb of these two numbers are both all o's; a means has to be set up to distinguish between them. In this work, a block of memory that maps to the block that stores the 16-lsb of the data is set up to stand for the msb (the 17th bit), such that the word in the mapped msb block corresponds to zero is set to 1 and all others are set to 0, as illustrated in FIG. 3-4-3. The increase in addressing complexity is trivial with the use of the accumulators, AC2 and AC3, as a pair of index registers for addressing the two blocks at the same time.

Value (mod 17)	Diminished-1 Representation	Two's Complement Representation (4-bit)
1	0 0000	0001
2	0 0001	0010
3	0 0010	0011
4	0 0011	0100
5	0 0100	0101
6	0 0101	0110
7	0 0110	0111
8	0 0111	
-8 (9)	0 1000	1000
-7 (10)	0 1001	1001
-6 (11)	0 1010	1010
-5 (12)	0 1011	1011
-4 (13)	0 1100	1100
-3 (14)	0 1101	1101
-2 (15)	0 1110	1110
-1 (16)	0 1111	1111
0 (17)	1 0000	0000

TABLE 3-4-1 Correspondence between Diminished-1 and Two's Complement Representations

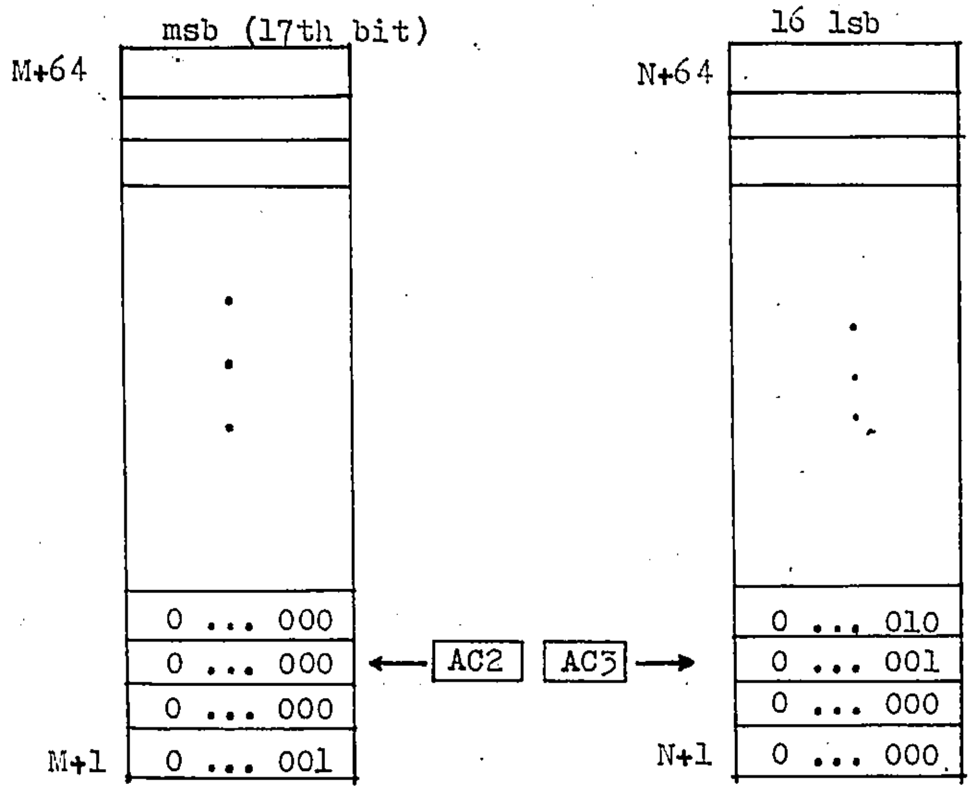

FIG.3-4-3 Two mapping blocks of memory for storage of 17-bit data sequence in diminished-1 representation. AC2 and AC3 are used as index register pair for addressing the two corresponding words.

3-4-3 Consideration on Possible Error due to Code Translation

In the Fermat Number Transform, there are 2^b+1 possible integers, which representd values within the range $-2^b/2$ to $2^b/2$. With the diminished-1 coding scheme they can all be represented, using an additional bit for the representation of zero. But with two's complement representation using b bits word, the range of integers representable is $-2^b/2$ to $(2^b/2-1)$; the positive integer, $2^b/2$, can not be represented. To avoid ambiguity without increasing the complexity in both computation and memory addressing, the mapped block of memory that stands for the msb's has to be maintained through out the procedure of computation for convolution, i.e.,starting from the two's complement-to-diminished-1 code translation, forward FNT, multiplication in the transformed domain, to the end of inverse transform. The possible error due to ambiguity in number representation may then happen during the code translation from diminished-1 back to two's complement representation. The representable dynamic range in various stage of the convolution proceduce is depicted in fig.3-4-4. If the diminished-1 number, $(2^b/2-1)$, happens to occur after the inverse FNT, the normal code translation will add 1 to the number, and the result in the b-bit register will be then $2^b/2$. But in two's complement $2^b/2$ represents the value $-2^b/2$ instead of $+2^b/2$; a change of sign occurs. The error is -2^b in magnitude. If this number is not to be translated, the error would be -1.

This error can be avoided. Consider the diminished-1 number, $(2^b/2-1)$, which represents $2^b/2$ in normal value for the result of the convolution. If the input sequences are properly scaled such that the magnitude of their convolution is less that $2^b/2$, then the diminished-1 number, $(2^b/2-1)$, will never occur at the end of the inverse FNT. Hence, ambiguity in code translation from diminished-1 code to 2's complement representation is avoided.

3-4-4 Computational Modules for Multiplication by Powers of 2

In the FFT-type algorithms with $F_t = 2^{16}+1$, $N = 64$ and $a = \sqrt{2} \bmod F_t$, there are six stages. Each stage requires $N/2 = 32$ butterfly computations. Each butterfly requires a multiplication by a power of $\sqrt{2}$. The power of $\sqrt{2}$ for each stage in the DIT-type algorithm are listed below,

Stage 1 (stage 6 for DIF-type)
$32 \times \{0\}$,

Stage 2 (stage 5 for DIF-type)
$16 \times \{0,16\}$,

Stage 3 (stage 4 for DIF-type)
$8 \times \{0,8,16,24\}$

Stage 4 (stage 3 for DIF-type)
$4 \times \{0,4,8,12,16,20,24,28\}$

Stage 5 (stage 2 for DIF-type)
$2 \times \{0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30\}$

Stage 6 (stage 1 for DIF-type)
$1 \times \{0,1,2,3,4,.........28,29,30,31\}$

Multiplication by an even power of $\sqrt{2}$ is equivalent to multiplication by half that power of 2, for example $(A.(\sqrt{2})^{16}) \bmod F_t = A.2^8 \bmod F_t$. Multiplication by an odd power of $\sqrt{2}$ is equivalent to multiplication by half of the next lower power of 2 and then by $\sqrt{2}$ mod F_t, for example, $A.(\sqrt{2})^{17} \equiv A.2^8.\sqrt{2}$ mod F_t.

By the modified algorithm, multiplications in various butterfly are reduced to multiplications by

$$\sqrt{2} \bmod F_t$$

$$(\sqrt{2})^2 \equiv 2 \quad \bmod F_t$$

$$(\sqrt{2})^3 \equiv 2/2 \bmod F_t$$

$$\vdots$$

$$(\sqrt{2})^{17} \equiv 2^8\sqrt{2} \bmod F_t$$

$$(\sqrt{2})^{18} \equiv 2^9 \equiv -2^{-(16-9)} \equiv -2^{-7} \bmod F_t$$

$$(\sqrt{2})^{19} \equiv 2^9\sqrt{2} \equiv -2^{-7}\sqrt{2} \bmod F_t$$

$$(\sqrt{2})^{20} \equiv 2^{10} \equiv -2^{-6} \bmod F_t$$

$$\vdots$$

$$(\sqrt{2})^{31} \equiv 2^{15}\sqrt{2} \equiv -2^{-1}\sqrt{2} \bmod F_t$$

In order to increase the efficiency, these operations are implemented in seperate modules within the program. Only two sets of test and indexing instructions are needed, one for updating the index of the stages, and the other for updating the address of the modules associated with the current butterfly. 16 modules are necessary, 15 of which for performing multiplications by 2^1, 2^2, 2^3, ..., 2^{15} mod F_t, and one of which for performing multiplication by ($\sqrt{2}$ mod F_t) $\equiv$ 4080.

For inverse FNT, the same set of computational modules can be used in the reversed order for multiplication by negative powers of $\sqrt{2}$.

The basic instruction set used to perform multiplication by 2 mod Ft, (having accumulator loaded with data to be operated), are

```
MOVZL   1,1,SNC
INC     1,1
```

If the probablity of a 1 shifting into the carry bit after the shift operation is 1/2, then the average execution time for this pair of instructions is $(1.0 + 0.8/2) = 1.4$ µsec.

If multiplication by 2^8 is performed by 8 repetitions of this instruction pair, $8 \times 1.4 = 11.2$ µsec is required. The operation can be performed, instead, by complementing the higher significant byte of the data followed by a swap of the two bytes, as illustrated in FIG. 3-4-5.

Data in AC1: $(15453)_{10}$

H	L
00111100	01011100

→ Compl. and move to AC0

H	L
11000011	10100011

Get low byte of AC1:

	L
00000000	01011100

Get high byte of AC0

H	
11000011	00000000

Add AC0 to AC1 and swap:

01011100	11000011

→ $15453 \times 2^8 = 23748 \bmod (2^{16} + 1)$

FIG. 3-4-5 Operation for Multiplication by 2^8 mod Ft.

The set of instructions to accomplish multiplication by 2^8 mod Ft is: (included in the program routine through MACRO command)

```
        ; data having loaded in Ac1, and content of AC3 is
          (11111111 00000000)2

COM     1,0         ;move compl. of C(AC1) to AC0
AND     3,0         ;get high sig. byte of AC0
COM     3,2
AND     2,1         ;get low sig. byte of Ac1
ADDS    0,1         ;add and swap
```

Instead of 11.2 μsec required by 8 repeated shifts, this module requires $5 \times 0.8 = 4.0$ μsec which is approximately equivalent to the time required for 3 repeated shifts.

The total number of bit-shifts for multiplication by powers of 2 in one foward FNT using the unmodified algorithm is 904. While the time required for the same operation with the modifications described is equivalent to the time required for approximately 389 single-bit shifts.

The execution time of a single-bit shift instruction is the same as that for one addition. The total number of additions (includes subtractions) for transform is $N\log_2 N = 64 \times 6 = 192$. Thus the average execution time for one "multiplication by a power of 2" is reduced to approximately twice that for addition.

3-4-5 Algorithms for Forward and Inverse Transforms

As can be seen from FIG. 2-5-2 and FIG. 2-5-4, the input sequence of DIF-type transform algorithm is in natural order, but the transformed sequence is in index's bit-reversed order; while it is the contrary for that of the DIT-type algorithm. Since the index order of the transformed sequence is not important, the routine for index sorting and reordering can be avoided by using DIF-type algorithm for forward FNT and DIT-type algorithm for inverse FNT computation. The number of program instructions increase is little compare to the program using the same algorithm for both forward and inverse transform with an index sorting and reordering routine.

CHAPTER 4

APPLICATION TO DIGITAL SIGNAL PROCESSING

The Fermat Number Transform cofficients do not have any physical meaning. The purpose of implementing the FNT is to utilize it as a tool to compute the finite convolution sum of two sequences. Convolution has many important applications in signal processing. The ability of an algorithm to compute convolution is also able to compute auto- and cross-correlations. This chapter discusses the practical application of the particular FNT investigated in this work. Emphasis is made on two-dimensional digital signal processing which could be most rewarding area of its applications. Some examples of image processing application are thus done to test the efficiency of the improved algorithm,(the shift-diminished algorithm,) developed in chapters two and three.

4-1 Convolution Using The FNT

The cyclic convolution of two sequences can be computed by multiplying their FNT's and followed by an inverse transform (IFNT). This is shown in the following.

If $X(k)$ and $H(k)$ are the FNT's of the two N-point sequences $x(n)$ and $h(n)$ respectively, then by definition,

$$X(k) = \left(\sum_{n=0}^{N-1} x(n)a^{nk} \right) \bmod F_t,$$

$$H(k) = \left(\sum^{N-1} h(n)a^{nk} \right) \bmod Ft , \qquad k = 0,1,\ldots,N-1 .$$

where $Ft = 2^{16} + 1 = 65537$, $a = \sqrt{2} \bmod Ft = 4080$

and $N = 64$.

Let $Y(k)$ be the product in the transform domain, defined as

$$Y(k) = (X(k)H(k)) \bmod Ft, \qquad k = 0,1,\ldots,N-1$$

and $y(n)$ be the IFNT of $Y(k)$, then

$$y(n) = \left(Q \sum_{k=0}^{N-1} Y(k)a^{-nk} \right) \bmod Ft, \qquad n = 0,1,\ldots,N-1$$

and $QN = 1 \bmod Ft$

or $$y(n) = \left(Q \sum_{k=0}^{N-1} \sum_{m=0}^{N-1} x(m)a^{mk} \sum_{r=0}^{N-1} h(r)a^{rk} a^{-nk} \right) \bmod Ft$$

$$= \left(\sum_{m=0}^{N-1} \sum_{r=0}^{N-1} x(m)h(r) \, Q \sum_{k=0}^{N-1} a^{-k(n-m-r)} \right) \bmod Ft$$

since $\left(Q \sum_{k=0}^{N-1} a^{-k(n-m-r)} \right) \bmod Ft = 1$ if $r = n-m$

$= 0$ otherwise

therefore $$y(n) = \left(\sum_{m=0}^{N-1} x(m)h(n-m) \right) \bmod Ft \qquad \ldots.(4.1.1)$$

If $x(n)$ and $h(n)$ are properly bounded such that

$\left(\sum_{m=0}^{N-1} x(m)h(n-m), \; n=0,1,\ldots,N-1 \right)$ lies within $-Ft/2$ and

$Ft/2$, then (4.1.1) has the equivalent relation, i.e.,

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

Advantages of Convolution via the FNT

The main advantage of convolution via the FNT is its efficiency. The reason for the speedup is that the FNT computation requires $N \log_2 N$ addition (includes subtractions) and $(N/2)\log_2 N$ "multiplications by powers of $\sqrt{2}$ mod F_t" which, by the shift-diminished algorithm developed in chapter 3, require a total execution time approximately equivalent to twice that for the additions.

Assume the FNT of one of the two convoluing sequences is known, the implementation of convolution by the FNT requires N multiplications mod F_t and a total of operations which requires an execution time approximately equivalent to the execution time required by $4N \log_2 N$ additions Mod F_t. While the direct computation of the convolution requires N^2 multiplications and N^2 additions.

Disadvantages of Convalution via the FNT

The short coming of the FNT results from restrictions due to the contraints of the Modulus, F_t, the basis function, a, and the transform length, N, For the particular FNT investigated, the application is thus subjected to the restrictions:

1). The transform sequence length should be less than or equal to 64 points and

2). The magnitude of the convolved sequence should be within 32767 and -32768 for signed number.

4-2 Some Techniques for Weakening the Restrictions

There are several techniques proposed by different authers possible for relaxing the limitations of FNT. This section discribes three of the techniques proposed specifically to cope with the constraints. Their practical values to the present case of study are investigated.

4-2-1 One-dimensional Convolution by Multi-dimensional Method

In [9] R. C. Agarwal and C. S. Burrus presented the formulation of multi-dimensional array from one-dimensional digital sequence, so that one-dimensional convolution could be obtained from the multi-dimensional convolution. The scheme to convolve long one-dimensional (1-D) sequences by two-dimensional (2-D) convolution is described.

Assume that x(n) and h(n) are the two sequences, both of duration N_1 points, to be convolved to give the sequence y(n), and that N_1 can be factorized as $N_1 = L \cdot M$. x(n) and h(n) are then arranged as shown in (4.2.1) and (4.2.2) to form the 2-D arrays of x_2 and h_2 respectively. The size of x_2 and h_2 are both N x N, where $N = 2L$, and $N \times N$ is the size of the two-dimensional FNT.

Two-dimensional cyclic convolution is then carried out by multiplying x_2 and h_2 in the transform domain and taking the two-dimensional inverse FNT. The result is

$$y_2(l,m) = x_2(l,m) * h_2(l,m), \qquad l,m = 0,1,\ldots,N-1$$

where ' * ' denotes 2-D cyclic convolution.

$$x_2(l,m) = \begin{bmatrix} x(0) & x(L) & \cdots & x(N_1-L) & 0 & 0 & \cdots & 0 \\ x(1) & x(L+1) & \cdots & x(N_1-L+1) & 0 & 0 & \cdots & 0 \\ x(2) & x(L+2) & \cdots & x(N_1-L+2) & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x(L-1) & x(2L-1) & \cdots & x(N_1-1) & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \quad \cdots (4.2.1)$$

$$h_2(l,m) = \begin{bmatrix} 0 & h(0) & h(L) & \cdots & h(N_1-L) & 0 & \cdots & 0 \\ 0 & h(1) & h(L+1) & \cdots & h(N_1-L+1) & 0 & \cdots & 0 \\ 0 & h(2) & h(L+2) & \cdots & h(N_1-L+2) & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & h(L-1) & h(2L-1) & \cdots & h(N_1-1) & 0 & \cdots & 0 \\ h(0) & h(L) & h(2L) & \cdots & 0 & 0 & \cdots & 0 \\ h(1) & h(L+1) & h(2L+1) & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ h(L-1) & h(2L-1) & h(3L-1) & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix} \quad \cdots (4.2.2)$$

$$y_2 = x_2 * h_2$$

The columns of the lower $L \times N$ of y_2 is the desired linear convolution of x and h, i.e.,

$$y(n) = y_2(l, m) , \qquad n = l + mL$$
$$l = 0,1,2,\ldots,L-1$$
$$m = 0,1,2,\ldots,N-1$$

For FNT with larger modulus, such as $F_t = F_5 = 2^{32} + 1$, this technique could be useful for lengthening the transform length. But for the FNT implemented with the 16-bit mini-computer, this technique for extending the transform length is greatly limited. This is because with 16-bit word, and modulus equal to $2^{16} + 1$, the dynamic range of the convolved result is limited to within $\pm 2^{15}$. The convolution of two 32-point sequences of magnitudes larger than 2^5 by using the original 64 points transform length could cause overflow. The extension of the transform length would make the problem of overflow even worse. For example, if the transform length is ~~to be~~ lengthen to 512 points, the magnitude of the input sequences, assume they have the same dynamic range, should be scaled down to about 2^3. This is too severe for most ~~of the~~ one-dimensional digital signal processing applications.

4-2-2 Wordlength Sectioning

The wordlength limitation could be relaxed by sectioning the input data word into shorter blocks and convolve them seperately. The results are then added to the proper scale to give the desired convolution. This is shown as follows [2].

$$x(n) = x_1(n).2^k + x_2(n), \qquad |x_2(n)| < 2^k$$

and $h(n) = h_1(n).2^k + h_2(n)$, $|h_2(n)| \angle 2^k$

Then

$$y(n) = x(n) * h(n)$$
$$= x_1(n)*h_1(n)2^{2k} + (x_1(n)*h_2(n) + x_2(n)*h_1(n))2^k$$
$$+ x_2(n)*h_2(n), \qquad n = 0,1,\ldots,N-1 \qquad \ldots.(4.2.3)$$

This method can increase the dynamic range considerably, and bring some light to the applicability of the FNT to 1-D digital signal processing.

4-2-3 Impulse Response Sectioning

N. S. Reddy and V. U. Reddy [10] proposed the partitioning of the impulse response sequence into shorter lengths. The segments of the impulse response sequence are then convolved seperately with the signal sequence. The seperate results are then added together with appropriate delays. This is shown in the following.

The impulse response, h(n), n=0,1,...,N-1, could be partitioned into M sections, each of length L points, such that

$$h_k(n) = h(kL + n) \qquad \text{for } n = 0,1,\ldots,L-1$$
$$= 0 \qquad \text{otherwise;}$$
$$k = 0,1,\ldots,M-1$$

The seperate convolutions yield $y_k(n) = x(n)*h_k(n)$. The desired result is

$$y(n) = y_0(n) + y_1(n-L) + y_2(n-2L) + \ldots + y_{M-1}(n-ML+L),$$
$$n = 0,1,2,\ldots$$

This technique can be used to extend the length of the impulse response sequence as well as the dynamic range. But the extension is rather limited. In practical situations, considerations may include the comparison of the resultant efficiency with that of the FFT technique. As will be seen in the later sections, convolution using the FNT without applying any of these extension techniques is 4 to 5 times faster than using FFT. Unless the exact computation is required, the impulse response sectioning of more than five sections is not attractive.

4-3 One-dimensional Signal Filtering

The realization of the a finite impulse response (FIR) filter involves the convolution of a digital signal sequence with a finite length impulse response sequence. The implementation could utilize the FNT for fast convolution.

The signal sequence, x(n), n = 0,1,2,..., can be considered to be infinitely long. Using the overlap-save or overlap-add technique [7], it can be partitioned into the appropriate length sequences, $x_k(n)$, n = 0,1,...,L-1, and convolve via the FNT with the finite length impulse response sequence, h(n), n = 0,1,...,N_1, as depicted in Fig.4-3-1, where FFNT denotes the forward FNT, IFNT denotes the inverse FNT and ⊗ means transform domain multiplication modulo 65537.

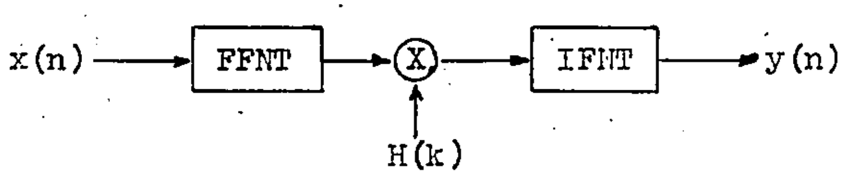

FIG.4-3-1

It should be noted that the sequences involved in the transform are integer valued. If the input sequences are originally not integers, modification is needed to ensure that they have integer values lying within the appropriate dynamic ranges, so that overflow would not occur at the output. The dynamic ranges could be set by the bound [2],

$$|x(n)|_{max} \sum_{n=0}^{N-1} |h(n)| < F_t/2$$

The wordlength limitation is rather severe with the modulus of 65537. One or the combination of the techniques described in the last section may be needed to release the restrictions on wordlength and transform length. The following situations could be considered.

1) Without wordlength or sequence length segmentation

For each lap of convolution, the maximum length of the sum of the two convolving sequences is 65 points. While the accuracy of the data, assuming it is the same for both sequences, could only be about 5 bits.

2) With wordlength segmentation

If the wordlength of both the transform sequences is segmented, the accuracy of the data could be raised to about 10 bits. That is, if the segmentation is such that

$$x(n) = x_1(n)2^5 + x_2(n), \qquad |x_1(n)| < 2^5, \quad |x_2(n)| < 2^5$$

and $$h(n) = h_1(n)2^5 + h_2(n), \qquad |h_1(n)| < 2^5, \quad |h_2(n)| < 2^5$$

then,

$$y(n) = x(n)*h(n)$$
$$= x_1*h_1.2^{10} + (x_1*h_2 + x_2*h_1).2^5 + x_2*h_2 \quad ....(4.3.1)$$

In (4.3.1), the summation within the parentheses could be done in the transform domain, and the last term which is very small compared with the first term, can be negleted. In this case, each lap of the convolutions requires 2 FFNT's, 3N multiplications and additions mod 65537, 2 IFFT's, 2N multiplications by 2^{10} and 2^5 and N additions. The execution time would be more than twice that for case one.

3) With both wordlength and sequence length segmentation

This xould further increase both the accuracy of the input data and filter length to some extend. But the efficiency may not be competitive with the FFT technique. The following example illustrates this.

Assume that the magnitude of both the input data, x(n), and impulse response, h(n), are of 10 bits; and that the impulse response is of 64 points long. The wordlength can be partitioned into two blocks such that

$$x(n) = x_1(n).2^5 + x_2(n), \qquad |x_1(n)|\ , |x_2(n)| \leq 2^5$$

$$h(n) = h_1(n).2^5 + h_2(n), \qquad |h_1(n)|\ , |h_2(n)| \leq 2^5$$

The impulse response can be partitioned into two subsequences such that

$$h'(n) = h(0),\ h(1),\ \ldots,\ h(31)$$

$$h''(n) = h(32),\ h(33),\ \ldots,\ h(63)$$

The seperately convolved results would be

$$y'(n) = h'(n) * x(n)$$

$$= x_1 * h_1' \cdot 2^{10} + (x_1 * h_2' + x_2 * h_1') \cdot 2^5 + x_2 * h_2'$$

$$y''(n) = h''(n) * x(n)$$

$$= h_1'' * x_1 \cdot 2^{10} + (x_1 * h_2'' + x_2 * h_1'') \cdot 2^5 + x_2 * h_2''$$

The desired result is

$$y(n) = y'(n) + y''(n)$$

Eight convolutions are required. Each convolution is performed via the 64-point FNT.

4-3-1 Example of 1-D Signal Filtering

This example shows the application of FNT technique to one-dimensional digital signal filtering. The input data sequence is digitized speech signal. The sampling interval of the speech signal is 100 usec. The dynamic range of the recorded signal is from -46 to 48. It is scaled down by multiplying the data by the factor, 0.6, and truncated to get the integer part. The filter used is a low pass filter. The normalized cutoff frequency (-3db) is 0.08 and the stop frequency is 0.16.

The original impulse response has a maximum value of o.233546 and minimum value of -0.038112. The multiplying factor for scaling up is 100, and the integer part is taken.

Overlap-save technique is used. Each lap of convolution processes 41 points of data.

Fig.4-3-2 and Fig.4-3-3 show the original signal waveform and the filtered waveform respectively.

4-4 Correlation via FNT

Consider the FNT and inverse FNT of the sequences, x(n) and y(n), n = 0,1,2,...,N-1, respectively.

$$X(k) = \left(\sum_{n=0}^{N-1} x(n)a^{nk} \right) \bmod F_t \qquad \text{.... (4.4.1)}$$

$$Y'(k) = \left(\sum_{n=0}^{N-1} y(n)a^{-nk} \right) \bmod F_t \qquad \text{.... (4.4.2)}$$

$$k = 0,1,\ldots,N-1$$

If $\quad R_{xy}(k) = \left(X(k)Y'(k) \right) \bmod F_t$

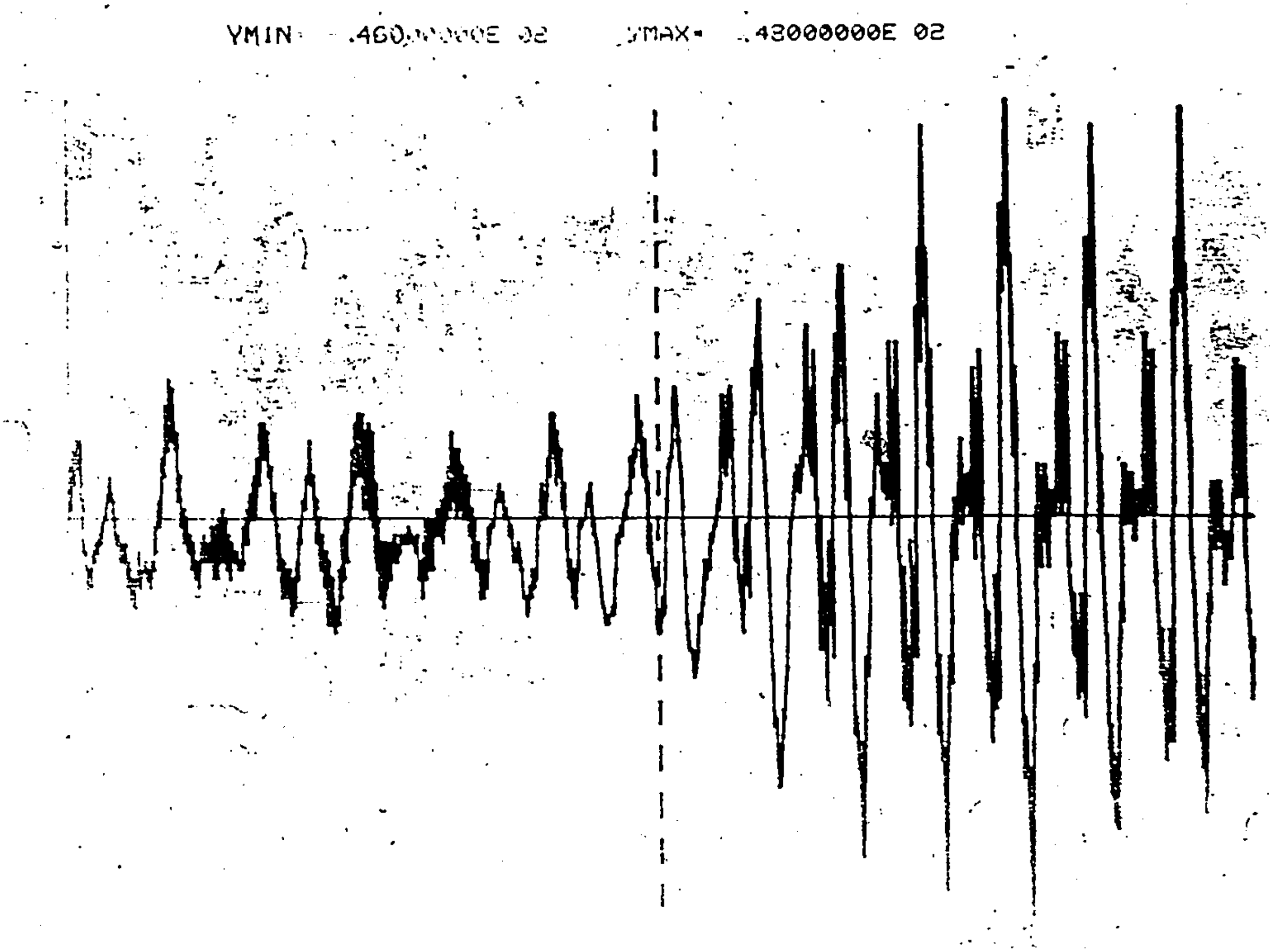

Fig. 4-3-2 A plot of a section of speech signal.

Number of points plotted: 1024

Sampling interval: 100 μsec.

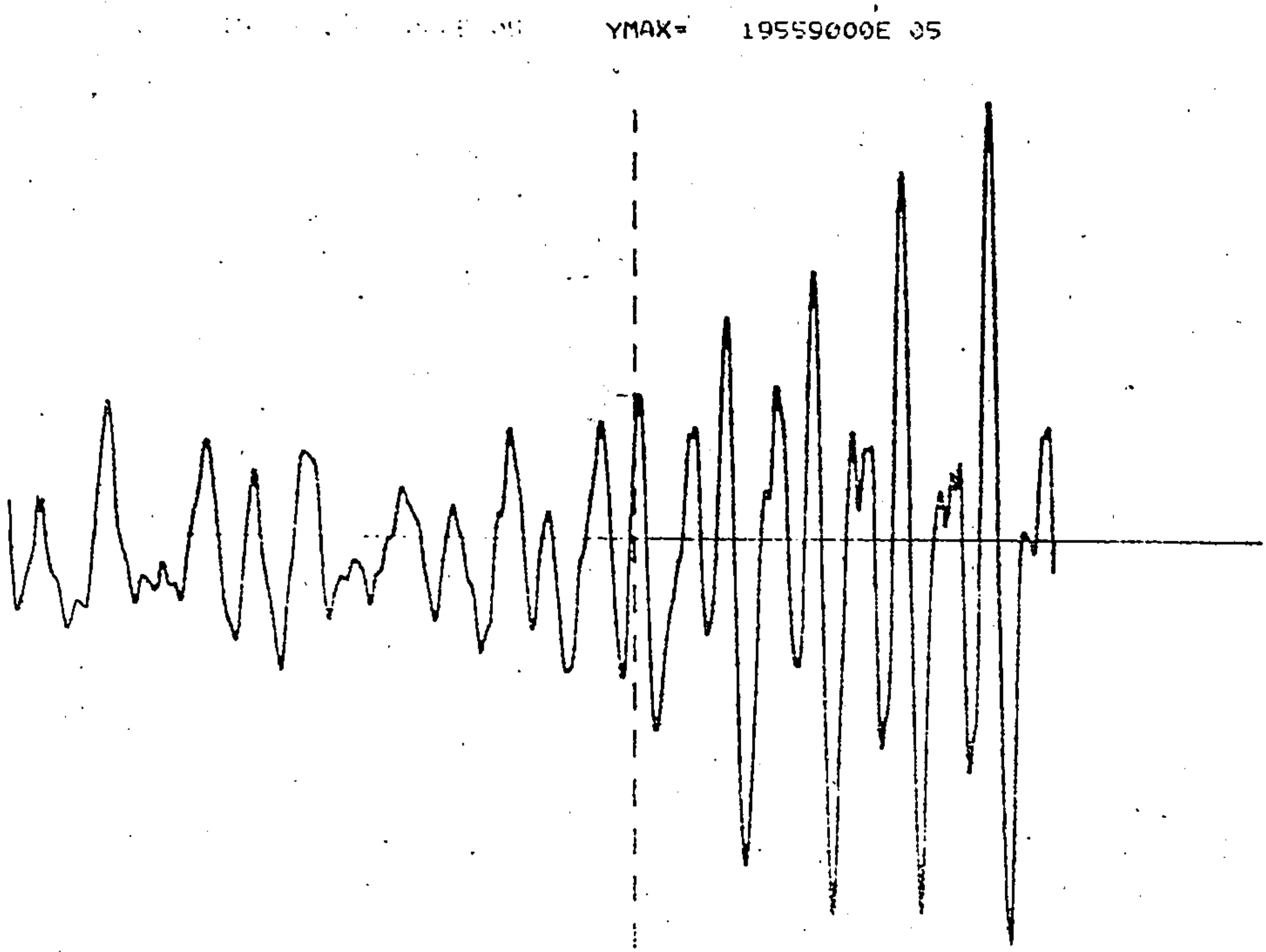

Fig.4-3-3 Speech signal after low pass filtering using FNT technique.

The original speech signal is shown in Fig.4-3-2.

-3 db frequency = 800 Hz.

the inverse FNT of $R_{xy}(k)$ is

$$r_{xy}(m) = \left(Q \sum_{k=0}^{N-1} R_{xy}(k) a^{-mk} \right) \bmod F_t, \quad m = 0,1,\ldots,N-1$$

$$= \left(Q \sum_{k=0}^{N-1} \sum_{p=0}^{N-1} x(p) a^{pk} \sum_{n=0}^{N-1} y(n) a^{-nk} a^{-mk} \right) \bmod F_t$$

$$= \left(\sum_{n=0}^{N-1} \sum_{p=0}^{N-1} x(p) y(n) \; Q \sum_{k=0}^{N-1} a^{-k(m+n-p)} \right) \bmod F_t$$

But $\left(Q \sum_{k=0}^{N-1} a^{-k(m+n-p)} \right) \bmod F_t = 1 \quad$ if $\; p = n+m$

$= 0 \quad$ otherwise

Therefore $\quad r_{xy}(m) = \sum_{n=0}^{N-1} x(n+m) y(m)$

or $\quad = \sum_{n=0}^{N-1} x(m) y(n+m) \quad \ldots.(4.4.3)$

(4.4.3) is the cyclic correlation of the sequences, $x(n)$ and $y(n)$, $n = 0,1,2,\ldots,N-1$.

If $x(n)$ has been constructed in such a way that only the first $(N/2 + 1)$ points are the actual sample values, and the rest rest of the points are zeros, then the sequence $r_{xy}(m)$, $m = 0,1,2,\ldots,N/2$, is the non-cyclic cross-correlation of $x(n)$ and $y(n)$.

In (4.4.2), if $Y'(k)$ and $y(n)$ are replaced by $X'(k)$ and $x(n)$ respectively, then by the same evaluation as for (4.4.3), the auto-correlation of $x(n)$ can be obtained, which is

$$r_{xx}(m) = \sum_{n=0}^{N-1} x(m) x(n+m), \qquad 0 \leq m \leq N/2$$

4-5 Two-dimensional (2-D) Signal Processing

The Fermat Number Transform can be extended to two-dimensional case. It could be particularly useful for the realization of two-dimensional finite impulse response filter, where the filter size and signal dynamic range are small. The two-dimensional Fermat Number Transform can be used as a tool to comput the two-dimensional convolution of the 2-D input signal and the 2-D filter sequence, as depicted in Fig.4-5-1.

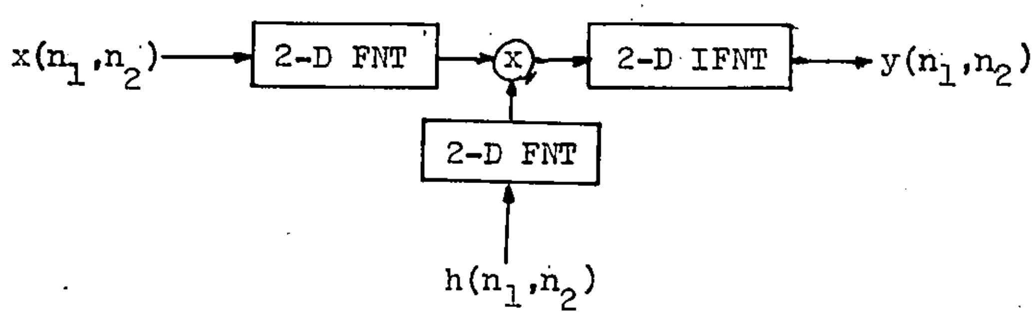

FIG. 4-5-1

4-5-1 Two-dimensional Fermat Number Transform(2-D FNT)

The 2-D FNT can be defined as an extension of the 1-D FNT. If $x(n_1,n_2)$ is assumed to be one period of a periodic 2-D signal with period N_1 points along both dimensions. The 2-D FNT of $x(n_1,n_2)$ is

$$X(k_1,k_2) = \left(\sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1,n_2) a^{n_1 k_1} a^{n_2 k_2} \right) \bmod F_t$$

$$k_1, k_2 = 0,1,\ldots,N-1$$

where $F_t = 2^{16} + 1 = 65537$

$a = \text{mod } F_t = 4080$

$N = 64$

The two-dimensional inverse Fermat Number Transform (2-D IFNT) of $x(k_1,k_2)$ is

$$x(n_1,n_2) = \left(QQ \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{N-1} x(k_1,k_2)\, a^{-n_1k_1}\, a^{-n_2k_2} \right) \text{mod } F_t$$

$$n_1,n_2 = 0,1,\ldots,N-1$$

....(4.5.2)

where Q is the multiplicational inverse(mod F_t) of N,i.e. $QN = 1 \text{ mod } F_t$

In (4.5.1), an interchange of the summation gives

$$x(k_1,k_2) = \left(\sum_{n_2=0}^{N-1} \sum_{n_1=0}^{N-1} x(n_1,n_2)\, a^{n_1k_1}\, a^{n_2k_2} \right) \text{mod } F_t$$

....(4.5.3)

If $x(n_1,n_2)$ is represented as a row-column array as does a matrix, then from (4.5.3), it can be seen that the 2-D FNT can be obtained by performing a series of 1-D FNT'S, first over the column and then over the rows. With the 64-point 1-D FNT algorithm developed, a 64x64 points 2-D FNT can be performed efficiently.

4-5-2 Two-dimensional Convolution via 2-D FNT

If $x(n_1,n_2)$ represents a 2-D sequence of duration L points on both dimensions and $h(n_1,n_2)$ represents another 2-D sequence of duration M points on both dimensions, $(x(n_1,n_2) = h(n_1,n_2) = 0$ for $n_1,n_2 \neq 0,1,\ldots,N-1)$, the linear 2-D convolution of x and h is

$$y(n_1,n_2) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(m_1,m_2)h(n_1-m_1,n_2-m_2),$$

$$N = L+M$$

$$n_1,n_2 = 0,1,2,\ldots N-1$$

Let both x and h be appended with zeros to the dimension of NxN, and $x(k_1,k_2)$, $H(k_1,k_2)$ represent the 2-D FNT'S of the appended sequences respectively. Then, the 2-D IFNT of the product,

$$Y(k_1,k_2) = (\ X(k_1,k_2)H(k_1,k_2)\ ) \bmod F_t \qquad k_1,k_2=0,1,\ldots N-1,$$

which is

$$y(n_1,n_2) = (QQ \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{N-1} Y(k_1,k_2)\, a^{-n_1k_1}\, a^{-n_2k_2}) \bmod F_t$$

$$n_1,n_2 = 0,1,\ldots,N-1 \qquad \ldots.(4.5.4)$$

is equivalent to the 2-D convolution of x and h.

The problem of wordlength constraint also exists in this application. The dynamic ranges could be set by the bound,

$$\left| x(n_1,n_2) \right|_{max} \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} \left| h(n_1,n_2) \right| \leq F_t/2, \quad \ldots(4.5.5)$$

If this dynamic range restriction is too severe, the wordlength segmentation described in section 4-2-2 could be applied to the filter array, $h(n_1,n_2)$, so that each word of $h(n_1,n_2)$ is partitioned into two blocks of appropriate numbers of bits, such as

$$h = h_1 2^k + h_2, \quad |h_2| < 2^k$$

The convolution is then computed as

$$y = x * h_1 2^k + x * h_2$$

The execution time required would be roughly 50% more of that required by (4.5.4).

The sectioned 2-D convolution technique [7] can also be applied to the processing of large pictures by the 2-D FNT.

4-6 Picture Processing

In [12] , Rader proposed that the FNT could be employed to implement picture filtering which involves, in most of the practical cases, two-dimensional FIR filters of size less than 20 × 20 samples.

This application of the FNT could be particularly rewarding for implementation with small computers, such as the NOVA-840 mini-computer. The short sequence length along each dimension of the filter array is one reason that makes the FNT technique for picture processing superior to the conventional method, such as the FFT's. The other reasons are:

1) The dynamic range of the digital image signal is not large. Four bits of intensity value for each pixel is usually acceptable. In fact, a 4-bit array and a 6-bit array of the same image, when normalized to 256 gray levels and displayed on the cathod ray tube, are difficult to be distinguished by telling from nake eyes. The FNT method is efficient for such type of processing.

2) The main memory size of the mini-computer limits the use of large two-dimensional array for transform. The efficiency of short sequence 2-D filtering using the conventional FFT method is not attractive. This is not true for the FNT method.

3) FNT computation employs integer arithmetics, which is simple and faster on the mini-computer than floating point arithmetics.

4-7 Examples of Picture Processing Application

To test the efficiency of the FNT Shift-Diminished Algorithm developed in this work, a main program in FORTRAN is written. This program makes use of the two-dimensional overlap-save technique to process the image array of size up to 256 x 256 samples. 24 k-words of the extended memory is assigned for temporary storage of the intermediate results during the computation. This is to reduce the frequency of the time consuming I/O operations between the disk files and CPU. The size of the filter array can be less than or equal to 33 x 33 points.

Fig. 4-7-1 depicts the 2-D sectioned convolution by overlap-save technique. The 256x256 points picture array is partition into 64 blocks of size 32x32 points each as shown in Fig.4-7-1 (a). Each lap of convolution processes one new block. At the beginning of each lap of processing, the new block of data is placed into the lower right quadrant of the 64x64 array to be transformed, and the other three quadrants are appended with the last three adjoining blocks. The array is forward transformed to give its 2-D FNT array which is then multiplied by $H(k_1,k_2)$ mod 65537, where $H(k_1,k_2)$ is the 2-D FNT of the impulse responseappended with zeros to the size of 64x64 points. The transform domain product is finally inverse transformed to give the convolved result. The 32x32 partition at the lower right quadrant of the resultant array is the desired result for that lap of convolution. The process is repeated for all the 64 blocks.

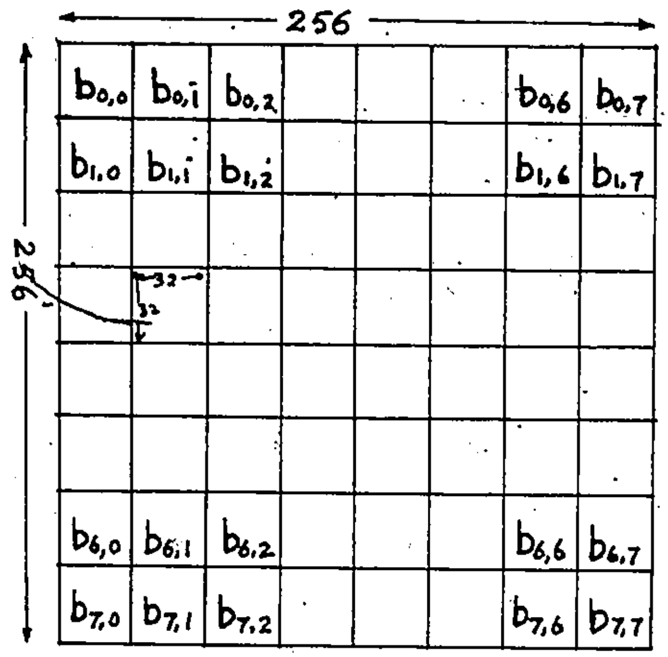

Fig.4-7-1 (a). Partition of 256x256 points picture into 64 blocks of size 32x32 points each for overlap-save convolutions.

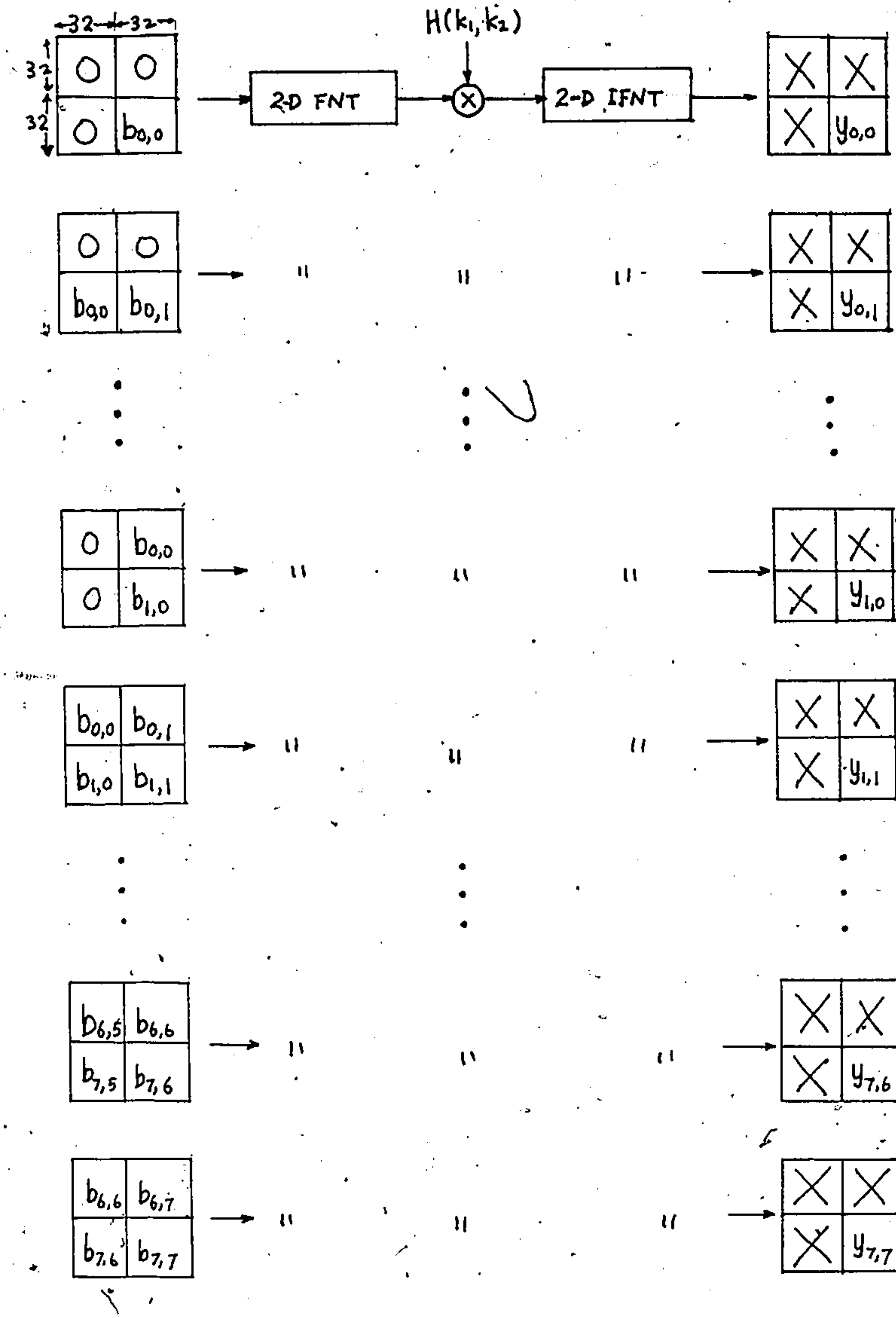

Fig. 4-7-1 (b). 2-D sectioned convolution (overlap-save method) via 2-D FNT. 'O' denotes zeros appended; 'X' denotes useless result.

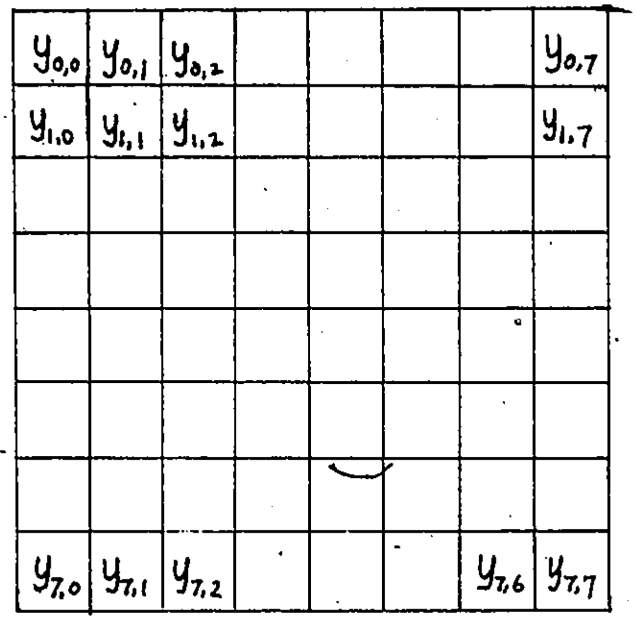

Fig. 4-7-1 (c). 256x256 points processed picture formed of 64 blocks of partial results from sectioned convolution by 2-D FNT technique.

4-7-1 Edge Enhancement

This example shows how the FNT is used for edge enhance enhancement of an image pattern.

The original image is shown in Fig.4-7-2. The image is of size 128x128 points with 8 bits representation for intensity at each point. It is scaled to 4 bits by truncation before processing.

The filter is the Laplacian [14], [15] whose magnitude specification is

$$H(w_{ij}) = w_{ij}^2$$

where $w_{ij} = (w_1^2 + w_2^2)^{\frac{1}{2}}$

and w_1, w_2 are frequency variables along the two dimensions. This specification is computer generated for the size of 32x32 points. Two-dimensional inverse DFT is performed over this array to obtain the impulse response. The impulse response is scaled up to 5 bits by multiplying each sample by 2^5. It is then appended with zeros to the size of 64x64 points, and its 2-D FNT, $H(k_1,k_2)$, is taken.

The filtering is performed via FNT as described previously. The execution time for each lap of convolution (processing one block of 32x32 points image partition) is recorded. The filtered image is shown in Fig.4-7-3.

The same image filtering using the FNT algorithm of Agarwal and Burrus [2], as well as using the FFT techniques have also been performed; and their execution times are recorded for reference.

Average execution time for each lap of convolution using 64x64-point transform:

By FFT	approximately	29 sec.
By FNT (algorithm of Agarwal and Burrus [2])	"	7.4 sec
By FNTSD	"	4.6 sec

Overall processing time for 128x128 points image - (includes approximately 20 sec. of I/O operation between the CPU and disk):

By FFT	approximately	484 sec.
By FNT (algorithm of Agarwal and Burrus [2])	"	138 sec.
By FNTSD	"	94 sec.

** Note: The FFT used corresponds to the fastest algorithm of its family, which transforms two blocks of real data at the same time.

FNTSD is the FNT Shift-Diminished algorithm developed in this work.

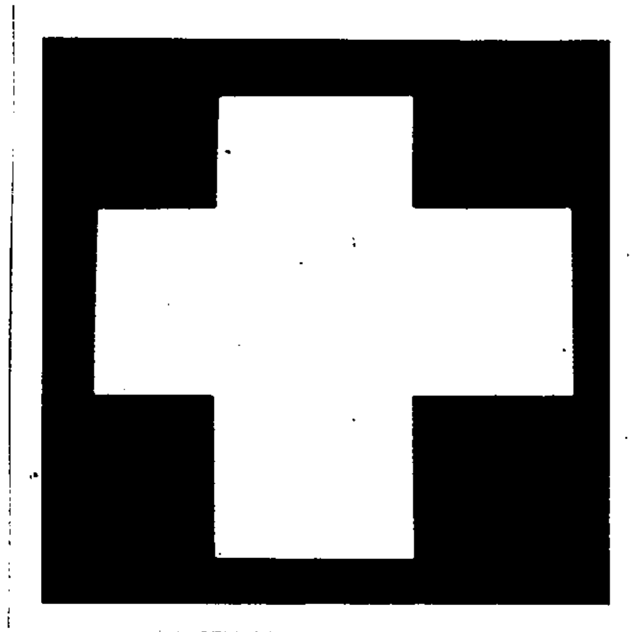

Fig.4-7-2 Original image

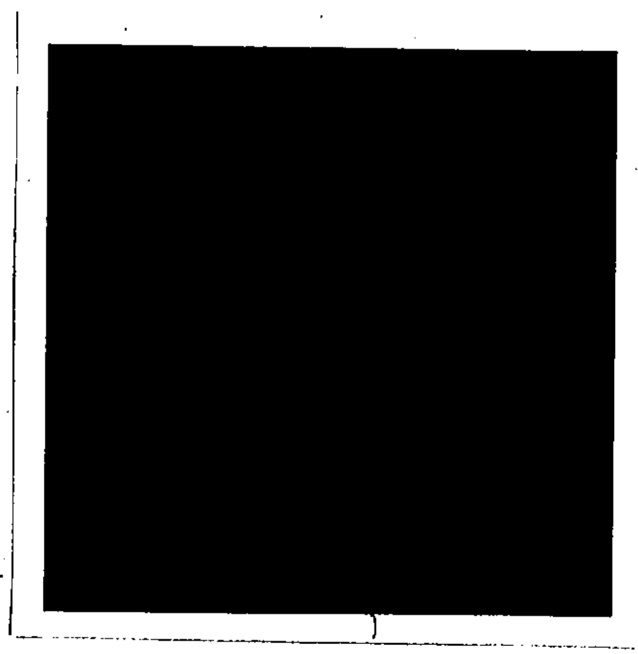

Fig.4-7-3 Edge enhanced Image

CHAPTER 5

SUMMARY AND CONCLUSION

The DFT has long been known to possess the property for computing convolution. Nevertheless the study and systematic description of the cyclic convolution property that the transform possesses is the matter of recent years. The FNT investigated in this work is one of the results of generalizing the cyclic convolution property to other class of transform in different number system. FNT has turned out to be a useful transform which can be utilized for computing convolution using computer of wordlength equals to a power of 2. A few research works have been done on the hardware realization of FNT to gain the full advantage of this transform without multiplication, 11 , 13 . In this work, the objective was to develop an efficient, error free algorithm which could be implemented with a general purpose computer to meet the requirement in general situation where the special purpose machine is not readily available.

An algorithm, the Shift-Diminished algorithm, is developed, and the efficiency is verified. This algorithm is particularly advantageous in speed for implementation with the computer which does not have high speed hardware for multiplication. The efficiency of this FNT algorithm is tested with examplesof actual application. It is more than 30% faster for convolution than the implementation using the algorithm of Agarwal and Burrus.

Table 5-1 lists the number of arithmetic and logical operations as well as the memory requirement for computing convolution via 64-point FNT Shift-Diminished algorithm and the algorithm of Agarwal and Burrus, 2, .

Table 5-2 lists the number of arithmetic and logical operations as well as the memory requirement for computing convolution via 64x64-point 2-D FNT using Shift-Diminished algorithm and the algorithm of Agarwal and Burrus.

Also entered in these two tables are the number of operations and memory requirement (floating point number) for convolution by the conventional FFT technique. This serves to provide the idea on the superiority of the FNT technique in terms of speed. It should be noted that the programs written for the FFT's are all in FORTRAN and that the variable and arrays are for floating point number operations. While the subprograms for the FNT evaluations are written in assembly language with arithmetics operated on integer numbers. An appropriate comparison between these two classes of transform should not be made by merely looking at the figures on the tables but should include the difference between FORTRAN and assembly languages as well as the difference between operations on floating and integer numbers.

	FFT	FNTAB	FNTSD
Multiplications	448	94	94
Additions	768	768	768
Shifts	0	904	389
Memory (words)	512	128	256

<u>Table 5-1</u> Number of arithmetic and logical operations and memory requirement for computing convolution using 64-point transforms.

Note: *FNTAB represents FNT with the algorithm of Agarwal and Burrus 2.

*FNTSD is the FNT Shift-Diminished algorithm developed in this work.

*The arithmetic operations under FFT are on complex number.

	FFT	FNTAB	FNTSD
Multiplications	57344	12032	12032
Additions	98304	98304	98304
Shifts	0	115712	49792
Memory (words)	32768	8192	16384

Table 5-2 Number of arithmetic, logical operations and memory requirements for computing 2-dimensional convolution using 64x64-point 2-D transforms.

Notes:

1. FNTAB is the FNT algorithm of Agarwal and Burrus.
2. FNTSD is the the FNT algorithm developed in this work.
3. The arithmetic operations under FFT are on complex number.

The features incorporated in FNT implementation with the Shift-Diminished algorithm developed in this work are summarized in the following.

1. Modification of the fast transform algorithm to reduce the powers of the basis function, $\sqrt{2}$ mod F_t, in the butterfly computations from the range of 0 to N/2-1 to the range of 0 to N/4, where N is the transform length. The "butterflies", which are the major computations in the fast transform algorithm, can thus be computed more efficiently as the multiplications of data by the basis function are performed by less bit shifting.

2. Incorporation of the Diminished-1 number coding [4] with an efficient scheme to avoid the error due to ambiguity in number representation.

The advantages of this FNT algorithm are

1. High speed, and
2. Error free computation.
3. Using integer arithmetics which is simple in programming and requir-ing less memory space.

The disadvantages are due to the nature of FNT, they are:

1. Limited dynamic range. The result of convolution via this approach should lie within the values, $-F_t/2$ and $F_t/2-1$, where F_t is the modulus of the FNT.
2. The transform length is fixed and restricted by the constraints of F_t, a and N.

A conclusion can be drawn with regard to the application of the FNT to digital signal processing. Knowing the advantages and disadvantages of the FNT technique, it can be concluded that the problems with the following characteristics could be benefit from the FNT approach.

1. Requiring convolution where the sequence lengths are short and the dynamic range is small.
2. Multiplication is costly.
3. Exact computation during convolution is needed.

Possible Area of Further Study

Futher expansion of the transform length and dynamic range could be possible by slicing a large mudulus into smaller moduli for seperate operations and combine the partial results.

Searching into other number fields may unveil other transforms with the advantages of FNT and less or none of the disadvantages.

APPENDIX

Included in the following pages are the compiled or assembled listing of the programs:

1. OSFNT: The mainline program written in FORTRAN-5 of the NOVA-840. This program performs image filtering via FNT technique (Shift-diminished algorithm developed in this thesis work). Overlap-save method is used for 2-D sectioned convolution.
2. Subroutine TDFNT: This subprogram is written in FORTRAN-5. It performs two-dimensional Fermat Number Transform by a sequence of 1-D FNT over the rows and columns of the data array.
3. MULMOD: This is the Assembly language subprogram which performs multiplication mod 65537 of two sequences of data in diminished-1 coding.
4. FNTSD: This subprogram is written in the assembly language of the NOVA-840. It performs Fermat Number Transform on a 64-point sequence using the Shift-diminished algorithm developed in this thesis:

```
C      FILENAME: OSFNT
C      THIS PROGRAM PERFORMS IMAGE FILTERING VIA 2D-FNT
C      TECHNIQUE. THE MAXIMUM SIZE OF FILTER(IMPULSE RESPONSE)
C      IS 33 X 33 POINTS.
C      THE MAXIMUM SIZE OF IMAGE IS 256 X 256 POINTS.
C      TWO DIMENSIONAL OVERLAP-SAVE METHOD IS EMPLOYED FOR
C      SECTIONED CONVOLUTION.
C
C      FILENAMES OF  SUBROUTINE REQUIRED:- TDFNT,FNTSD,MULMOD
C
       INTEGER IWIND(8192),XI(32,256),XC(64,64),XT(64,64)
       INTEGER H(64,64),HC(64,64),XT1(64),H1(64),XC1(64),HC1(64
       DIMENSION R(64),NAME(5),ITIME(3)
       COMMON/WINDOW/IWIND
       EQUIVALENCE(IWIND(4097),XT(1))
       EQUIVALENCE(IWIND,XI,XC,R)
       COMMON/TEMP/ XT1,XC1
       DATA NAME/5*'  '/,NT/64/,NX/32/,NBLK/8/
   100 FORMAT(' ',1X,'FILE NAME OF T.F.(IMPULSE RESP.): ',Z)
   200 FORMAT(5A2)
     1 WRITE(10,100)
       READ(11,200) NAME
       ACCEPT"SIZE OF T.F.=# COLUMN=#ROWS=",NH
       ACCEPT"INPUT TYPE=(INTEGER:0, REAL:1) ",ITY
       ACCEPT"MULTIPLY FACTOR=",AH
       I=2
       IF(ITY.EQ.1) I=4
       OPEN 0,NAME, LEN=I*NH, REC=NH
       DO 10 I=1,NT
       DO 10 J=1,NT
    10 H(I,J)=0
       DO 11 I=1,NH
       IF(ITY.EQ.0) READ(0) (H(I,J),J=1,NH)
       IF(ITY.EQ.0) GO TO 11
       IF(ITY.EQ.1) READ(0) (R(J),J=1,NH)
       DO 11 J=1,NH
       H(I,J)=IFIX(R(J)*AH+0.5)
    11 CONTINUE
       CALL TDFNT(H,NT,0,HC)
   110 FORMAT(' ',1X,'FILE NAME OF IMAGE: ',Z)
       WRITE(10,110)
       READ(11,200) NAME
       TYPE"SIZE OF IMAGE:"
       ACCEPT"# OF COLUMNS=",NJ
       ACCEPT"# OF ROWS=  ",NI
       ACCEPT"WISH TO SCALE IMAGE MAG.? YES(1),NO(0):",IYES
       IF(IYES.EQ.1) ACCEPT"SCALE FACTOR= ",AH
       OPEN 1, NAME, LEN=2*NJ, REC=NI
       TYPE"WISH TO STORE PROCESSED IMAGE IN"
       ACCEPT"ORIGINAL(IO=0) OR NEW(IO=1) FILE, IO=",IO
       IF(IO.EQ.0) GO TO 19
       IO=2
   120 FORMAT(' ',1X,'OUTPUT FILE NAME: ',Z)
       WRITE(10,120)
       READ(11,200) NAME
       OPEN 2, NAME, LEN=2*NJ, REC=NI
    19 IF(IO.EQ.0) IO=1
       CALL VMEM(K,IER)
```

```
      IF(IER.GE.5) GO TO 1000
      TYPE"EXT. MEMORY AVAILABLE (1024-WORD BLOCKS): ",K
      CALL MAPDF(K,IWIND,8,IER)
      IF(IER.GE.5) GO TO 1001
      NXBI=NI/NX
      NXBJ=NJ/NX
      NOMAP=3*NXBJ
      DO 90 I=1,NXBI
      IB=0
      IF((I.GT.1).AND.(MOD(I,2).EQ.0)) IB=NXBJ
      CALL REMAP(0,IB,NXBJ,IER)
      IER1=19
      IF(IER.GE.5) GO TO 1002
      IF(IO.EQ.2) GO TO 21
      IREC=(I-1)*NX+1
      CALL FSEEK(1,IREC)
   21 DO 20 L=1,NX
      READ(1) (XI(L,K),K=1,NJ)
      IF(IYES.NE.1) GO TO 20
      DO 20 K=1,NJ
      XI(L,K)=XI(L,K)*AH
   20 CONTINUE
      IER1=20
      CALL REMAP(0,NOMAP,NBLK,IER)
      IF(IER.GE.5) GO TO 1002
      DO 30 L=1,NT
      DO 30 K=1,NX
   30 XT(L,K)=0
      DO 60 J=1,NXBJ
      IF(I.GT.1) GO TO 35
      IF(J.GT.1) GO TO 32
      IER1=31
      CALL REMAP(0,IB,1,IER)
      IF(IER.GE.5) GO TO 1002
      DO 31 L=1,NX
      DO 31 K=1,NX
      XT(L,NX+K)=0
   31 XT(NX+L,NX+K)=XI(L,K)
      GO TO 39
   32 IER1=32
      JB=J-2
      CALL REMAP(0,JB,2,IER)
      IF(IER.GE.5) GO TO 1002
      DO 33 L=1,NX
      DO 33 K=1,NT
      XT(L,K)=0
   33 XT(NX+L,K)=XI(L,K)
      GO TO 39
   35 IF(J.GT.1) GO TO 37
      IER1=35
      JB=NXBJ-IB
      CALL REMAP(0,JB,1,IER)
      IF(IER.GE.5) GO TO 1002
      DO 360 L=1,NX
      DO 360 K=1,NX
  360 XT(L,NX+K)=XI(L,K)
      JB=IB
```

```
      CALL REMAP(0,JB,1,IER)
      DO 361 L=1,NX
      DO 361 K=1,NX
  361 XT(NX+L,NX+K)=XI(L,K)
      GO TO 39
   37 JB=NXBJ-IB+J-2
      CALL REMAP(0,JB,2,IER)
      IER1=37
      IF(IER.GE.5) GO TO 1002
      DO 370 L=1,NX
      DO 370 K=1,NT
  370 XT(L,K)=XI(L,K)
      JB=IB+J-2
      CALL REMAP(0,JB,2,IER)
      IER1=370
      IF(IER.GE.5) GO TO 1002
      DO 371 L=1,NX
      DO 371 K=1,NT
  371 XT(NX+L,K)=XI(L,K)
   39 CONTINUE
      IER1=39
      CALL REMAP(0,NOMAP,NBLK,IER)
      IF(IER.GE.5) GO TO 1002
      TYPE" "
  130 FORMAT(' ',1X,'PROCESSING BLOCK:',8X,'(',I3,' ,',I
      WRITE(10,130) I,J
      CALL TIME(ITIME,IER)
      TYPE"CONV BEGIN, TIME ->",ITIME(1),ITIME(2),ITIME(
      CALL TDFNT(XT,NT,1,XC)
      DO 41 L=1,NT
      DO 40 K=1,NT
      HC1(K)=HC(L,K)
      XC1(K)=XC(L,K)
      H1(K)=H(L,K)
   40 XT1(K)=XT(L,K)
      CALL MULMOD(XT1,H1,NT,XC1,HC1)
      DO 41 K=1,NT
      XC(L,K)=XC1(K)
   41 XT(L,K)=XT1(K)
      CALL TDFNT(XT,NT,-1,XC)
      CALL TIME(ITIME,IER)
      TYPE" "
      TYPE"CONV END, TIME   ->",ITIME(1),ITIME(2),ITIME(
      JB=J-1+NXBJ*2
      IER1=50
      CALL REMAP(0,JB,1,IER)
      IF(IER.GE.5) GO TO 1002
      DO 50 L=1,NX
      DO 50 K=1,NX
   50 XI(L,K)=XT(NX+L,NX+K)
   60 CONTINUE
      JB=2*NXBJ
      CALL REMAP(0,JB,NXBJ,IER)
      IER1=60
      IF(IER.GE.5) GO TO 1002
      IF(IO.EQ.1) CALL FSEEK(1,IREC)
      DO 70 L=1,NX
```

```
   70 WRITE(IO) (XI(L,K),K=1,NJ)
   90 CONTINUE
      IF(IO.EQ.2) CLOSE 2
      CLOSE 1
      CLOSE 0
      ACCEPT"WISH TO CONTINUE? NO:0, YES:1 ",K
      IF(K.EQ.1) GO TO 1
      GO TO 1004
 1000 TYPE"ERROR IN VMEM  -->ERROR # =",IER
      GO TO 1003
 1001 TYPE"ERROR IN MAPDF -->ERROR # =",IER
      GO TO 1003
 1002 TYPE"ERROR IN REMAP -->ERROR # =",IER
      TYPE"ERROR NEAR STATEMENT # ",IER1
 1003 IF(IO.EQ.2) CLOSE 2
      IF(IO.EQ.2) DELETE NAME
      CLOSE 1
      CLOSE 0
 1004 STOP
      END
```

TDFNT

```
      SUBROUTINE TDFNT(XT,ND,IFNT,XC)
C     THIS ROUTINE PERFORMS TWO-DIMENSIONAL
C     FERMAT NUMBER TRANSFORM (2-D FNT).
C
C     XT CONTAINS 2-DIMENSIONAL DATA ARRAY
C     TO BE TRANSFORMED. THE TRANSFORMED DATA
C     WILL BE RETURNED VIA THIS SAME ARRAY.
C     ND=64
C     IFNT=0 FOR FORWARD 2-D FNT,
C     IFNT=1 FOR INVERSE 2-D FNT;
C     XC HOLDS THE 17TH BIT OF CORRESPONDING
C     DIMINISHED-1 CODED DATA.
      INTEGER XT(64,64),XC(64,64),XT1(64),XC1(64)
      COMMON/TEMP/ XT1,XC1
      LCODX=1
      KCODX=0
      IF(IFNT.LT.0) LCODX=0
      IF(IFNT.LT.0) KCODX=1
      DO 2 L=1,ND
      DO 1 K=1,ND
      XC1(K)=XC(L,K)
    1 XT1(K)=XT(L,K)
      CALL FNT64(XT1,ND,IFNT,LCODX,XC1)
      DO 2 K=1,ND
      XC(L,K)=XC1(K)
    2 XT(L,K)=XT1(K)
      DO 4 K=1,ND
      DO 3 L=1,ND
      XC1(L)=XC(L,K)
    3 XT1(L)=XT(L,K)
      CALL FNT64(XT1,ND,IFNT,KCODX,XC1)
      DO 4 L=1,ND
      XC(L,K)=XC1(L)
    4 XT(L,K)=XT1(L)
      RETURN
      END
```

```
0001 MULMO MACRO REV 06.00                         13:08:27 07/15/79
                         ;          FILENAME: MULMOD
                         ;          THIS ROUTINE PERFORMS DIMINISHED-1
                         ;          MULTIPLICATION MOD 65537 OF TWO
                         ;          64-POINT DATA SEQUENCES.
                         ;          CALLING STATEMENT FROM FORTRAN ROUTINE
                         ;          CALL MULMOD(IX, IH, N, IXC, IHC)
                         ;          WHERE
                         ;          IX AND IH ARE BOTH 64-ELEMENT 1-D ARRA
                         ;          CONTAINING 2 DIMINISHED-1 CODED DATA SE
                         ;          RESPECTIVELY;
                         ;          N=64;
                         ;          IXC AND IHC ARE BOTH 64-ELEMENT 1-D ARR
                         ;          CONTAINING THE 17TH BIT OF THE CORRESPO
                         ;          DATA POINTS IN IX AND IH RESPECTIVELY.
                         ;

                                    .TITL   MULMOD
                                    .ENT    MULMOD
                                    .EXTU
                                    .ZREL
   00000-000000'MULMOD:             .MLD
                                    .NREL
                        .MLD:       SAVE    0
                                       STA     3,.ND1
                                       JSR     @.SAV2
                                       0               ;STACK EXTENSION
                                       -               ;OFFSET TO MODEL STACK
   00002'023773                     LDA     0,@-5,3
   00003'040456                     STA     0,K
   00004'126520                     SUBZL   1,1
   00005'021775                     LDA     0,-3,3
   00006'122400                     SUB     1,0
   00007'040453                     STA     0,X
   00010'021774                     LDA     0,-4,3
   00011'122400                     SUB     1,0
   00012'040445                     STA     0,H
   00013'021772                     LDA     0,-6,3
   00014'122400                     SUB     1,0
   00015'040446                     STA     0,XC
   00016'021771                     LDA     0,-7,3
   00017'122400                     SUB     1,0
   00020'040440                     STA     0,HC
                                    ;
   00021'010442 ML1:                ISZ     XC
   00022'010440                     ISZ     X
   00023'010435                     ISZ     HC
   00024'010433                     ISZ     H
   00025'022436                     LDA     0,@XC
   00026'101014                     MOV#    0,0,SZR
   00027'000425                     JMP     ML3
   00030'022430                     LDA     0,@HC
   00031'101005                     MOV     0,0,SNR
   00032'000403                     JMP     .+3
   00033'126400                     SUB     1,1
   00034'000416                     JMP     ML2
```

```
00035'176400          SUB     3,3
00036'026424          LDA     1,@X
00037'032420          LDA     2,@H
00040'141000          MOV     2,0
00041'123022          ADDZ    1,0,SZC
```

```
 0002 MULMO
00042'000403          JMP     .+3
00043'101405          INC     0,0,SNR
00044'101400          INC     0,0
00045'073301          MUL
00046'106023          ADCZ    0,1,SNC
00047'125404          INC     1,1,SZR
00050'000403          JMP     .+3
00051'102520          SUBZL   0,0
00052'042411 ML2:     STA     0,@XC
00053'046407          STA     1,@X
00054'014405 ML3:     DSZ     K
00055'000744          JMP     ML1
                      RTN
00057'000000 H:       0
00060'000000 HC:      0
00061'000000 K:       0
00062'000000 X:       0
00063'000000 XC:      0
                      .END

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

```
                          ;        FILENAME: FNTSD
                          ;        THIS ROUTINE PERFORMS 64-POINT
                          ;        FERMAT NUMBER TRANSFORM (FNT) USING
                          ;        SHIFT-DIMINISHE ALGORITHM.
                          ;        CALLING STATEMENT FROM FORTRAN ROUTINE:
                          ;        CALL FNT64(IX,N,IFT,IC,IXC)
                          ;        WHERE
                          ;        IX IS A 64-POINT 1-D ARRAY WHICH CONTAINS
                          ;        DATA POINTS TO BE TRANSFORM;
                          ;        N=64
                          ;        IFT IS A CONTROL VARIABLE,
                          ;        IFT=0 FOR FORWARD FNT,
                          ;        IFT=1 FOR INVERSE FNT
                          ;        IC IS ANOTHER CONTROL VARIABLE,
                          ;        IC=1 IF CODE TRANSLATION REQUIRED,
                          ;        IC=0 IF NO CODE TRANSLATION REQUIRED;
                          ;        IXC IS A 64-POINT 1-D ARRAY WHICH
                          ;        CONTAINS THE 17TH BIT OF THE DIMINISHED-1 COD
                          ;        DATA.
                          ;
                                   .TITL    FNT64
                                   .ENT     FNT64
                                   .EXTU
                                   .ZREL
   00000-000060'FNT64:             .DFDT

                          ;        MACRO DEFINITION FOR SHIFT AND
                          ;        MODULO REDUCTION
                                   .MACRO   SHFT
                          **       .IFE ^1-0
                          **       .DO ^2
                                   MOVZL    1,1,SNC
                                   INC      ^1,1
                          **       .ENDC
                          **       .ENDC
                          **       .IFE     ^1-1
                                   SUBZR    0,0
                          **       .DO      ^2
                                   MOVZR    1,1,SNC
                                   ADD      ^0,1
                          **       .ENDC
                          **       .ENDC
                                   JMP      @RT2
                                   %

                          ;        MACRO DEFINITION FOR
                          ;        PERFORMING MULTIPLICATION OF
                          ;        DATA POINTS BY 2**8 MOD 65537
                                   .MACRO   SWAP
                                   COM      1,0
                                   ANDS     3,0
                                   COM,     3,2
                                   ANDS     2,1
                                   ADD      0,1
                                   %
   00001-000665'TWD1. :            .TWD1
   00002-000715'D1TW. :            .D1TW
   00003-000040'FFNT:              .DIF
   00004-000446'IFNT:              .DIT
   00005-000617'DVN. :             .DVN
```

```
                            .NREL
                    .DFDT:  SAVE     0
                             STA      3,.ND1
                             JSR      @.SAV2
                             0                      ;STACK EXTENSION
                             -.                     ;OFFSET TO MODEL STACK
00002'023774                LDA      0,@ARG1,3
00003'040017-               STA      0,N
00004'030045-               LDA      2,MNS6
00005'050020-               STA      2,NL
00006'025775                LDA      1,ARG0,3
00007'124400                NEG      1,1
00010'124000                COM      1,1
00011'044035-               STA      1,X0
00012'107000                ADD      0,1
00013'044036-               STA      1,XN
00014'023772                LDA      0,@-6,3
00015'040046-               STA      0,CODX
00016'031771                LDA      2,-7,3
00017'050037-               STA      2,XC
                            ;;
                            ;IFT=-1  :FOR IFNT WITHOUT X (1/N)
                            ;IFT=1   :FOR FFNT  "  "       "
                            ;IFT=0   :FOR FFNT FOLLOWED BY X (1/N)
00020'027773                LDA      1,@ARG2,3
00021'044041-               STA      1,IFT
00022'125234                MOVZR#   1,1,SZR
00023'000410                JMP      IVSTR
00024'101004                MOV      0,0,SZR
00025'006001-               JSR      @TWD1.
00026'006003-               JSR      @FFNT
00027'024041-               LDA      1,IFT
00030'125005                MOV      1,1,SNR
00031'006005-               JSR      @DVN.
                            RTN
00033'006004-IVSTR:         JSR      @IFNT
00034'020046-               LDA      0,CODX
00035'101004                MOV      0,0,SZR
00036'006002-               JSR      @D1TW.
                            RTN
                    ;
                    ;       SUBROUTINE .DIF
                    ;       TO PERFORM FORWARD FNT
                    ;
00040'054015-.DIF:          STA      3,AC3
00041'034043-               LDA      3,HBYT
00042'102400                SUB      0,0
00043'040006-               STA      0,RT1
00044'024010-               LDA      1,RT3
00045'044007-               STA      1,RT2
00046'024012-               LDA      1,SL0
00047'044016-               STA      1,AB3
00050'030017-               LDA      2,N
00051'050032-               STA      2,LE1
00052'126520                SUBZL    1,1
00053'044030-               STA      1,K
00054'030032-.DOL1:         LDA      2,LE1
00055'050031-               STA      2,LE
00056'151220                MOVZR    2,2
00057'050032-               STA      2,LE1
```

```
00060'050021-            STA      2,NJ
00061'102400             SUB      0,0
00062'040033-            STA      0,PW
00063'020012-            LDA      0,SL0
00064'125235             MOVZR#   1,1,SNR
00065'020010-            LDA      0,RT3
00066'040034-            STA      0,APW
00067'024035-            LDA      1,X0
00070'125400             INC      1,1
00071'044026-            STA      1,J
00072'030037-            LDA      2,XC
00073'050027-            STA      2,JC
00074'024026-.DOJ1:      LDA      1,J
00075'030027-            LDA      2,JC
00076'044024-.DOI1:      STA      1,I
00077'050022-            STA      2,IC
00100'020032-            LDA      0,LE1
00101'107000             ADD      0,1
00102'044025-            STA      1,IP
00103'143000             ADD      2,0
00104'040023-            STA      0,IPC
00105'025000             LDA      1,0,2
00106'125005             MOV      1,1,SNR
00107'000414             JMP      .A0
00110'026023-            LDA      1,@IPC
00111'125005             MOV      1,1,SNR
00112'000404             JMP      .+4
00113'102400             SUB      0,0
00114'040006-            STA      0,RT1
00115'000437             JMP      .LPI1
00116'045000             STA      1,0,2
00117'026025-            LDA      1,@IP
00120'046024-            STA      1,@I
00121'124000             COM      1,1
00122'002034-            JMP      @APW
00123'022023-.A0:        LDA      0,@IPC
00124'101005             MOV      0,0,SNR
00125'000434             JMP      .A1
00126'046023-            STA      1,@IPC
00127'026024-            LDA      1,@I
00130'002034-            JMP      @APW
00131'022024-.A1:        LDA      0,@I
00132'032025-            LDA      2,@IP
00133'144000             COM      2,1
00134'113022             ADDZ     0,2,SZC
00135'000406             JMP      .A2
00136'151404             INC      2,2,SZR
00137'000404             JMP      .A2
00140'152520             SUBZL    2,2
00141'052022-            STA      2,@IC
00142'152400             SUB      2,2
00143'052024-.A2:        STA      2,@I
00144'107022             ADDZ     0,1,SZC
00145'002034-            JMP      @APW
00146'125404             INC      1,1,SZR
00147'002034-            JMP      @APW
00150'044006-            STA      1,RT1
00151'152520             SUBZL    2,2
00152'052023-            STA      2,@IPC
00153'046025-.A3:        STA      1,@IP
```

```
00154'024024- LPI1:    LDA      1,I
00155'020031-          LDA      0,LE
00156'030022-          LDA      2,IC
00157'107000           ADD      0,1
00160'113000           ADD      0,2
00161'020036-          LDA      0,XN
00162'122513           SUBL#    1,0,SNC
00163'000713           JMP      .DOI1
00164'030030-          LDA      2,K
00165'020033-          LDA      0,PW
00166'143020           ADDZ     2,0
00167'040033-          STA      0,PW
00170'101222           MOVZR    0,0,SZC
00171'000421           JMP      .RT10
00172'030011-          LDA      2,SL
00173'113000           ADD      0,2
00174'025000           LDA  1,0,2
00175'044034-          STA      1,APW
00176'010026- IXJ1:    ISZ      J
00177'010027-          ISZ      JC
00200'014021-          DSZ      NJ
00201'000673           JMP      .DOJ1
00202'020012-          LDA      0,SL0
00203'040007-          STA      0,RT2
00204'024030-          LDA      1,K
00205'125120           MOVZL    1,1
00206'044030-          STA      1,K
00207'010020-          ISZ      NL
00210'000644           JMP      .DOL1
00211'002015-          JMP      @AC3
00212'126520  RT10:    SUBZL    1,1
00213'044006-          STA      1,RT1
00214'000762           JMP      .IXJ1
                       ;
              ;        TO PERFORM DIMINISHED-1 MULTIPLICATION
              ;        OF DATA POINT BY 4080 (=2**1/2 MOD 65537)
              ;
00215'014006- RT3:     DSZ      RT1
00216'002016-          JMP      @AB3
00217'121123           MOVZL    1,0,SNC
00220'101400           INC      0,0
00221'101123           MOVZL    0,0,SNC
00222'101400           INC      0,0
00223'101123           MOVZL    0,0,SNC
00224'101400           INC      0,0
00225'101123           MOVZL    0,0,SNC
00226'101400           INC      0,0
00227'152620           SUBZR    2,2
00230'124223           COMZR    1,1,SNC
00231'147000           ADD      2,1
00232'125223           MOVZR    1,1,SNC
00233'147000           ADD      2,1
00234'125223           MOVZR    1,1,SNC
00235'147000           ADD      2,1
00236'125223           MOVZR    1,1,SNC
00237'147000           ADD      2,1
00240'106023           ADCZ     0,1,SNC
00241'125400           INC      1,1
00242'002016-          JMP      @AB3
                       ;
```

```
                      ;        ROUTINE MODULES FOR
                      ;        PERFORMING 'MULTIPLICATION' OF
                      ;        DATA POINTS BY DIFFERENT POWERS OF 2
                      ;
00243'124000   .SR16:  COM     1,1
00244'002007-          JMP     @RT2
00245'124000   .SR15:  COM     1,1
               .SL1:   SHFT    0,1
00246'125123           MOVZL   1,1,SNC
00247'125400           INC     1,1
                       SUBZR   0,0
                       MOVZR   1,1,SNC
                       ADD     0,1
00250'002007-          JMP     @RT2
00251'124000           .SR14:: COM       1,1
               .SL2:   SHFT    0,2
00252'125123           MOVZL   1,1,SNC
00253'125400           INC     1,1
00254'125123           MOVZL   1,1,SNC
00255'125400           INC     1,1
                       SUBZR   0,0
                       MOVZR   1,1,SNC
                       ADD     0,1
00256'002007-          JMP     @RT2
00257'124000           .SR13:  COM       1,1
               .SL3:   SHFT    0,3
00260'125123           MOVZL   1,1,SNC
00261'125400           INC     1,1
00262'125123           MOVZL   1,1,SNC
00263'125400           INC     1,1
00264'125123           MOVZL   1,1,SNC
00265'125400           INC     1,1
                       SUBZR   0,0
                       MOVZR   1,1,SNC
                       ADD     0,1
00266'002007-          JMP     @RT2
00267'124000           .SR12:  COM       1,1
               .SL4:   SHFT    0,4
00270'125123           MOVZL   1,1,SNC
00271'125400           INC     1,1
00272'125123           MOVZL   1,1,SNC
00273'125400           INC     1,1
00274'125123           MOVZL   1,1,SNC
00275'125400           INC     1,1
00276'125123           MOVZL   1,1,SNC
00277'125400           INC     1,1
                       SUBZR   0,0
                       MOVZR   1,1,SNC
                       ADD     0,1
00300'002007-          JMP     @RT2
00301'124000           .SR11:  COM       1,1
               .SL5:   SHFT    0,5
00302'125123           MOVZL   1,1,SNC
00303'125400           INC     1,1
00304'125123           MOVZL   1,1,SNC
00305'125400           INC     1,1
00306'125123           MOVZL   1,1,SNC
00307'125400           INC     1,1
00310'125123           MOVZL   1,1,SNC
00311'125400           INC     1,1
```

```
0006 FNT64
   00312'125123          MOVZL     1,1,SNC
   00313'125400          INC       1,1
                         SUBZR     0,0
                         MOVZR     1,1,SNC
                         ADD       0,1
   00314'002007-         JMP       @RT2
                         .SL6:     SWAP
   00315'120000          COM       1,0
   00316'163700          ANDS      3,0
   00317'170000          COM       3,2
   00320'147700          ANDS      2,1
   00321'107000          ADD       0,1
                                   SHFT      1,2
                         MOVZL     1,1,SNC
                         INC       1,1
   00322'102620          SUBZR     0,0
   00323'125223          MOVZR     1,1,SNC
   00324'107000          ADD       0,1
   00325'125223          MOVZR     1,1,SNC
   00326'107000          ADD       0,1
   00327'002007-         JMP       @RT2
                         .SL7:     SWAP
   00330'120000          COM.      1,0
   00331'163700          ANDS      3,0
   00332'170000          COM       3,2
   00333'147700          ANDS      2,1
   00334'107000          ADD       0,1
                                   SHFT      1,1
                         MOVZL     1,1,SNC
                         INC       1,1
   00335'102620          SUBZR     0,0
   00336'125223          MOVZR     1,1,SNC
   00337'107000          ADD       0,1
   00340'002007-         JMP       @RT2
                         .SL8:     SWAP
   00341'120000          COM       1,0
   00342'163700          ANDS      3,0
   00343'170000          COM       3,2
   00344'147700          ANDS      2,1
   00345'107000          ADD       0,1
   00346'002007-                   JMP       @RT2
                 .SL9:   SWAP
   00347'120000          COM       1,0
   00350'163700          ANDS      3,0
   00351'170000          COM       3,2
   00352'147700          ANDS      2,1
   00353'107000          ADD       0,1
                                   SHFT      0,1
   00354'125123          MOVZL     1,1,SNC
   00355'125400          INC       1,1
                         SUBZR     0,0
                         MOVZR     1,1,SNC
                         ADD       0,1
   00356'002007-         JMP       @RT2
                         .SL10:    SWAP
   00357'120000          COM       1,0
   00360'163700          ANDS      3,0
   00361'170000          COM       3,2
   00362'147700          ANDS      2,1
   00363'107000          ADD       0,1
```

```
                                SHFT    0,2
00364'125123                    MOVZL   1,1,SNC
00365'125400                    INC     1,1
00366'125123                    MOVZL   1,1,SNC
00367'125400                    INC     1,1
                                SUBZR   0,0
                                MOVZR   1,1,SNC
                                ADD     0,1
00370'002007-                   JMP     @RT2
00371'124000            .SL11:  COM     1,1
                 .SR5:          SHFT    1,5
                                MOVZL   1,1,SNC
                                INC     1,1
00372'102620                    SUBZR   0,0
00373'125223                    MOVZR   1,1,SNC
00374'107000                    ADD     0,1
00375'125223                    MOVZR   1,1,SNC
00376'107000                    ADD     0,1
00377'125223                    MOVZR   1,1,SNC
00400'107000                    ADD     0,1
00401'125223                    MOVZR   1,1,SNC
00402'107000                    ADD     0,1
00403'125223                    MOVZR   1,1,SNC
00404'107000                    ADD     0,1
00405'002007-                   JMP     @RT2
00406'124000            .SL12:  COM     1,1
                 .SR4:          SHFT    1,4
                                MOVZL   1,1,SNC
                                INC     1,1
00407'102620                    SUBZR   0,0
00410'125223                    MOVZR   1,1,SNC
00411'107000                    ADD     0,1
00412'125223                    MOVZR   1,1,SNC
00413'107000                    ADD     0,1
00414'125223                    MOVZR   1,1,SNC
00415'107000                    ADD     0,1
00416'125223                    MOVZR   1,1,SNC
00417'107000                    ADD     0,1
00420'002007-                   JMP     @RT2
00421'124000            .SL13:  COM     1,1
                 .SR3:          SHFT    1,3
                                MOVZL   1,1,SNC
                                INC     1,1
00422'102620                    SUBZR   0,0
00423'125223                    MOVZR   1,1,SNC
00424'107000                    ADD     0,1
00425'125223                    MOVZR   1,1,SNC
00426'107000                    ADD     0,1
00427'125223                    MOVZR   1,1,SNC
00430'107000                    ADD     0,1
00431'002007-                   JMP     @RT2
00432'124000            .SL14:  COM     1,1
                 .SR2:          SHFT    1,2
                                MOVZL   1,1,SNC
                                INC     1,1
00433'102620                    SUBZR   0,0
00434'125223                    MOVZR   1,1,SNC
00435'107000                    ADD     0,1
00436'125223                    MOVZR   1,1,SNC
00437'107000                    ADD     0,1
```

```
00440'002007-              JMP     @RT2
00441'124000              .SL15:   COM     1,1
                   .SR1:  SHFT    1,1
                          MOVZL   1,1,SNC
                          INC     1,1
00442'102620              SUBZR   0,0
00443'125223              MOVZR   1,1,SNC
00444'107000              ADD     0,1
00445'002007-             JMP     @RT2
                          ;
                   ;      SUBROUTINE .DIT
                   ;      TO PERFORM INVERSE FNT
00446'054015-.DIT:        STA     3,AC3
00447'034044-             LDA     3,LBYT
00450'102400              SUB     0,0
00451'040006-             STA     0,RT1
00452'024014-             LDA     1,SR0
00453'044007-             STA     1,RT2
00454'044016-             STA     1,AB3
00455'126520              SUBZL   1,1
00456'044031-             STA     1,LE
00457'020042-             LDA     0,NB
00460'040030-             STA     0,K
00461'030031-.DOL2:       LDA     2,LE
00462'050032-             STA     2,LE1
00463'050021-             STA     2,NJ
00464'151120              MOVZL   2,2
00465'050031-             STA     2,LE
00466'102400              SUB     0,0
00467'040033-             STA     0,PW
00470'020014-             LDA     0,SR0
00471'040034-             STA     0,APW
00472'024035-             LDA     1,X0
00473'125400              INC     1,1
00474'044026-             STA     1,J
00475'030037-             LDA     2,XC
00476'050027-             STA     2,JC
00477'024026-.DOJ2:       LDA     1,J
00500'030027-             LDA     2,JC
00501'044024-.DOI2:       STA     1,I
00502'050022-             STA     2,IC
00503'020032-             LDA     0,LE1
00504'107000              ADD     0,1
00505'044025-             STA     1,IP
00506'113000              ADD     0,2
00507'050023-             STA     2,IPC
00510'021000              LDA     0,0,2
00511'026025-             LDA     1,@IP
00512'101005              MOV     0,0,SNR
00513'002034-             JMP     @APW
00514'102400              SUB     0,0
00515'040006-             STA     0,RT1
00516'026022-             LDA     1,@IC
00517'125004              MOV     1,1,SZR
00520'000435              JMP     .LPI2
00521'045000              STA     1,0,2
00522'032024-             LDA     2,@I
00523'000431              JMP     .B3
00524'022022-.B0:         LDA     0,@IC
00525'101005              MOV     0,0,SNR
```

0009 FNT64

```
00526'000406            JMP     .B1
00527'102400            SUB     0,0
00530'042022-           STA     0,@IC
00531'046024-           STA     1,@I
00532'130000            COM     1,2
00533'000421            JMP     .B3
00534'032024-.B1:       LDA     2,@I
00535'120000            COM     1,0
00536'147022            ADDZ    2,1,SZC
00537'000406            JMP     .B2
00540'125404            INC     1,1,SZR
00541'000404            JMP     .B2
00542'126520            SUBZL   1,1
00543'046022-           STA     1,@IC
00544'126400            SUB     1,1
00545'046024-.B2:       STA     1,@I
00546'113022            ADDZ    0,2,SZC
00547'000405            JMP     .B3
00550'151404            INC     2,2,SZR
00551'000403            JMP     .B3
00552'102520            SUBZL   0,0
00553'042023-           STA     0,@IPC
00554'052025-.B3:       STA     2,@IP
00555'024024-.LPI2:     LDA     1,I
00556'020031-           LDA     0,LE
00557'030022-           LDA     2,IC
00560'107000            ADD     0,1
00561'113000            ADD     0,2
00562'020036-           LDA     0,XN
00563'122513            SUBL#   1,0,SNC
00564'000715            JMP     .DOI2
00565'030030-           LDA     2,K
00566'020033-           LDA     0,PW
00567'143000            ADD     2,0
00570'040033-           STA     0,PW
00571'101222            MOVZR   0,0,SZC
00572'000463            JMP     .RT20
00573'030013-           LDA     2,SR
00574'113000            ADD     0,2
00575'025000            LDA   1,0,2
00576'044034-           STA     1,APW
00577'010026-.IXJ2:     ISZ     J
00600'010027-           ISZ     JC
00601'014021-           DSZ     NJ
00602'000675            JMP     .DOJ2
00603'020030-           LDA     0,K
00604'101220            MOVZR   0,0
00605'040030-           STA     0,K
00606'024020-           LDA     1,NL
00607'125405            INC     1,1,SNR
00610'002015-           JMP     @AC3
00611'044020-           STA     1,NL
00612'125404            INC     1,1,SZR
00613'000646            JMP     .DOL2
00614'024010-           LDA     1,RT3
00615'044007-           STA     1,RT2
00616'000643            JMP     .DOL2
                  ;
                  ;     TO PERFORM DIVISION BY 2**6 MOD 65537
                  ;
```

```
00617'054015-.DVN:    STA     3,AC3
00620'020017-         LDA     0,N
00621'040024-         STA     0,I
00622'030035-         LDA     2,X0
00623'034037-         LDA     3,XC
00624'174400          NEG     3,3
00625'174000          COM     3,3
00626'102620          SUBZR   0,0
00627'151400 .DV1:    INC     2,2
00630'175400          INC     3,3
00631'025400          LDA     1,0,3
00632'125004          MOV     1,1,SZR
00633'000417          JMP     .DV2
00634'025000          LDA     1,0,2
      000006          .DO     6
00635'125223          MOVZR   1,1,SNC
00636'107000          ADD     0,1
00637'125223          MOVZR   1,1,SNC
00640'107000          ADD     0,1
00641'125223          MOVZR   1,1,SNC
00642'107000          ADD     0,1
00643'125223          MOVZR   1,1,SNC
00644'107000          ADD     0,1
00645'125223          MOVZR   1,1,SNC
00646'107000          ADD     0,1
00647'125223          MOVZR   1,1,SNC
00650'107000          ADD     0,1
00651'045000          STA     1,0,2
00652'014024-.DV2:    DSZ     I
00653'000754          JMP     .DV1
00654'002015-         JMP     @AC3
00655'126520 .RT20:   SUBZL   1,1
00656'044006-         STA     1,RT1
00657'030013-         LDA     2,SR
00660'113000          ADD     0,2
00661'151400          INC     2,2
00662'025000          LDA     1,0,2
00663'044034-         STA     1,APW
00664'000713          JMP     .IXJ2
                  ;
                  ;   SUBROUTINE TWD1
                  ;   TO PERFORM TWO'S COMPLEMENT TO
                  ;   DIMINISHED-1 CODE TRANSLATION
                  ;
00665'054015-.TWD1:   STA     3,AC3
00666'126520          SUBZL   1,1
00667'034037-         LDA     3,XC
00670'136400          SUB     1,3
00671'030035-         LDA     2,X0
00672'020017-         LDA     0,N
00673'040024-         STA     0,I
00674'175400 .TD1:    INC     3,3
00675'151400          INC     2,2
00676'021000          LDA     0,0,2
00677'101015          MOV#    0,0,SNR
00700'000410          JMP     .TD2
00701'101112          MOVL#   0,0,SZC
00702'000403          JMP     .+3
00703'122400          SUB     1,0
00704'041000          STA     0,0,2
```

```
00705'102400 .TD0:   SUB     0,0
00706'041400         STA     0,0,3
00707'000403         JMP     .TD3
00710'102520 .TD2:   SUBZL   0,0
00711'041400         STA     0,0,3
00712'014024-.TD3:   DSZ     I
00713'000761         JMP     .TD1
00714'002015-        JMP     @AC3
                  ;
                  ;  SUBROUTINE D1TW
                  ;  TO PERFORM DIMINISHED-1 TO
                  ;  2'S COMPLEMENT CODE TRANSLATION
                  ;
00715'054015-.D1TW:  STA     3,AC3
00716'020017-        LDA     0,N
00717'040024-        STA     0,I
00720'034037-        LDA     3,XC
00721'174400         NEG     3,3
00722'174000         COM     3,3
00723'030035-        LDA     2,X0
00724'151400   D11:  INC     2,2
00725'175400         INC     3,3
00726'021400         LDA     0,0,3
00727'101005         MOV     0,0,SNR
00730'000403         JMP     .+3
00731'126400         SUB     1,1
00732'000407         JMP     D10
00733'025000         LDA     1,0,2
00734'125535         INCZL#  1,1,SNR
00735'000404         JMP     D10
00736'125112         MOVL#   1,1,SZC
00737'000403         JMP     .+3
00740'125400         INC     1,1
00741'045000 D10:    STA     1,0,2
00742'014024-D12:    DSZ     I
00743'000761         JMP     D11
00744'002015-        JMP     @AC3
                  ;
                  ;  ADDRESS POINTERS
00745'000153'.SL0:   .A3
00746'000246'        .SL1
00747'000252'        .SL2
00750'000260'        .SL3
00751'000270'        .SL4
00752'000302'        .SL5
00753'000315'        .SL6
00754'000330'        .SL7
00755'000341'        .SL8
00756'000347'        .SL9
00757'000357'        .SL10
00760'000371'        .SL11
00761'000406'        .SL12
00762'000421'        .SL13
00763'000432'        .SL14
00764'000441'        .SL15
00765'000524'.SR0:   .B0
00766'000442'        .SR1
00767'000433'        .SR2
00770'000422'        .SR3
00771'000407'        .SR4
```

```
0012 FNT64
00772'000372'          .SR5
00773'000357'          .SL10
00774'000347'          .SL9
00775'000341'          .SL8
00776'000330'          .SL7
00777'000315'          .SL6
01000'000301'          .SR11
01001'000267'          .SR12
01002'000257'          .SR13
01003'000251'          .SR14
01004'000245'          .SR15
01005'000243'          .SR16
                 ;
                 ;     PAGE ZERO VARIABLES AND POINTERS
                       .ZREL
00006-000000 RT1:      0
00007-000000 RT2:      0
00010-000215'RT3:      .RT3
00011-000745'SL:       .SL0
00012-000153'SL0:      .A3
00013-000765'SR:       .SR0
00014-000524'SR0:      .B0
00015-000000 AC3:      0
00016-000000 AB3:      0
00017-000000 N:        0
00020-000000 NL:       0
00021-000000 NJ:       0
00022-000000 IC:       0
00023-000000 IPC:      0
00024-000000 I:        0
00025-000000 IP:       0
00026-000000 J:        0
00027-000000 JC:       0
00030-000000 K:        0
00031-000000 LE:       0
00032-000000 LE1:      0
00033-000000 PW:       0
00034-000000 APW:      0
00035-000000 X0:       0
00036-000000 XN:       0
00037-000000 XC:       0
00040-000001$AXC:      .XC
00041-000000 IFT:      0
00042-000040 NB:       40
00043-177400 HBYT:     177400
00044-000377 LBYT:     000377
00045-177772 MNS6:     177772
00046-000000 CODX:     0
                       .END

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

REFERENCES

1. C. M. Rader, "Discrete Convolution via Mersenne Transform", IEEE, Trans. Compt., Vol C21, Dec. 1972.

2. R. C. Agarwal and C. S. Burrus, "Fast Convolution using Fermat Number Transform with Application to Digital Signal Processing", IEEE Trans. Acoust., Speech, Signal processing , April, 1974.

3. R. C. Agarwal and C. S. Burrus, "Number Theoretic Transform to Implement Fast Digital Convolution", IEEE Proceeding, April, 1975.

4. L. M. Leibowitz, "A Simplified Binary Arithmetic for the Fermat Number Transform", IEEE Trans. Acoust., Speech, Signal Processing, Oct 1976.

5. L. R. Rabiner and B. Gold, "Theory and Application of Digital Signal Processing", Prentice Hall, 1974.

6. E. Grosswald, "Topics from the Theory of Number", Macmillan Co., N.Y. 1966.

7. A. V. Oppenhein and R. W. Schafer, "Digital Signal Processing", Prentice Hall, 1975.

8. P. R. Chevillat and F. H. Closs, "Signal Processing with Number Theoretic Transforms and Limited Wordlengths" IEEE ASSP, Jun 1978.

9. R. C. Agarwal and C. S. Burrus, "Fast One -dimensional Digital Convolution by Multidimensional Techniques", IEEE Trans. Acoust., Speech & Signal Processing, Feb 1974

10. N. S. Reddy and V. U. Reddy, "Simulation of Large Length Filter using Fermat Number Transform", Proc. IEEE Int. Conference on Acoust., Speech & Signal Processing, 1978.

11. J. H. Mc Clellan, "Hardware Realization of a Fermat Number Transform", IEEE Trans. Acoust., Speech, Signal Processing, Jun 1976.

12. C. M. Rader, "On the Application of Number Theoretic Method of High Speed Convolution to Two-dimensional Filtering", IEEE Trans. Circuits and Systems, Jun 1975.

13. R. H. Vanderkraat and A. N. Venetsanopoulos, "Two-dimensional Filtering Using Fermat Number Transforms", Proc. IEEE Int. Conference on ASSP, 1978

14. A. Rosenfeld, "Picture Processing by Computer", Academic Press, N. Y., 1969.

15. A. Chottera and G. A. Jullien, "Recursive Digital Filter in Image Processing", Dept. of Elec. Eng., Windsor.

VITA AUCTORIS

1950 Born 17th January, in Kedah, Malaysia.

1967 Completed secondary education at Chung Ling High School, Penang, Malaysia.

1973 Graduated from Cheng Kung University, Taiwan.

1974 Worked in Aviation Electronic Department, Air Asia Co. Ltd, Taiwan.

1979 Candidate for the Degree of M.A.Sc. in Electrical Engineering at the University of Windsor.