1-1-1984

Image processing using a two-dimensional digital convolution filter.

Rajendra P. Rathi
University of Windsor

IMAGE PROCESSING USING A TWO-DIMENSIONAL DIGITAL CONVOLUTION FILTER

by

Rajendra P. Rathi

A thesis
presented to the University of Windsor
in partial fulfillment of the
requirements for the degree of
Master of Applied Science
in
Department of Electrical Engineering

Windsor , Ontario, 1984

ABSTRACT

A two-dimensional digital convolution filter (referred as Convolver) employing the Fast number theoretic transform (FNTT) algorithm was built in the Department as an external peripheral to the SEL mini-computer for Image Processing. The purpose of this research is to analyse the design, suggest improvements, provide a working system and illustrate the use of the convolver through various examples.

The thesis describes the theoretical background and the hardware implementation of the convolver. A detailed explaination of the design considerations has been developed to provide an easy and complete reference for the user. Several comparisons have been presented, as part of analysis, to establish the efficiency of the techniques used in the design of the convolver. Timing diagrams have been prepared to facilitate the understanding of the processing of signals through the filter. Through-put rate calculations are included to indicate the speed of processing.

A systematic way to write the interfacing software has been explained. A directory of the available software, and a table of the main Integrated Circuits used in the convolver is included. Software has been written to make the convolver part of a user friendly image processing system.

Two design methods to improve the speed of processing are proposed.

dedicated to

Carl and Viola Glos

with love, honor and respect

ACKNOWLEDGEMENTS

I am deeply indebted to Dr. W.C. Miller and Dr. G.A. Jullien for their constant help, support and guidance throughout the course of my degree. I express my sincere gratitude to Dr. M. Shridhar for his valuable advices and encouragements. I express my sincere thanks to Dr. M. Ahmadi, Dr. M. Sid-Ahmed, and Dr. J. J. Soltis for their constructive comments and criticisms. I am also thankful to Mrs. S.A. Ouellette and Mrs. L.J. Kennedy for their invaluable assistance.

I sincerely acknowledge the support provided by the American-Nepal Education Foundation, Oregon, U.S.A. In particular I am grateful to Dr. Hugh B. Wood, Executive Director of the Foundation, for his personal interest in me.

I am especially grateful to Mr. Carl M. Glos and Mrs. Viola Glos whose guidance has been a perennial source of inspiration for me in my personal life throughout my degree at Windsor. I also thank Mr. Edwin D. Reimer and Mrs. Elfrieda Reimer for being my wonderful friends at Windsor. I express my sincerest thanks and gratitude to my parents for their help, love and patience. I also wish to mention my gratefulness to Dr. K.K. Aggarwal, my professor at Regional Engineering College, Kurukshetra, India, who motivated me for graduate studies and research.

TABLE OF CONTENTS

Appendix page

TABLE OF FIGURES

LIST OF TABLES

Table

of Basic block for larger images

M=N=256, J=K=17, t1=1 usec, t2=5 usec

LIST OF ABBREVIATIONS

A/D	Analog to Digital
B/R	Binary to Residue Conversion
CRT	Chinese Remainder Theorem
FIR	Finite Impulse Response
FNT	Fermat Number Transform
HSD	High Speed Device
MEM	Memory Buffer
MRC	Mixed Radix Conversion Method
NTT	Number Theoretic Transform
OIOO	Ordered Input Ordered Output
R/B	Residue to Binary Conversion
RNS	Residue Number System
RT	Rader Transform
G	Generator used in definition of NTT
pi	i th prime moduli
ri	i th residue
|.|m	operation modulo M
*	Sign to represent multiplication
**	Sign to represent exponentiation

Chapter I

INTRODUCTION

1.1 OBJECTIVE AND OUTLINE OF THE RESEARCH WORK

High speed digital filtering of two-dimensional signals is an essential element of contemporary research on signal processing. The application areas include image processing, pattern recognition, digital communication and robotic vision. The pre-processing of signals is also important for images obtained from space exploration photographs, radiographs, nuclear medical images, and geophysical data. The filtering, or the convolution of images with a filter kernel, can be achieved either by direct computation or indirectly, by the use of a transform having the cyclic convolution property.

One of the indirect techniques for the convolution is use of the Discrete Fourier Transform (DFT). The use of DFT for convolution became popular when Cooley and Tukey [23] introduced the efficient Fast Fourier Transform (FFT) algorithm to compute DFT resulting in a significant saving in computation and performance improvement over the direct method. The FFT uses the cyclic property of the complex exponential function to reduce the number of multiplications. The speed of the FFT and therefore the maximum data-processing speed still remains proportional to the (complex) multiplication time. Attempts were made to reduce the multiplication time, for example, Liu and Peled [25] proposed to use a bit-slicing algorithm and table look-up scheme to replace the conventional multiplier. Also, there is a definite drawback in the use of DFT for machine implementation, that there is finite precision representation of the transcendental multiplier functions. These approximations contribute to the error noise in the output.

An alternative transform domain technique which has attracted considerable interest in last few years is the use of the Number Theoretic Transform (NTT). Fermat Number Transforms (FNT) and Rader Transforms (RT), which are specific NTT's have been implemented [7,14]. A very attractive method of implementation of NTT's for convolution is, however, over the rings that are isomorphic to direct sums of Galois Fields [13]. This implies the use of the Residue Number System (RNS), which itself is of particular interest in digital signal processing [31] because of the parallel nature of its arithmetic. The RNS was extensively investigated by Szabo and Tanaka [3] in 1967 for use in the design of a general purpose computer. Residue techniques, however, did not receive wide-spread attention because the ferrite core memories used at that time were too expensive and bulky to justify their use to store the needed tables.

With the current advances in semi-conductor memory technology, the implementation of NTT's for signal processing is a viable alternative to conventional methods [7]. Though the transform domain representation of NTT sequences has no known practical interpretation, its implementation for convolution is meaningful. The only restriction to be observed is that the data points are small enough (scaled) so that the final result does not produce a data point greater than the ring modulus. Also, since the transform is defined over a finite ring, the results are exact. Further, in RNS arithmetic implementation, a multiplication can be replaced by a table look-up operation, and thus the throughput rate can be expected to be high with relatively low hardware cost. The number of bits used for data representation, however, should be small so that memories required for look-up tables are commercially available. Jullien [5] suggested a method to implement multiplications which results in tremendous memory saving and reduces memory requirement to a viable size, still using look-up tables.

The transform domain techniques are only attractive when one of the fast algorithms are employed in their computation. Fast Fourier Transform type algorithms can be applied to compute the NTT. The heart of such fast algorithm is a computational unit called a "Butterfly". One or multi dimensional butterflies have been used to compute the transforms [10]. The Ordered-Input-Ordered-Output (OIOO) algorithm [4] is of particular interest since it eliminates the data pre-shuffling. This further requires unity twiddle factors at the last stage of the butterrfly computation. Hence, the transform of the coefficients can be multiplied by the transform of the data points at the last stage of butterfly computation instead of the multiplication by the twiddle factors. This results in saving in the total number of computations.

For its practical use, any RNS arithmetic structure requires Binary to Residue (B/R) and Residue to Binary (R/B) converter units. Several hardware techniques [3] are available for a B/R converter. The separate need of such a B/R can be avoided if the A/D converter used gives the binary output which also is a residue. This can be a case when the ring modulus is larger than the possible maximum value of any data point. The final outputs obtained from a two or more moduli, however, have to be combined to obtain the result. The Chinese Remainder Theorem (CRT) is one of the methods [3], but it suffers from the disadvantage that it needs a mod M adder, where M is the dynamic range. The other method is via the use of a Mixed Radix Conversion (MRC) technique. The multiplication needed in this method can be implemented using look-up tables. This method has computational advantages over the CRT when fewer moduli are used.

Hardware realizations are normally fixed for a specific size of image operated on by a particular algorithm. Multiple use of the same piece of hardware and use of over-lap techniques can be used in processing of images of larger dimensions [18] than that of the basic block size. Such algorithms are attractive when working in a limited memory system, such as that of a mini-computer.

A special purpose digital signal processor is a dedicated piece of hardware whose function is to perform a specific set of processing algorithms (in real time) as a self contained subsystem. Obviously all the signal procesing algorithms can be implemented on a general purpose computer, however the speed of such implementations on general purpose computers are not particularly attractive. Many industrial needs have only one application in mind, for example, faulty part detection in an assembly line. Secondly, most general purpose computer architectures can not normally handle simultaneous computations. A dedicated piece of hardware, however, is designed to handle a large number of computations, and employs a parallel processing and pipelining to achieve speeds several orders of magnitude faster than general purpose computers.

This research is an extensive investigation into the processer architecture of a Fast 2-dimensional Digital Convolution Filter using Number theoretic transform techniques. The processor was built by Dr.H.K.Nagpal for the Signals and Systems group, Dept. of Electrical Engineering at the University of Windsor [29]. Although different hardware structures for the realization of Fast Fourier Transforms have been proposed, processing images and other inherently two-dimensional signals using a Fast Number Theoretic Transform (FNTT) with a two-dimensional butterfly structure is a relatively recent method. The various components which make up the complete processor are examined in this thesis.

In the realization of digital systems using special purpose hardware, the concepts of parallelism, multiplexing, and pipeling are of great importance in achieving a maximum value of performance-cost ratio for the particular application being considered. The theoretical considerations useful with respect to 'speed and cost trade-offs' are reviewed in this work.

The memory architecture needed for implementation of two-dimensional Ordered-Input-Ordered-Output NTT algorithm is investigated in light of the speed/cost trade-off. The implementation of such a butterfly is described in detail. The use of table look-up for mathematical operations, in particular multiplication, by a sub-modular approach, is investigated.

Several design aspects used in implementation of the processor are compared to establish the efficiency of the convolver. For example, computation saving by the use of a two-dimensional butterfly is compared with that of one-dimensional butterfly. Several other comparisons are made to support the architecture used in the convolver. For instance, the use of the Mixed Radix Conversion method is justified compared to the Chinese Remainder Theorem in the implementation of Residue to Binary converter. The implementation of multiplication by sub-modular look-up table and by the use of direct ROM multipliers is compared.

Timing considerations are made for serial sequential procesing (used in the convolver) and cascade processing, and speed/cost (efficiency) consideration for these methods are investigated for video-rate processing speed. Two structures, namely, a three-memory buffer structure and use of a 'complete' butterfly structure, have been proposed to improve the processing speed. The timing diagrams with respect to register contents in the butterfly of the convolver are presented.

Several examples of image filtering are presented to illustrate the application of the processor. The examples are taken from well defined images. A simple and approximate method to obtain the coefficients of a two-dimesnsional finite impulse response filter is described. Several standard filters are used for Image Smoothening, Image Enhancement and other feature extraction on images. The results obtained from three different methods in software, namely direct use of convolution, using the FFT and using the FNTT.

The use of block-mode filtering is investigated in filtering of very large sequences, in a limited main-memory system. The choice of the basic-block size is a trade-off with speed. Theoretical comparisons for this trade-off are presented.

1.2 THESIS ORGANIZATION

Chapter-2 provides the theoretical background on the application of Fast Number Theoretic Transform techniques in digital filtering of two dimensional sequences. It details the modular arithmetic, algebraic constraints to be observed in the use of the NTT and the restrictions imposed from practical point of view. Further it describes the concepts of the 2-dimensional OIOO-NTT algorithm, and the method of Mixed Radix conversion used in the residue to binary conversion. The design considerations used in the convolver are detailed in this chapter.

The implementation of the transform computational element, the butterfly, and the multiplication in the butterfly using the sub-modular approach, are described in Chapter-3. A number of comparisons with respect to speed, cost and memory storage are included in this part to describe the performance of the processor. Various timing diagrams are also included in this part.

Chapter-4 deals with the hardware and the functional details of the image processor. In particular, both the memory architecture and the butterfly operation are described. The High Speed Device (HSD) interface of the Filter with the mini-computer SEL and the control logic are examined. The discussion on the software of the HSD is included in the appendix. The steps for the use of the convolver are described in this chapter.

In the next part, Chapter-5, we detail the results of filtering by the use of several standard filters on test images. The applications in mind were Image Smoothening and Edge Enhancement. This final part considers the processing of large arrays (larger than can be processed in one block) by block-mode filtering. The time of processing, which depends on block size has been compared. Chapter-6 presents the conclusions of this research work.

Chapter II

DIGITAL FILTERING USING FAST NUMBER THEORETIC TRANSFORM TECHNIQUES

2.1 INTRODUCTION

In digital image processing, as well as in other areas, it is desirable to filter a two-dimensional discrete signal $x(i,j)$ by convolving that signal with the two-dimensional digital pulse response of applied filter $h(i,j)$ producing an output signal $y(i,j)$. The two-dimensional convolution is defined as

$$\begin{aligned} y(i,j) &= x * h \\ &= \sum_{k=0}^{M-1} \sum_{l=0}^{M-1} x(k,l) \cdot h(i-k, j-l) \qquad (2\text{-}1) \\ & \qquad i,j = 0,1,\ldots\ldots M-1 \end{aligned}$$

where the sequences x, h and y are assumed to have square shape of dimension $(N\times N)$, $(L\times L)$ and $(M\times M)$ respectively, $M \geq N+L-1$.

Processing signals with a digital computer or with special purpose digital hardware involves the implementation of computational schemes on sequences of numbers. For example, Eqn. (2.1) can be implemented by actually taking the sum of products as defined or by indirect methods. The indirect method consists of taking the transforms of sequences x and h, multiplying the two transforms and taking an inverse transform of the product. The indirect methods are attractive, because with viable restrictions on the length of sequences, computationally efficient algorithms can be developed which have advantages over direct methods in terms of speed and thus the cost of filtering. The most common technique to reduce the computational cost of convolution is by the use of the Discrete Fourier Transform computed via use of Fast Fourier Transform (FFT) algorithm.

It is interesting to note that the FFT has been used to compute convolutions and many hardware structures have been implemented [4,24] with slight variations to the basic algorithm suggested by Cooley and Tuckey [23]. Each structure looks at the hardware/speed trade-off associated with both the computational elements and the supporting structure. However, this procedure is time-consuming on mini-computers even with multiplication hardware installed, due to the large number of complex multiplications required. Further there is considerable build-up of round-off error because of the finite precision in representing real numbers on digital computers. Filter designs using ROM oriented RNS arithmetic units [11] and implementation of the FFT with the use of Residue Number System [31] have been suggested for improved efficiency. Since for convolution we are only interested in the Cyclic Convolution Property (CCP) of the transform, it is natural that alternatives to complex multiplications involved in FFT twiddle factors have been investigated. Number Theoretic Transforms (NTT), which are defined as part of Generalized Discrete Fourier Transforms (GDFT), use integer twiddle factors and have gained considerable interest for several years as a class of signal processing algorithms. The hardware of the Image convolver uses Number Theoretic Transform for employing the indirect method of filtering.

2.2 DEFINITION OF NUMBER THEORETIC TRANSFORM

Number Theoretic Transforms are defined as part of a class of Generalized Discerete Fourier Transforms and are computed over finite fields [13],

$$X_k = \left| \sum x_n \, \varepsilon^{nk} \right|_M$$

$$x_n = \left| N^{-1} \sum X_k \, \varepsilon^{-nk} \right|_M \qquad (2.2)$$

where N is the sequence length and M represents the modulus of the field arithmetic; the generator ε is an Nth root of unity (ε**N=1; ε**N1$\neq$1 mod M for 1$\leq$N1<N) and N exists. It has been suggested that NTT's be implemented in rings which are isomorphic to a direct sum of Galois fields:

$$R = GF(p_1^{\,r_1}) + GF(p_2^{\,r_2}) + \ldots\ldots \qquad (2.3)$$

where the pi are primes and r represents the degree of the extension fields. The results of the operation can be recovered by either using the Chinese Remainder Theorem or a mixed radix conversion [3] algorithm. This amounts to implementing the NTT using the Residue Number System (RNS) and the inherent parallelism of RNS implementation can be made to advantage to obtain faster speed of processing. We first discuss some of the basic concepts of RNS from number theory relevant to the NTT in the next section.

2.3 MODULAR ARITHMETIC

DEFINITION-2.1: Two integers a and b are said to be congruent mod M if

$$a = b + k.M \qquad (2.4)$$

where k is some integer and M is the modulus. The b is residue of a mod M when

$$0 < b < M$$

and is written as

$$a = b \pmod{M}$$

DEFINITION-2.2: All integers are congruent mod M to some integer in the finite set (0,1,2,......,M-1) and let the set of elements be combined by two different operations '+' and '.' both mod M . Then this set is called the ring of integers mod M and is denoted by Zm. Such a ring is a commutative ring with identity [9].

DEFINITION-2.3: If in a ring of integers multiplicative inverses exist for all nonzero integers, this ring is known as a Field . It can be shown that Zm is a field if and only if M is a prime. The set of all invertible elements of a ring is a group with respect to the operation of multiplication and is called a "multiplicative group".

The following basic arithmetic operations are defined in modular arithmetic.

1. Addition: Example, $7+12 \equiv 2 \pmod{17}$
2. Negation: Example, $-7 \equiv 10 \pmod{17}$
3. Subtraction: Example, $7-12 \equiv 7+(-12) \equiv 7+5 \equiv 12 \pmod{17}$
4. Multiplication; Example, $7 \times 12 \equiv 16 \pmod{17}$
5. Multiplicative Inverse: Multiplicative Inverse of an integer b in Zm exists if and only if b and M are relatively prime. In that case b is an integer such that $b \times b^{-1} \equiv 1 \pmod{M}$. It may be however noted that when M is a non-prime integer, not all members of the set have multiplicative inverses.

   Example: $7^{-1} \equiv 5 \pmod{17}$

   for $7 \times 5 \equiv 1 \pmod{17}$

   $3^{-1} \equiv 5 \pmod{14}$ as $3 \times 5 \equiv 15 \equiv 1 \pmod{14}$

   but $2^{-1} \pmod{14}$ does not exist.
6. Divison: x/y exists if and only if y has an inverse and x/y is contained in the ring. In that case $x/y \equiv x.y^{-1}$.

   Example: $12/6 = 12 \times 3 = 2 \pmod{17}$

DEFINITION-2.4: If pi is a prime, the elements {0,1,2,....pi-1} form a field with addition and multiplication modulo pi. In any finite field the number of elements must be a power of a prime (pi**ri), where ri is a positive integer and an element (primitive root) must exist, powers of which can generate all the non-zero elements of the field. Such a field is commonly denoted by the symbol GF(pi**ri) and is called a Galois Field [9].

DEFINITION-2.5: The Residue representation of an integer in the RNS takes the form of an L-tuple

$$X = (x1, x2, \ldots\ldots, xl)$$

of the least positive residues with respect to the set of moduli

$$(m1, m2, \ldots\ldots ml)$$

The range of numbers which can be uniquely coded in RNS are

$$0 \leq X < \prod_{i=1}^{L} mi = M$$

A signed integer system can be developed attaching a positive sign to numbers in the range {0,1,.....M/2-1} for M even or {0,1,....(M-1)/2} for M odd, and a negative sign to the number in the range {M/2,M/2+1,... M-1} or {(M+1)/2,......M-1} respectively. The operations in RNS can be carried independently for each of the moduli. The correct answers would be obtained regardless of intermediate overflows of an arithmetic computation if the result is within the range of the number system.

As mentioned in introduction, for the existence of transforms with the DFT structure given in Eqn.(2-2) and having the Cyclic Convolution Property (CCP), it is necessary that an integer exist that is an Nth root of unity. We will consider this problem using modular arithmetic.

First Euler's function $\Phi(M)$ is defined as the number of integers in Zm that are relatively prime to M. Obviously then for M a prime number $\Phi(M)=M-1$. If M is a composite number and its prime factored form is denoted by

$$M=(p1)^{r1}\cdot(p2)^{r2}\cdots\cdots(pl)^{rl}$$

then the general expression for Φ is [9]

$$\Phi(M)=M(1-1/p1)\cdot(1-1/p2)\cdots\cdot(1-1/pl).$$
$$=\prod_{i=1}^{l}(pi-1) \qquad (2.5)$$

THEOREM-2.1: Euler's theorem states that for every δ prime to M

$$\delta^{\Phi(M)}=1 \pmod M$$

For M prime this reduces to Fermat's theorem.

THEOREM-2.2: Fermat's theorem states that for M a prime number,

$$\delta^{(M-1)}=1 \pmod M$$

which holds for all nonzero elements of Zm since they are all relatively prime to M if M is a prime.

There are certain roots of unity that are of particular interest. If N is the least positive integer such that

$$\varepsilon^{N} \equiv 1 \pmod{M} \;;\quad \varepsilon^{N1} \neq 1 \pmod{M};\; 1 \leq N1 < N \qquad (2.7)$$

then ε is said to be a root of unity of order N, or ε is a primitive Nth root of unity.

If the order of ε is equal to $\phi(M)$, then ε is called a primitive root. If M is a prime and ε is a primitive root, the set of integers

$$X = \{\varepsilon^{k} \pmod{M},\; k=0,1,2,\ldots,M-2\} \qquad (2.8)$$

is the total set of nonzero elements in Zm, and all nonzero elements in Zm can be generated by powers of the primitive root. This, thus characterizes the entire field. The nonzero classes of Zm form a cyclic multiplicative group of order M-1 {1,2,.....M-1}, with multiplication modulo M, isomorphic to the addition group {0,1,.....M-2} with addition modulo M-1.

Euler's theorem implies that if ε is of order N then N must divide $\phi(M)$, denoted by $N|\phi(M)$. If M is a prime it can be shown that roots of order N exist if and only if $N|(M-1)$ and the roots are given by

$$\varepsilon = \varepsilon_o^{(M-1)/N} \qquad (2.9)$$

where ε_o denotes a primitive root. More generally if ε is a root of order N then

$\varepsilon^{**}k$ is of order N/k if $k \mid N$

$\varepsilon^{**}k$ is of order N if N and k are relatively prime.

This implies that the number of roots of order N is given by $\phi(N)$ and, therefore, the number of primitive roots is $\phi(\phi(M))$. These relations allow one to calculate all of the roots of all possible orders from one primitive root.

Example:

Let $M=7$, $Zm=\{0,1,2,3,4,5,6: '+', '.'\}$

$\phi(1)=1$ $\phi(2)=1$ $\phi(3)=2$

$\phi(4)=2$ $\phi(5)=4$ $\phi(6)=2$

$\phi(7)=6$

Consider raising each element of Z7 to powers from 1 to 6 (mod 7), Tab-(2.1).

Table-(2.1)

N=	0	1	2	3	4	5	6
1	1	1	1	1	1	1	1
2	1	2	4	1	2	4	1
3	1	3	2	6	4	5	1
4	1	4	2	1	4	2	1
5	1	5	4	6	2	3	1
6	1	6	1	6	1	6	1

This illustrates several very interesting features. Consider the various roots of order N, Tab-(2.2).

Table-(2.2) Roots of order N

N	roots of order N
1	1
2	6
3	2,4
6	3,5

Only those N that divide $\phi(M)=\phi(7)=6$ have roots that belong to them. The number of roots is given by $\phi(N)$ and the number of primitive roots is $\phi(\phi(M))=2$ and they are 3 and 5. Note that both of the primitive roots generate all the nonzero elements.

For a nonprime M, δ has an inverse given by $\delta^{\phi(M)-1}$ if δ and M are relatively prime. It can be noted that for M a composite rather than a prime number, Zm is not a field since all elements will not have inverses. There is no primitive root that will generate the entire ring, only subsets with $\phi(M)$ elements. Let M have the following unique prime factorization.

$$M = (p1)^{r1} . (p2)^{r2} .... (pl)^{rl}$$

When the arithmetic has to be performed mod M, it can be performed modulo each prime power (pi)**ri separately [9] and the final result mod M can be obtained using the Chinese Remainder theorem [3]. When the arithmetic mod (pi**ri) is performed in finite fields, then every field with N elements is isomorphic to every other field with N elements.

Now we return to the discussion on the design of the NTT processor. We notice the following requirements for the CCP to exist and the NTT to be defined over the finite field.

THEOREM-2.3: A length N transform having the DFT structure will implement cyclic convolution if and only if there exists an inverse of N and an element α, a root of unity of order N, i.e.,N is the least positive integer such that

$$\alpha^{**}N = 1$$

This is a very general result applying to both rings and fields that are finite or infinite and it has been developed from a variety of points of view [25]. For M a composite number as represented in Eqn.(2-5), we can obtain the results of opeation mod M by combining the results obtained from the operation modulo each (pi**ri).

Therefore, the length N number theoretic transform having the CCP in Zm must also have the CCP in Z{pi**ri} for i=1,2,....l. This requires that α (mod pi**ri) be an integer of order N and must exist in Z{pi**ri}, i.e.,N is the least positive integer such that

$$\alpha^{**}N = 1 \pmod{pi^{**}ri}, \quad i=1,2,....l.$$

Furthermore, since the inverse transform requires N ; the inverse of N should exist in Z{pi**ri}, or, N should be relatively prime to M. Now we find that by Euler's theorem

$$N \mid \partial(pi^{**}ri), \qquad i=1,2,.....l.$$

$$\text{or } N \mid pi^{ri-1}(pi-1) \text{ because } \partial(pi^{**}ri)=pi^{ri-1}(pi-1).$$

Since N is relatively prime to M (or its factors)

$$N \mid (p_i - 1) \qquad i=1,2,\ldots\ldots l.$$

$$N \mid \gcd\{p_1-1, p_2-1, \ldots\ldots p_l-1\}$$

We define O(M) as the greatest common divisor (gcd) of the (p_i-1)

$$O(M) = \gcd\{p_1-1, p_2-1, \ldots\ldots p_l-1\}$$

therefore, $N \mid O(M)$

This gives us

THEOREM-2.4: A length N transform having the DFT structure will implement cyclic convolution mod M if and only if

$$N \mid O(M)$$

and this establishes the maximum transform length in Zm as

$$N_{max} = O(M)$$

This is a very important theorem that states exactly what the possible transform lengths for a given modulus are.

2.3.1 An Example of convolution using NTT when M is a prime

Consider two sequences

$$x = (2, -2, 1, 0)$$

$$h = (1, 2, 0, 0)$$

whose convolution is desired. From overflow consideration, it is sufficient if we define the transforms over GF(17)

$$M=17 \qquad N=4$$

Now since M=17, the integer 2 is of order 8

therefore $(2^{**}2) = 4$ is an α of order 4.

The transformation matrix T is given by

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

or,

$$T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & 16 & 13 \\ 1 & 16 & 1 & 16 \\ 1 & 13 & 16 & 4 \end{bmatrix} \quad (\text{mod } 17)$$

since 4 =-4 (mod 17) =13 (mod 17) and the inverse Transformation Matrix is

$$T = 13 \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 13 & 16 & 4 \\ 1 & 16 & 1 & 16 \\ 1 & 4 & 16 & 13 \end{bmatrix}$$

The Transforms of x and h are given by

$$X = T.x = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 4 & 16 & 13 \\ 1 & 16 & 1 & 16 \\ 1 & 13 & 16 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 15 \\ 1 \\ 0 \end{bmatrix}$$

$$= [\ 1,\ 10,\ 5,\ 9\ ] \quad (\text{mod } 17)$$

similarly H = [3,9,16,10]

and thus Y = X.H

= [3,5,12,5] (mod 17)

Taking the inverse transform of Y,

y = (2,2,14,2) (mod 17)

According to our assumption, integers are supposed to lie between -8 and 8. Therefore

$$y = (2,2,-3,2)$$

which is the correct answer.

2.4 PRACTICAL CONSIDERATIONS IN CHOOSING δ, N AND M FOR AN NTT

Although the class of all possible number theoretic transforms seems very large at first consideration (in fact, infinite), closer examination shows that very few seem to be attractive for use in signal processing. Agarwal and Burrus [6] summerise the criteria which would make a particular NTT to be attractive in comparison to other implementations of convolution. They list that for NTT to be computationally efficient three requirements are:

1. (a) N should be highly composite (preferably a power of 2) for a fast FFT-type algorithm to exist and

   (b) N should be large enough for practical sequence lenths

2. since complex multiplications take most of the computation time in calculating the FFT, it is important that multiplication by powers of δ be a simple operation. For machine implementation, this is possible if the powers of δ have binary representations with very few bits; preferably also a power of two, where multiplication by a power δ reduces to a word shift.

3. In order to facilitate arithmetic mod M for machine implementations, M should have a binary representation with a very few bits and should be large enough to prevent overflow.

Unfortunately the conditions given by above theorems in sec-(2.3) do not give a systematic way of determining the "best" choices. Usually M has to be selected first and N and δ are determined suitably. When the modulus M is chosen to be a Fermat Number as

$$M = Ft = 2^{2^t} + 1 = 2^b + 1 \quad , \; b=2^{**}t \qquad (2.10)$$

then a promising class of NTT's can be obtained [8]. Such transforms are called Fermat Number Transform (FNT). A special class of such transform is when the value of δ is chosen δ=SQRT(2.) . These transforms are described by Rader and are known as Rader Transforms [9].

2.5 DESIGN CONSIDERATION FOR NTT USED IN CONVOLVER

As we mentioned in section-(2-4), it is usually but not always the case that a value of M is chosen first and suitable transform length N and the generator δ is determined. We will follow the same approach.

2.5.1 choosing M

In the ring of integers mod M, conventional integers can be unambiguously represented only if their absolute value is less than M/2. If the input integer sequences x(n) and h(n) are so scaled that |y(n)| never exceeds M/2, we would get the same results by implementing convolution in the ring of integers modulo M as that obtained with normal arithmetic. In most digital filtering applications, h(n) represents the impulse response and is known a priori; also the maximum magnitude of the input signal is usually known. In this case, we can bound the peak output magnitude [7] by

$$|y(n)| \leq |x(n)|max * \sum_{n=0}^{N-1} |h(n)|$$

One possible solution to this overflow problem involves segmenting the words into shorter blocks and convolving them separately [8]. Another approach to solving the sequence length vs word length constrain is to use block processing where the sequence of length N is broken into smaller blocks and the results are combined. However a better alternative to this problem can be argued when hardware implementation of digital filters using the FNTT is desired. The method works as follows:

The convolution is implemented modulo two different primes p1 and p2 where p1 and p2 are chosen such that cyclic convolution in Zp1 and Zp2 is easily implemented on the same machine. By exploiting the inherent parallelism of RNS arithmatic, the processing modulo p1 and p2 can be performed in parallel and the results combined to give the correct and exact solution by use of either the Chinese Remainder Theorem or a Mixed Radix Conversion method .

In our implementation this latter technique has been employed; the value of p1 and p2 have chosen to be 641 and 769 which are prime numbers. The operations with respect to prime moduli are generally represented by pi and we will follow the same convention. The prime moduli chosen are of the form pi=Mi=q.2**p+1, where (2**p)=128. These numbers are chosen for several reasons. First, both the numbers, 641 and 769, support 128 point transforms and are the largest two primes ($\leq$ 10-bits) , that support this transform length. Secondly, the design of the Convolver was aimed for Image processing where 128 or 256 levels are sufficient for image representation. Thus the input data array will have no number greater than 255 and hence the residues with respect to mod pi is same as the data itself. This will save us the hardware cost in the sense that a binary to residue converter can be avoided. Further since,

$$M = \prod_{i} p_i \qquad i=1,2,\ldots.1$$

in our case we obtain M=641x769 < 2**19 (19-bits). Also by simulation results operated over several images using the Fast Number Theoretic Transform, it was found that the convolution result never exceeded a 17-bit binary representation. In a sense, we have thus provided a 'cushion' of 2-bits which is reasonably sufficient. Also, it was essential from the point of view of Euler's Theorem (sec-2.2) that $N_{max} = O(M) = gcd\{p1-1, p2-1, \ldots\ldots, pl-1\}$ and since the sequence length was decided as N= 128 for 2-D processing of (128x128) images, the choice of p1=641 and p2=769 was most suitable for hardware implementation.

2.5.2 choosing N and ε

For the implementation of NTT's for digital image processing, it is essential that an FFT-type algorithm be utilized to compute the NTT of sequences in order to achieve the best speed/cost ratio. This requirement implies that the sequence length N should be a highly composite number, preferably a power of two. Gonqalez [16] descibes that for almost all image processing applications images represented by dimensions in the range (128x128) to (512x512) are sufficient when 8-bit representation is used for each data-point (gray level). Thus the choice is limited to have N= 128, 256 or 512. Now since the prime moduli 641 and 769 support a 128-point transform, the sequence length was chosen to be 128. The hardware cost was another factor in deciding the size of the basic block to be (128x128). Further this complies with the requirement of Euler's Theorem.

Once the value of M and N have been fixed the value of ε is determined by finding an Nth root of unity in each field as described previously. The main restriction on the parameters is the value of N so that a FNTT can be utilized.

The other parameters are chosen based on N. Since the implementation of multiplication is through the use of look-up tables, the restriction on δ_i is an algebraic rather than hardware related.

2.6 ORDERED-INPUT-ORDERED-OUTPUT-NTT ALGORITHM

The traditional FFT-type algorithms require pre-shuffling or post-ordering of the data which results in a reduced througnput rate and increased hardware cost. Using a hardware implementation, various structures have been suggested [1] where pre-shuffling is performed at the host-computer or by providing additional logic circuitry. The convolver, however, uses an Ordered-Input-Ordered-Output (OIOO) algorithm for implementing the FNTT. The algorithm was proposed by Corinthios [6] and was originally described for a 1D-radix-2-FFT employing use of serial-sequencial pipelining using a single Butterfly Unit (BU). In our implementation, the original idea was extended and modified [10]. For example, we utilize a 2D-radix-2 Butterfly for the FNTT and the operations are performed in parallel for the two moduli. The development of the two-dimensional OIOO-NTT algorithm is outlined in Appendix-(A).

It is seen from the Eqn.(A-6) that the computation of the NTT of the vector i can be divided into n-stages where each stage performs the operations specified by the operators $\Psi_i R_i$. The operators of any stage operates on the output of the previous stage and the operator $\Psi_n R_n$ of the first stage operate on the vector f. Since the two-dimensional operators Ψ_i and R_i of a stage have been defined as the Kronecker product of the 1-D operators, μ_i, q_i, s_i, the output of a stage can be computed by sequential application of the opeators along each dimension of the input to the stage. Thus a stage consists of the sequential application of an r-point NTT, permutation and twiddle factor multiplication operations to the points in each dimension of the two-dimensional representation of the input to the stage. An analysis of memory organization and saving in computation is given in the next chapter (sec-3.5).

2.7 RESIDUE TO BINARY CONVERSION

The use of an NTT computed over several Galois fields allows us to use Residue Number System concepts. The input data is converted to the corresponding residues before computation with the NTT processor. By the use of a signed representation, numbers in the range {-p/2, p/2-1} can be uniquely coded. A combinatorial logic circuit for such conversion is given in [3]. However, the input array can be suitably scaled so that numbers are in the range 0 and 255 (8-bit). Since the moduli are larger than the maximum possible value in the input data, the residues of the sequence is the sequence itself. Thus we avoid the need of a binary to residue converter unit.

The signals obtained after processing are the residues too, and have to be converted to their correct binary representation. This is done by either the Chinese Remainder Theorem or by the use of a Mixed Radix Conversion Method. For the reasons described in sec-(3.9) it was preferred to use the Mixed Radix Conversion (MRC) to obtain the final result from the residues.

2.7.1 Use of MRC in obtaining the final result

The mixed-radix system is a weighted number system where a number x is represented as <an,......,a2,a1> where

$$x = an \prod_{i=1}^{n-1} Ri + \ldots\ldots\ldots + a3.R2.R1 + a2.R1 + a1$$

where the Ri's are the radices and ai's are the mixed radix digits $0 \leq ai < Ri$. Numbers in the range $[0, \prod_{i=1}^{n} Ri - 1]$ can be represented by this manner. By choosing the radices to be the moduli (pi=Ri) when pi are prime numbers,

$$x = an \prod_{i=1}^{n-1} pi + \ldots\ldots\ldots + a3.p2p1 + a2.p1 + a1 \qquad (2.25)$$

which is also equivalently represented by

$$x = \langle rn, \ldots\ldots\ldots r2, r1 \rangle$$

$$= \text{residue of } x \text{ w.r.t. different moduli}$$

The ai's can be determined sequentailly in the following manner, starting with a1, since

$$a1 = x \pmod{p1} = \text{residue of } x \text{ w.r.t. } p1 = r1$$

$$a2 = \{(x-a1)/p1\} \pmod{p2} = |[x/p1]| \pmod{p2}$$

$$a3 = \{(x-a1-a2.p1)/p1.p2\} \pmod{p3}$$

$$= |[x/p1.p2]| \pmod{p3}$$

....

$$a_i = |[x/p_1.p_2....p_{i-1}]| \pmod{p_i}$$

Once the mixed digits are known, then by application of Eqn-(2.25) the value of x can be determined.

Example: Let $p_1=13$ $p_2=17$
then $p_1 \pmod{p_2} = 4$

if the given $x = \langle r_1,r_2 \rangle = \langle 8,9 \rangle$
then x would be obtained as follows:

Moduli	13	17	
$x=\langle r_1,r_2 \rangle$	8	9	$a_1=8$
$x-a_1$	0	1	
$[(x-a_1)/p_1]$ mod p_2		4	$a_2=4$

hence

$$x = a_2.p_1+a_1 = 4.13 + 8 = 60$$

which is the correct result.

The method of Mixed-radix conversion thus can be utilized to find the binary representation by performing operations in a serial pipeline fashion and is advantageous over the Chinese Remainder Method of conversion in our case.

2.8 CONCLUSION

In this introductory chapter, we have presented the theoretical background necessary to understand the processor architecture used in the convolver. The modular arithmetic necessary to develop the concepts for use in the implementation of Fast Number Theoretic Transforms for digital filtering has been discussed. The theoretical and practical considerations to choose an NTT are outlined. The Ordered-Input- Ordered-output (OIOO) algorithm has been considered. Finally the use of a Mixed Radix Conversion method for Residue to Binary conversion is presented.

Chapter III

IMPLEMENTATION OF BUTTERFLY AND ANALYSIS ON THE CONVOLVER

3.1 INTRODUCTION

In the previous chapter we developed the theoretical background necessary for implementation of Fast Number Theoretic Transform techniques to use them in digital filtering of two-dimensional sequences. In particular, an OIOO-NTT algorithm was considered for the 2-D-radix-2 butterfly structure.

This chapter has two parts. The first part discusses the structure and implementation of the butterfly unit. In particular, the implementation of multiplication in the butterfly unit by the use of a sub-modular look-up table approach is discussed in detail. Next, this chapter describes various topics related to the convolver as part of the analysis. This part includes the timing diagrams, throughput rate considerations, and various comparisons in terms of computational requirements. Two design extensions are also suggested to achieve a higher processing rate. Finally, the two popular methods of residue to binary conversion implementation are compared.

3.2 EFFICIENT IMPLEMENTATION OF BUTTERFLY

As stated in sec-(2.2) NTT's could be implemented over a ring which is isomorphic to a direct sum of Galois fields. This amounts to implementing the transform using the RNS. Since in RNS arithmetic the number of different elememts and the result of any operation (+,.) is bounded by a maximum number (the modulus of the operation), it is possible to precompute the results of all possible operations and store them in ROM arrays. Whenever an operation has to be performed on two operands, the operands are concatenated as a single address to a ROM and the result obtained as a table look-up. This results in tremendous speed, limited only by the access time of the ROM. As memory prices continue to decrease and as the advances in semiconductor high density memory systems multiply, the look-up table approach for mathematical operation in RNS becomes more and more attractive. In fact, the table-look up appraoach by use of ROM (or EPROM) arrays could be considered as the "best" solution [21] for high speed realization and hence for high throughput rates.

Implementation of multiplication is particularly attractive by this method, since implementation of mod M multiplication is difficult with the conventional binary multipliers. The multiplication in the butterfly unit is implemented through the table-look up approach for faster operation.

The basic idea of implementing an efficient Butterfly Operation (BO) was described by Jullien [5] for a 1D-radix-2 by substituting the conventional multiplier by the use of Look-up Tables and thus achieving increased throughput rate . Since the butterfly operation is basic to any transform domain implementation, the design suggested in [5] was implemented in the convolver with little modification. The actual implementation was extended to a 2-D-radix-2 structure and data were pre-multiplexed to obtain an efficient hardware configuration and to ensure an OIOO-NTT algorithm.

A butterfly operation can be performed having either a Decimation in Time (DIT) or Decimation in Frequency (DIF) structure. The DIF structure suggests [1,2] that the multiplication by twiddle factors is applied after the addition/ subtraction operation over the data points participating in the butterfly. A 1D-radix-2-DIF butterfly is described by

$$A= (a+b)$$

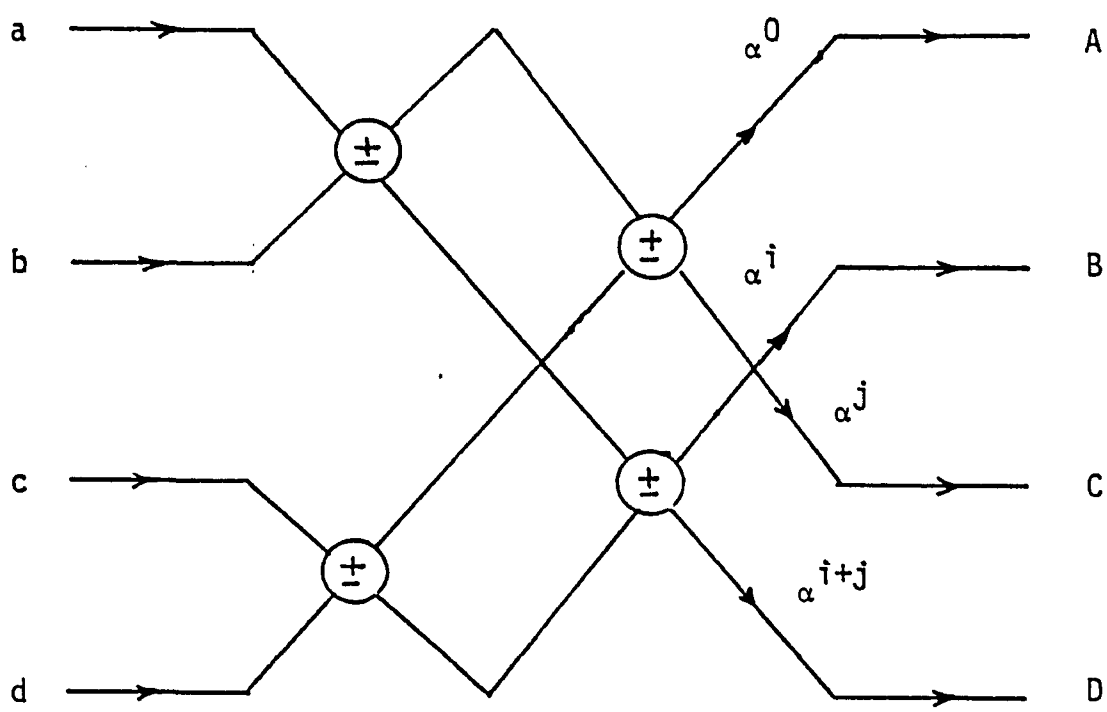
$$B= (a-b) .\delta^{**}k \qquad (3.1)$$

where a and b are the inputs to the butterfly at stage n-1, A and B are the outputs (input to stage n), and the index k depends on the location of the butterfly. Similarly, in the case of a 2-D-radix-2 butterfly [2] there are four data points aoo,ao1,a1o,d as input and the output A,B,C,D is as given below:

$$A= (a+b+c+d)$$

$$B= (a-b+c-d) .\delta^{**}i$$

$$C= (a+b-c-d) .\delta^{**}j$$



$$a = x(s.t) \qquad A = X(s,t)$$

$$b = x(s + \frac{N}{r^2}, t) \qquad B = X(s + \frac{N}{r^2}, t)$$

$$c = x(s, t + \frac{N}{r^2}) \qquad C = X(s, t + \frac{N}{r^2})$$

$$d = x(s + \frac{N}{r^2}, t + \frac{N}{r^2}) \qquad D = X(s + \frac{N}{r^2}, t + \frac{N}{r^2})$$

Fig. (3.1) A 2-D-radix-2 Butterfly (OIOO-algorithm).

$$D= (a-b-c+d) . \delta^{**}(i+j) \qquad (3.2)$$

where the input/output relations hold for a complete stage of butterflies and the indeces, i and j, depend on the exact location of the butterfly in a particular stage of the operation (Fig-3.1). There are two basic arithmetic operations in such an implementation, namely, multiplication and addition . The use of ROM-array structure over conventional adders for addition does not significantly improve the computation efficiency. For example, a 16-bit addition can be performed in less than 100 nsec by the use of fast adder circuits. The memories used in look-up tables have the same order of access time.. However, multiplication can be made a faster process through the use of look-up tables.

3.2.1 Implementing Multiplication using the sub-modular approach

By the use of look-up tables, the multiplication can be performed as simple and as fast as the addition. The computation time is given by the sum of ROM-access time (ta) plus latch settling time (tl). For example the throuput rate of operation through currently available 8Kx8 bit PROM's (such as the Intel 2732) is in excess of 5 MHz. Such a multiplier scheme was considered in [21] for butterfly pipelining frequently used in signal processing. To add further to the speed, and to decrease the net memory requirements for large dynamic range, a Sub-modular approach to multiplication was suggested by Jullien [5]. We will show the tremendous saving obtained through this approach. First, let us consider the algorithm.

It was shown in sec-(2.3) that if δ is a primitive root of the prime, pi, then a mapping given by

$$xn = δ^{**}kn$$

$$kn=0,1,2,\ldots\ldots,pi-1$$

will generate all the non-zero elements of the set given by Zpi. Also there exists an isomorphism between a multiplicative group x having elements {xn}= {1,2,3,.........,pi-1} with multiplication modulo pi, and the additive group k having elements {kn}= {0,1,2,......pi-2} with addition modulo pi-1 when pi is a prime. Thus,

$$\left|x_n x_j\right|_{p_i} = \varepsilon^{\left|k_n + k_j\right|_{p_{i-1}}} \qquad (3.4)$$

which suggests that multiplication can be performed in three steps:

1. Find the index ki for each number
2. Add indices, mod (pi-1)
3. Perform the inverse index operation

The above steps can be implemented directly using an all ROM-array structure with a pipe-lining arrangement. The memory requirement of order {(pi**2) xN-bits} in the second step, when addition is performed mod (pi-1), is equivalent to directly performing look-up for multiplication and it seems that no improvement results. However substantial saving accrue because we can perform addition in a modulus other than the prime modulus. We can compute addition by decomposing the modulus into two relatively prime (sub-)moduli u and v, and performing the addition in Zu and Zv. The result is reconstructed using a look-up table which incorporates the submodular reconstruction, modulus overflow correction and inverse index-look up. The choice of u and v should satisfy u.v > 2.pi. We here describe the steps involved in sub-modular approach of multiplication with an example.

3.2.2 calculating the entries in the Look-up tables

The steps in calculating the entries in the look-up table are as follows:

1. Generating Submodular Index Tables:

   Once the primitive root δ is decided for the modulus pi, a table based on x=|δ**k|pi is constructed. By inverting the table with respect to its address and contents of this table, we have a table of indices which is reduced to two index tables modulo u and v. For example, for pi=19 with {u,v=7,8} and δ=2 we form a table of mapping x=|2**k|(mod 19) and rearrange it to obtain |k|(mod 7) and |k|(mod 8), Fig-(3.2).

2. Submodular Addition Table Construction:

   The addresses of these tables are found by concatening the two-input sub-moduli residues to be ad-

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
x	1	2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10

(a)

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
k	0	1	13	2	16	14	6	3	8	17	12	15	5	7	11	4	10	9
k_7	0	1	6	2	2	0	6	3	1	3	5	1	5	0	4	4	3	2
k_8	0	1	5	2	0	6	6	3	0	1	4	7	5	7	3	4	2	1

(b)

Fig. (3-2) Submodular Index Tables.

ded. The contents of the table address is the submodulo addition of the input residues, Fig-(3.3). At certain locations corresponding to unused addresses in the table we store a code (say, 8) to represent the invalid operation of finding the index of zero.

3. Reconstruction Table:

As indicated above, the reconstruction table will incorporate the followings:

(a) Submodular Reconstruction: This is obtained using the Chinese Remainder Theorem for the residues r1 and r2 with respect to u and v and the table entries are computed to correspond to a value r_i given by

$$r_i = \left| r_1.v.\left|\frac{1}{v}\right|_u + r_2.u.\left|\frac{1}{u}\right|_v \right|_{u.v}$$

(b) Modulus Overflow Correction: The overflow of the modulus (pi-1) can be corrected by the following operation on r as

ri= |ri| (mod pi-1) (3.5)

(c) Inverse Index Look-up: An inverse mapping corresponding to

yi= |δ**ri| (mod pi) (3.6)

is employed on the corrected value of ri to obtain the entries for the table.

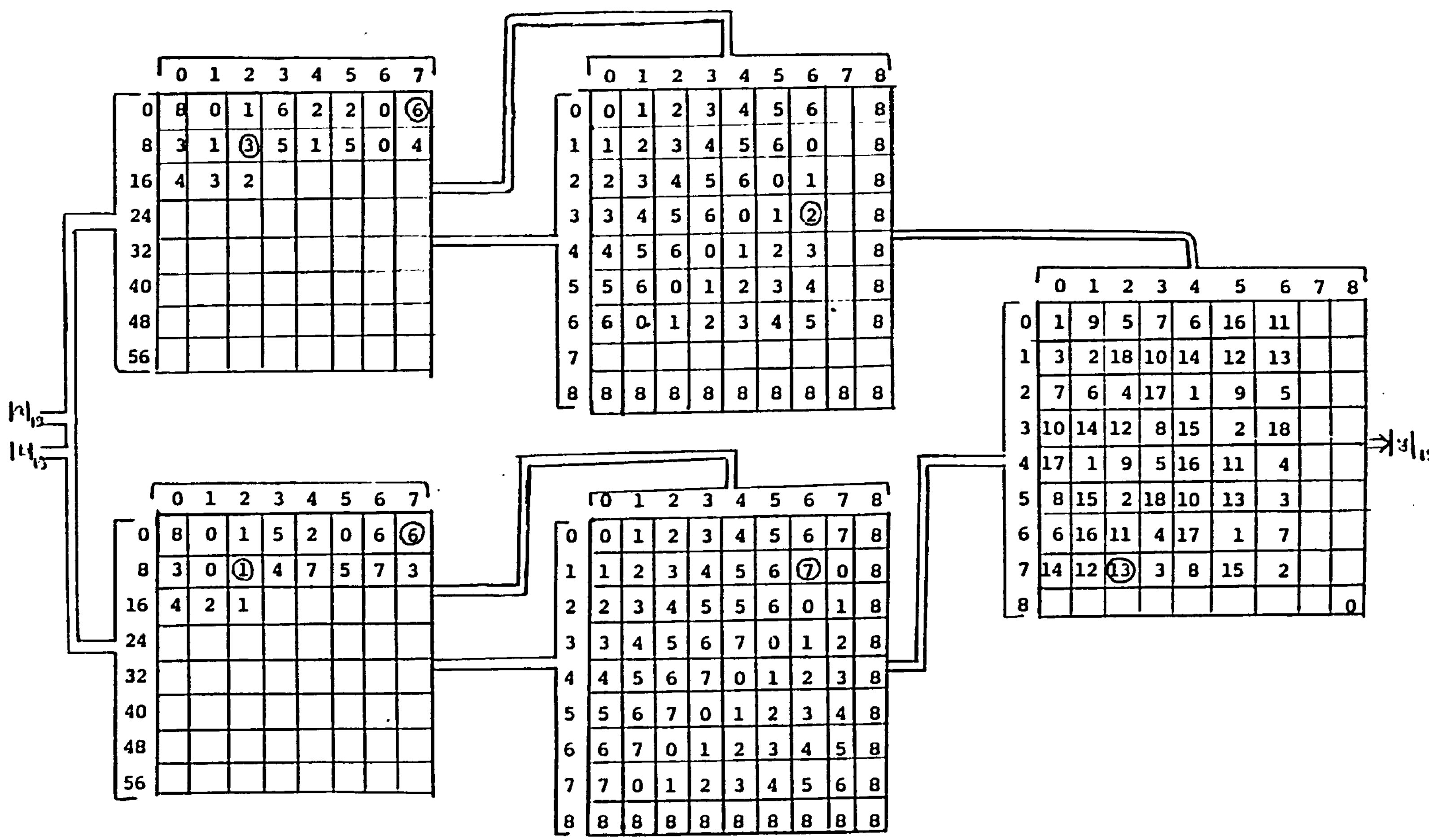


Fig. (3.3) A modulo 19 ROM array multiplier Sub-moduli 7 and 8.

Let us take, for example, {pi=19}, {u,v=7,8}, then u.v > 2.pi. Let us multiply 7 by 10 (mod 19). We find

$$\left|\frac{1}{v}\right|_u = \left|\frac{1}{8}\right|_7 = 1; \quad \left|\frac{1}{u}\right|_v = \left|\frac{1}{7}\right|_8 = 1$$

$$r_1 = 2;\ r_2 = 7 \text{ (from step 2)}$$

$$\therefore\ r = \left|r_1.8.1 + r_2.7.7\right|_{56} = \left|16 + 343\right|_{56}$$

$$= 23$$

$$\text{and}\quad y = \left|2^{|23|_{18}}\right|_{19} = \left|2^5\right|_{19} = 13$$

The interconnection employed in this all "ROM" structure for multiplication is shown in Fig. (3.3) and the intermediate results for the example just worked out are circled.

3.2.3 memory saving by the use of sub-modular approach

It is observed that the memory requirements using the submodular approach are greatly reduced. To calculate the memory requirements and to make a comparison of memory saving by the sub-modular approach of multiplication as compared to the direct table look-up, we define the Memory Saving Ratio (MSR) as,

$$MSR = \frac{\text{Mem (dir)}}{\text{Mem (sub)}} \qquad (3.7)$$

where Mem (dir) represents the memory requirement by direct multiplicatioon using look-up and Mem (sub) represents the memory requirement by use of sub-modular approach. Here we consider a case when one modulus is broken into two sub-moduli.

Let N be the number of bits by which each of the moduli is represented, c is the number of bits by which each of two sub-moduli is represented, and b is the number of bits at the output of the result.

1. a direct implementation of multiplication by ROM-look up would require (for each modulus),

$$\text{Mem (dir)} = (2^{**}N)\,x\,(2^{**}N)\,xb \quad \text{bits} \qquad (3.8)$$

In case of convolver, this will be

$$= (2^{**}24)\,x\,10 \quad \text{bits}$$

$$= 160 \text{ Mega bits !}$$

which is impractical and the cost unjustified.

2. an implementation using sub-modular approach would require (for each modulus),

In step a) for generating submodular residues of indices

$$\text{Mem (sub-a)} = (2^{**}N)\,x\,c \quad \text{bits/sub-modulus}$$

In step b) for performing the index addition

$$\text{Mem (sub-b)} = (2^{**}c)\,x\,(2^{**}c)\,x\,c \quad \text{bits/sub-modulus}$$

In step c) the reconstruction of the result is obtained. The memory required is

$$\text{Mem (sub-c)} = (2^{**}c)\,x\,(2^{**}c)\,xb \quad \text{bits}$$

Thus the total memory requirement using the sub-modular approach is (for each modulus),

$$\text{Mem (sub)} = 2\text{x}\text{Mem(sub-a)} + 2\text{x}\text{Mem(sub-b)} + \text{Mem(sub-c)}$$

$$= 2\text{x}(2^{**}N)\text{xc} + (2^{**}c)\text{x}(2^{**}c)\text{x}2c + (2^{**}c)\text{x}(2^{**}c)\text{xb}$$

and the Memory Saving Ratio (MSR) is

$$\text{MSR} = \frac{(2^{**}N)\text{x}(2^{**}N)\text{xb}}{2c.(2^{**}N+2^{**}2c)+b.(2^{**}2c)}$$

$$= \frac{b.(2^{**}2N)}{(2^{**}2c)(2c+b)+2c.(2^{**}N)} \qquad (3.9)$$

which for $N \cong 2c \cong b$ approximates

$$\text{MSR} \cong (1/3).(2^{**}N)$$

The MSR for various values of b corressponding to N=2c is listed in Table-(3.1). A memory saving ratio of about 341 is obtained in case of the convolver (for each modulus). It is observed that larger the dynamic range, higher is MSR. This is an expected result.

Table-(3.1) Memory Requirements by Direct and Submodular approach of Multiplication and the Memory Saving Ratio (MSR); 1K= 1024, 1M= 1024x1024

N=2c	Memory Requirements (bits): direct method	Memory Requirements (bits): sub-modular appr.	MSR
for b=8			
8	384 K	6 K	64
10	8 M	28 K	292.57
12	128 M	128 K	1024
for b=10			
8	512 K	6.5 K	78.77
10	10 M	30 K	341.33
12	160 M	136 K	1204.70
for b=12			
8	768 K	7 K	109.71
10	12 M	34 K	361.41
12	192 M	144 K	1365.30

3.2.4 further reduction in memory requirements

We can further reduce the memory requirement by providing an obvious simplification when the objective is to perform NTT where a multiplication by a twiddle factor of the form δ**l is involved. Since multiplication in Eqn.(3.2) is between some arbitary data and (δ**l) where l depends upon the position of the butterfly in a stage, the sequence {δ**l} can be prestored with the mapping already applied. This will reduce the memory requirement in step a) of sub-modular approach by half. This excludes the memory required to store the reduced sequence (δ**l). The memory required to store the twiddle factors in any case is small and depends simply on the number of points involved in the transform at a time ,e.g, in our case 128-different twiddle factors would require 128xcx2= 128x12 = 1536 bits/modulus = 1.5K-bits/modulus.

3.3 TIMING CONSIDERATION FOR THE BUTTERFLY

Next, we describe various topics which are related to the convolver as part of its analysis. In this section, the timing diagram for the butterfly unit is considered.

The working of the butterfly has been described in detail in sec-(4.7) and the stage-wise description is given there. Here we present a timing-analysis of the butterfly so that we can estimate the through- put rate. Fig-(3.4) shows the butterfly computational unit. The register-contents of the butterfly is shown in Fig-(3.5). With the first clock [pipclk5] register R11 receives the first data point (a1), which is buffered in R12 when the next data (b1) is latched in R22 at the second half of [pipclk5]. The result of the addition, (a1+b1), is then moved to R13 at the next clock when data (c1) arrives in R11. The next half of the clock allows fourth data sample (d1) in R22 while (a1-b1) is latched to R23. The next clockpulses allow addition, (c1+d1), and subtraction, (c1-d1), while data points (a2 and b2) for the second butterfly are received in the unit.The results (c1+d1) and (c1-d1) are latched in R34 and R44 in a

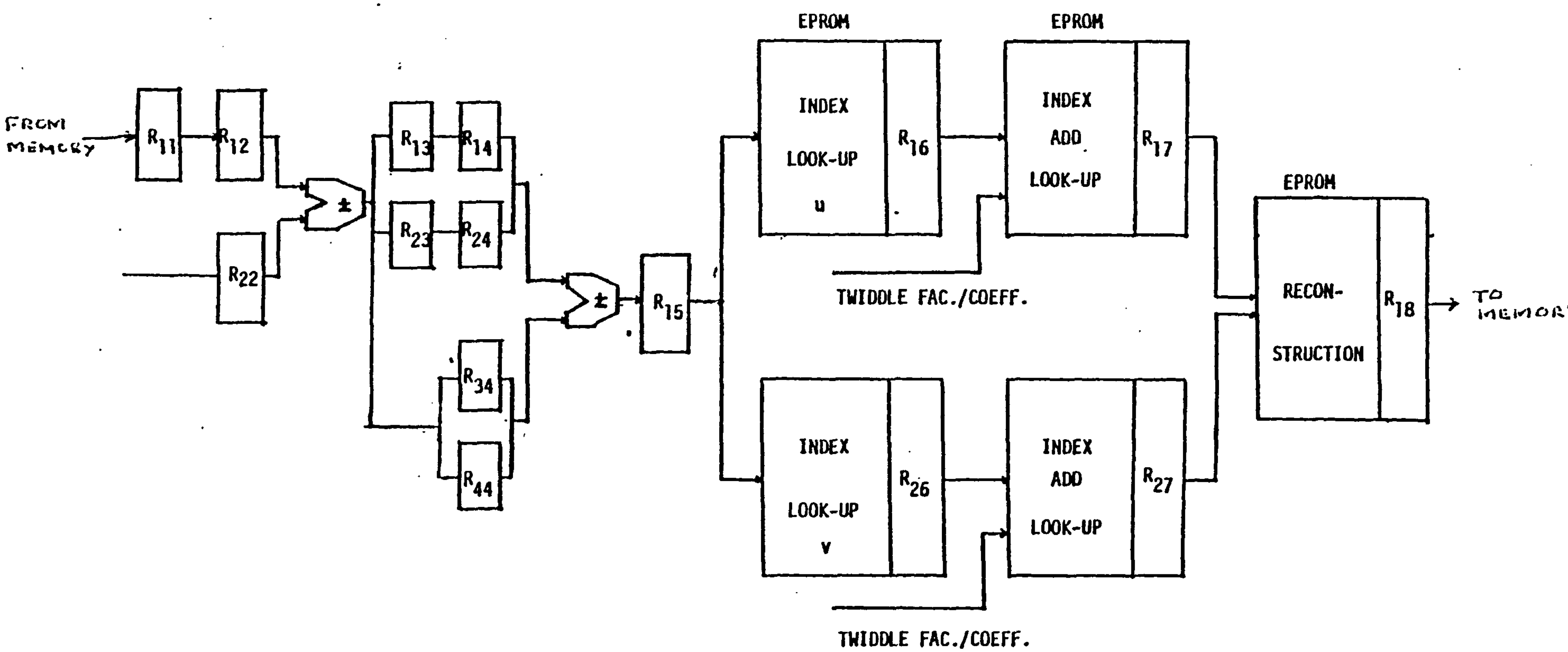
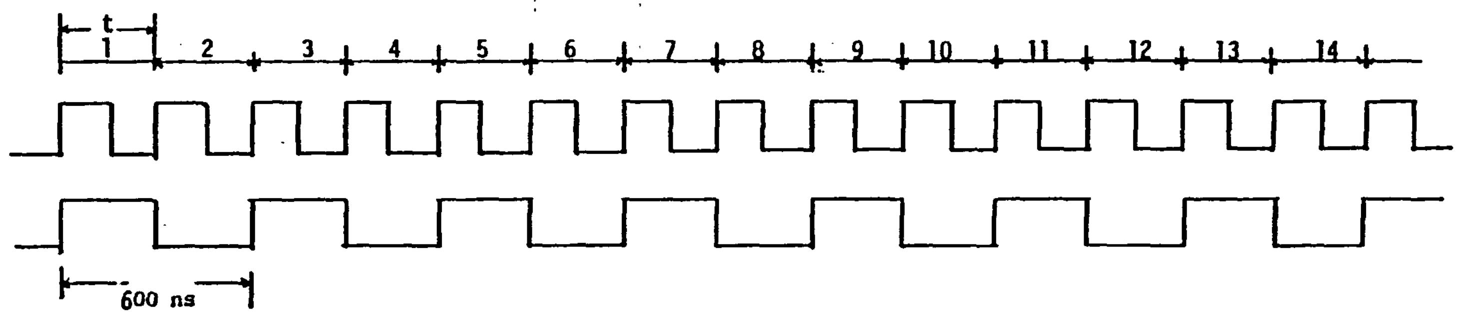


Fig. (3.4) The Butterfly Computation Unit



PIPELINE REGISTERS	REGISTER CONTENTS AT SUCCESSIVE CLOCK PULSES (CLR = CLEAR)													
R_{11}	a		c		a_1		c_1		a_2		c_2			
R_{12}/R_{22}		a/b		c/d		a_1/b_1		c_1/d_1		a_2/b_2		c_2/d_2		
R_{13}/R_{23}			a+b/CLR	a+b/a-b			a_1+b_1/a_1-b_1							
$R_{14}/R_{24}/R_{34}/R_{44}$					a+b/a-b c+d/c-d	a+b/a-b c+d/c-d								
R_{15}						a+b+c+d $=A_0$	a-b+c-d $=A_1$	a+b+c+d A_2	a-b-c+d $=A_3$	A_0'	A_1'	A_2'	A_3'		
R_{16} & R_{26}							Index A_0	Index A_1	Index A_2	Index A_3					
R_{17} & R_{27}								Add Index	Add Index	Add Index	Add Index				
R_{18}									Recons. A	Recon. B	Recon. C	Recon. D	A_1	B_1	C_1
										Output A	Output B	Output C	Output D	Output A_1	Output B_1

Fig. (3.5) The Timing Diagram and Register Contents in the Convolver Butterfly.

sequential fashion. When [pipclk1] occurs, the result of the second addition (a1+b1+c1+d1) and (a1+b1-c1-d1) are sequentially stored in R15. The other two data points, c2 and d2, are received in the unit. The operations after R15 for each of the butterflies are straight forward and occur with each basic clock pulse. The data is divided into two channels corresponding to two sub-moduli. The data goes to next register (Ri6, i=1,2) after error correction and index look-up. The added indices are then latched in R17 and R27 at the next clock pulse. Meanwhile, (a1+b1+c1-d1) and (a1-b1-c1+d1) are calculated and input for next butterfly computations are received in the pipeline. At the next clock pulse, the result from the index add look-up table is sent to the Reconstruction EPROMS and are latched in R18. Thus a rate of (1/t) data per sec is maintained where 't' is basic clock period or half of the period of [pipckl5].

It is noticed that a delay of 9-cycles is introduced (fig-3.4) between an input and the corresponding output of the butterfly of that stage. The butterfly operation proceeds for 14 stages taking into account the forward and the inverse transforms. At the final stage of the forward transform, the NTT of the filter coefficients is used as multilier instead of multiplication by unity twiddle factors.

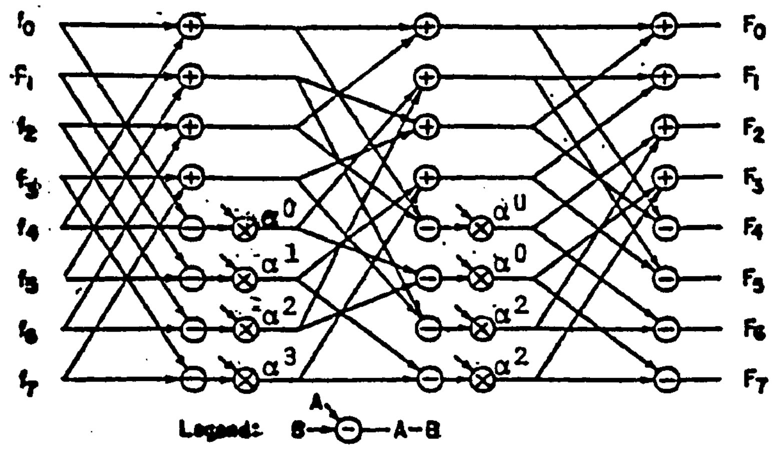
3.4 THROUGHPUT RATE CONSIDERATIONS (OIOO-ALGORITHM)

3.4.1 serial sequential processing

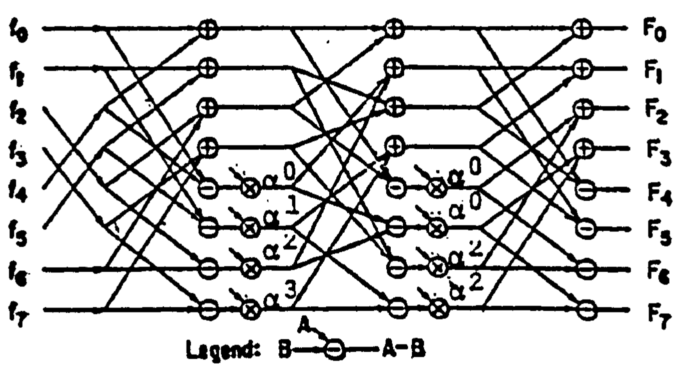
The Ordered-Input-Ordered-Output (OIOO) algorithm has been discussed in sec-(2.6). The mathematical development of the algorithm is given in Appendix-(A). The algorithm can be characterized by the following:

1. Let NxN be the size of the input matrix and r be the radix of the butterfly operation. Then the input matrix is divided into r**2 blocks and each block is further divided into r**2 sub-blocks. If the data is accessed sequentially, then the address seperation between the data points at any stage of operation, along any dimension, is always (N/r**2), except at the first stage. The data must be seperated by (N/r) words for the first stage of operation. A 8-point transform structure is shown in Fig-(3.6). To use the same hardware configuration for the first stage as that of other stages, the input matrix is permuted while loading the matrix into the input memory buffer.

2. A butterfly computation (OIOO-algorithm) consists of preweighting (addition/subtraction) followed by weighting (multiplication by twiddle factors). At the nth stage of operation, the twiddle factors are all unity and the weighting is not required.



(a)



(b)

Fig. (3.6) (a) Graphical Representation of the Ordered-input Ordered-output algorithm for the case N = 8 (1-D-radix-2)

(b) Fig. (a) with Permutation of Sequence While Loading in the Memory Buffer.

The above characterizations can be used to calculate the throughput rate.

MAXIMUM THROUGHPUT-RATE POSSIBLE:

Let tp be the time of performing preweighting and tm be the time of performing preweighting followed by weighting. Since the n-th iteration involves only preweighting, the total time required in the forward transform is

$$Tc = (n-1).(N.N/r.r).tm + (N.N/r.r).tp \qquad (3.10)$$

where n is the number of stages. In hardware implementations, the multiplication by the twiddle factors can be made as fast as addition through the use of a look-up table. In that case, we can simplify eqn-(3.10) by substituting tm = 2.tp, and

$$Tc= (N.N/r.r).(n-1/2).tm \qquad (3.10a)$$

Using the relation (3.10a), it is possible to calculate the time required in computing a forward transform. If addition can be performed in 300 nsec. (i.e. tm= 300 nsec.), then the transform of a matrix of size (128x128) can be performed in

$$Tc= 7.98 \text{ msec.} \qquad (3.10b)$$

which implies a sampling rate of 2.051 MHz. If fast adders are employed in the circuitary (say tm= 70 nsec) then the time for a transform would be given by,

$$Tc= 1.86 \text{ msec.} \qquad (3.10c)$$

which supports a sampling rate of 8.2 MHz.

To compute the convolutions, the filter coefficients is multiplied to the preweighted output from the final stage of the forward transform, and the inverse transform of the resulting sequence is obtained. The multiplication by the filter coefficients can assumed to be equivalent to weighting at the final stage of the forward transform. Since, there is no need of weighting at the final stage of the inverse transform, the total time to compute convolution is given by,

$$T(conv) = (N.N/r.r).\{n.tm + (n-1).tm + tp\} \quad (3.11)$$

Assuming tp = 2.tm as above, we have

$$T(conv) = (N.N/r.r).(2n + 1).\ tm \quad (3.11a)$$

Using tm = 300 nsec for a matrix of size (128x128), the time required to compute convolutions is

$$T(conv) = 17.8 \text{ msec.} \quad (3.11b)$$

indicating a sampling-rate of .919 MHz. The use of fast adders and memories (using tm = 70 nsec) will give a processing time of

$$T(conv) = 5 \text{ msec} \quad (3.11c)$$

which supports a sampling rate of 3.76 MHz.

VIDEO-RATE PROCESSING:

In signal processing, a process is said to operate in real-time if the data-processing rate is higher than or equal to the data sampling (input) rate. Thus, depending on the particular application, the "real-time processinq time" may be different.

In many applications, it is desired that the processinq speed supports a video rate. Such hiqh speed procesinq requirement is essential in diqital tele-vision transmission. Many Robotics applications also require a very hiqh processinq rate. The video-rate is

$$\begin{aligned} Vr &= 30 \text{ pictures per sec} \qquad (3.12) \\ &= 1 \text{ picture every } (1/30) \text{ sec} \\ &= 33.3 \text{ msec per picture} \\ &= .492 \text{ MHz, for a } 128 \times 128 \text{ pixel picture.} \end{aligned}$$

Based on the calculations (eqn.3-11), it appears that the application of the OIOO-alqorithm, with the basic time of a mathematical operation as tm = 300 nsec, supprots a video rate.

3.4.2 Speed consideration in the Convolver

The use of a 2-D-radix-2 OIOO-alqorithm requires that the four data points participate in any butterfly computation (Fiq-3.1). The mathematical operations required by the butterfly computation (eqn-3.2) can then be implemented by the use of 8 adder/subtracters and 3 multipliers. These requirements can be simplified if a sacrifice in speed is aqreed. In that case, a "quarter" of the butterfly is used to process the data. The data input to the butterfly is in a sequential fashion, one data at the occurrence of every clock pulse. This type of arrangement saves the hardware cost required to implement a "complete" butterfly unit. In fact, the convolver design uses only a quarter of the butterfly and the data flow is in a sequential fashion. The use of pipeline arrangement lets a data with every clock pulse, and there is delay of 9 clocks between the input and the corresponding output at every stage of computation. Thus, the total time of convolution depends on

1. the total number of data points
2. the time of a basic clock
3. the delay between the input and the corresponding output .

The total time of convolution is then given by,

$$T(conv) = 2.n\ [\,(N.N).t + 9.t\,] \qquad (3.13)$$

where t is the basic clock period. The basic clock used in the convolver is of 300 nsec.. Thus an image of size (128x128) is processed in a total time of

$$T(conv) = 68.8 \text{ msec.} \qquad (3.13a)$$

which implies a sampling rate of 0.24 MHz.

Table-(3.2) Comparison of Time required for convolution by different methods of Processing.

t= 300 nsec			
Method	Time Reqd. for Convolution (msec)		
Pic.Size =	64x64	128x128	256x256
Serial Seq. Processing	4.60	18.43	73.72
Processing in Convolver	-	68.8	-
7-stage Cascade Processing	0.62	2.63	10.53
t= 70 nsec			
Ser.Seq.Process.	1.07	4.30	17.20
Proc. in Convolv.	-	16.05	-
7-st.Cascade Processing	0.15	0.62	2.45

3.4.3 Cascade Processing

A cascade processor may prove to be prohibitively expensive. Neverhteless, instead of using an input buffer memory and oscillating data successively betweeen two memories, if we provide a number of memory arrays and butterflies equal to the number of stages, then the processing speed can be increased by a factor of n, where n is the number of stages. The convolution computation time, in that case, is given by,

$$T(conv) = 2. \lceil (N.N/r.r).t + D \rceil \qquad (3.14)$$

where D represents the delay between the input and the corresponding output data point. This delay is very small and can be neglected in calculation of speed of the processor. With t= 300 nsec., the convolution time for a matrix of size (128x128) is 2.45 msec. (a sampling rate of 6.5 MHz). A further gain in speed by a factor of 4 can be achieved through the use of a "complete" butterfly.

The convolution time obtainable through the different methods of processing is given in Table-(3.2). The I/O time between the SEL computer and the convolver for a picture of size (128x128) bytes (through the use of HSD interface) is approximately 15 msec. and requires an overhead time of 5 msec.. Since the use of convolver was intended in conjunction with the SEL mini-computer, efforts to use very high processing rates may not be supported. Thus the use of cascade processing is not recommended.

3.5 THREE-MEMORY STRUCTURE FOR FASTER PROCESSING

In this section and in the next section we consider two of the design improvements for faster processing. The I/O rate between the SEL minicomputer and the convolver is 1.2 usec/32-bit word (through the use of the High Speed Data Interface). An image of (128x128) bytes is transferred from the computer to the filter in 5 msec. by multiplexing 4 data points to form a word. The filtered image is received in an array of 16-bit per data point, and thus the data transfer

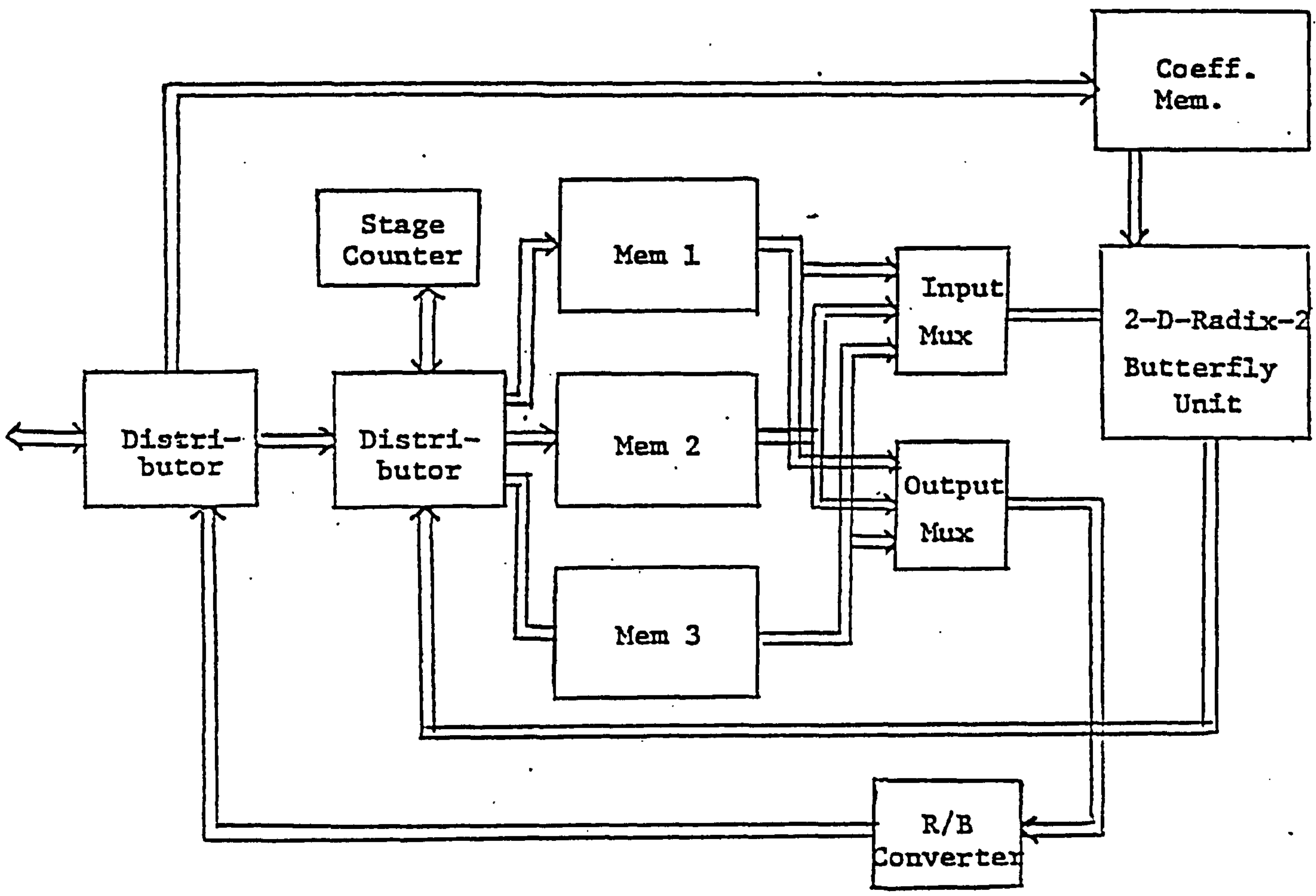


Fig. (3.7) A Three Memory Buffer Structure for Improved Speed of Processing.

time is 10 msec.. The total I/O time, including the overhead of about 5 msec., is 20 msec.. The processing time to filter an image is 68.8 msec.(sec-3.4). In processing of an image, the two times do not over-lap. This is because, the convolver design does not allow any communication between the host computer and the filter while the processing is in progress. Also, there are only two memory buffers, MEM1 and MEM2, in the filter. The data oscillates back and forth between these two buffers at the various stages of operation to compute the convolution. A three-memory structure is suggested (Fig.3.7) which can reduce the I/O idle time.

Whether the data is sent from a video-digitizer camera or from the host computer where the images have been stored, two of the buffers store the input and output of the intermediate NTT stages. The third buffer can be employed to collect the sampled input for a second image while the filter processes the first image. Similarly, the final result can be collected in one of the two buffers involved in the processing of that image and can be transferred through the I/O channel while the processing of the second image is in progress. All three memory structures are required to be identical. Assume that MEM1 stores the input for the first image. Now MEM1 and MEM2 are used to store all the intermediate results of the butterfly stages for that image and the final result is available in MEM2. While the first image is being processed, the second image is written into memory

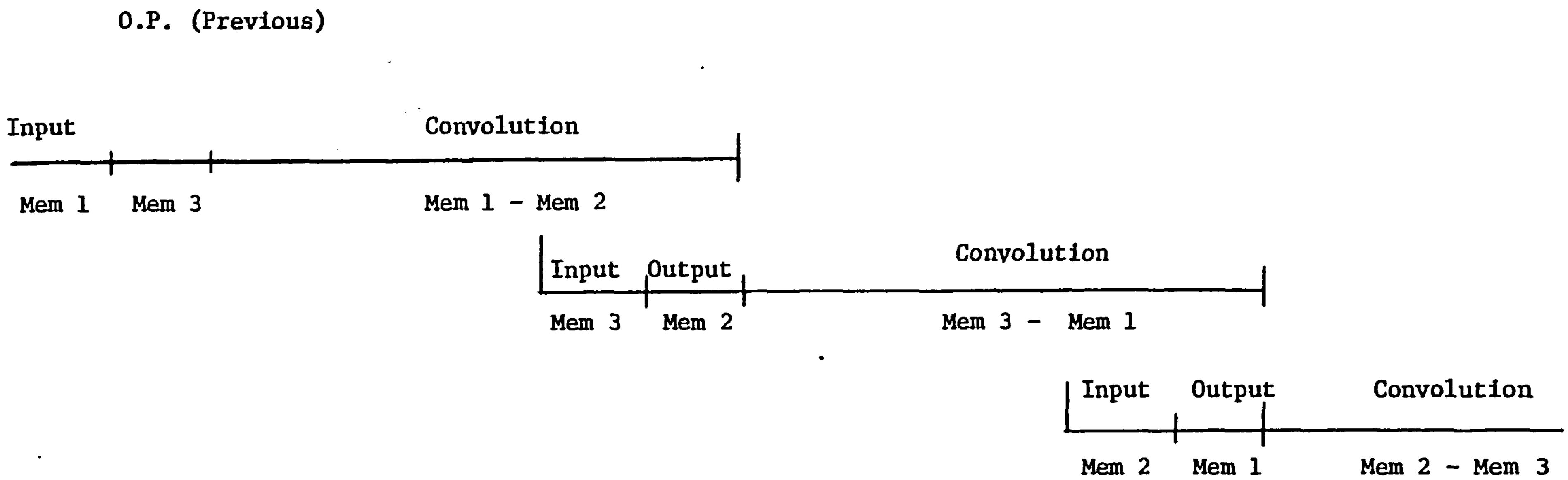


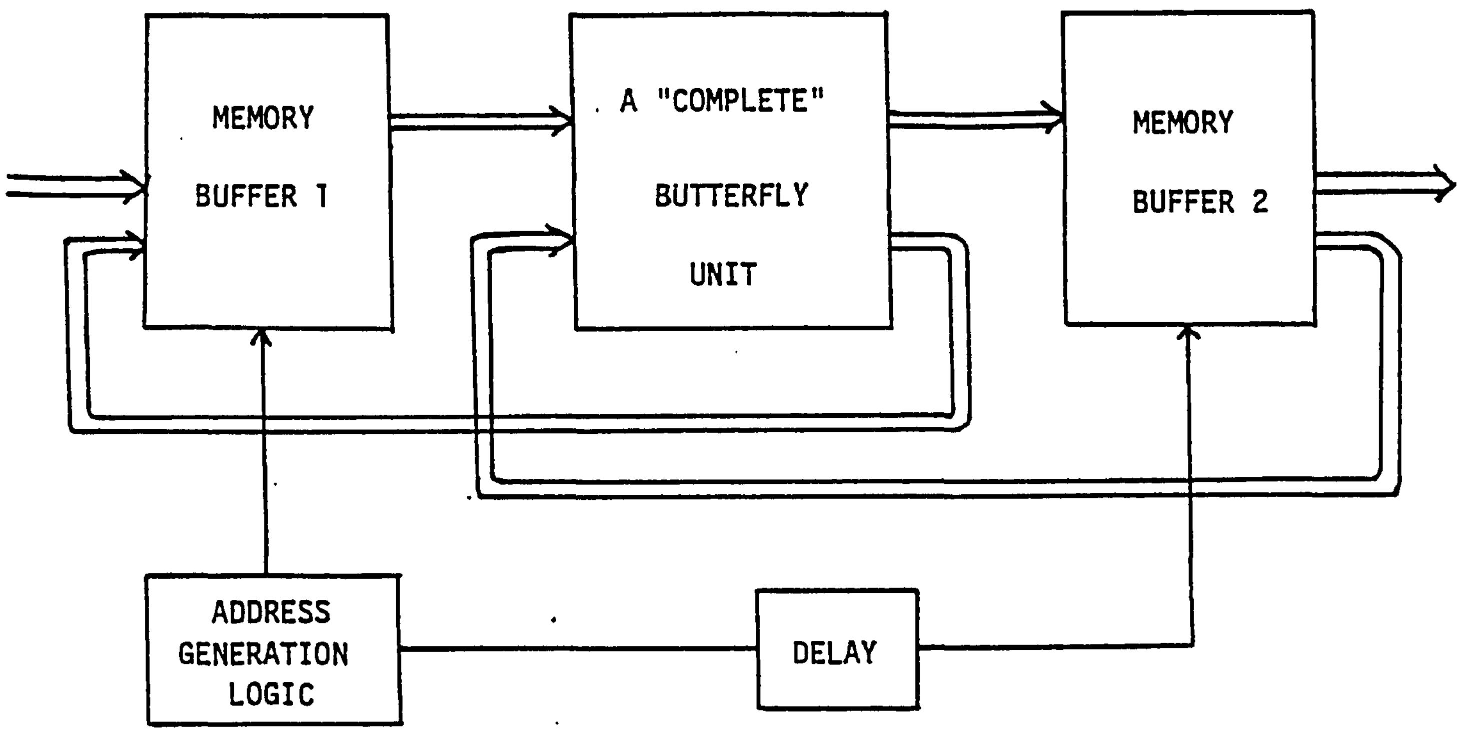
Fig. (3.8) Timing Consideration in the Use of Memory Configuration Shown in Fig. (3.7).

MEM3. At the completion of filtering of the first image, MEM3 and MEM1 are used to store the intermediate results for the second image. At the time when the processing of the second image begins, the output multiplexer gates the already processed image through the output channel. This is possible because the input and output of the NTT algorithm, as given by the OIOO-NTT algorithm, have the same addresses. At the begining of a new computation, the buffer selection is changed and during the computation of convolution, the I/O channel is utilized to perform the input and output operations.

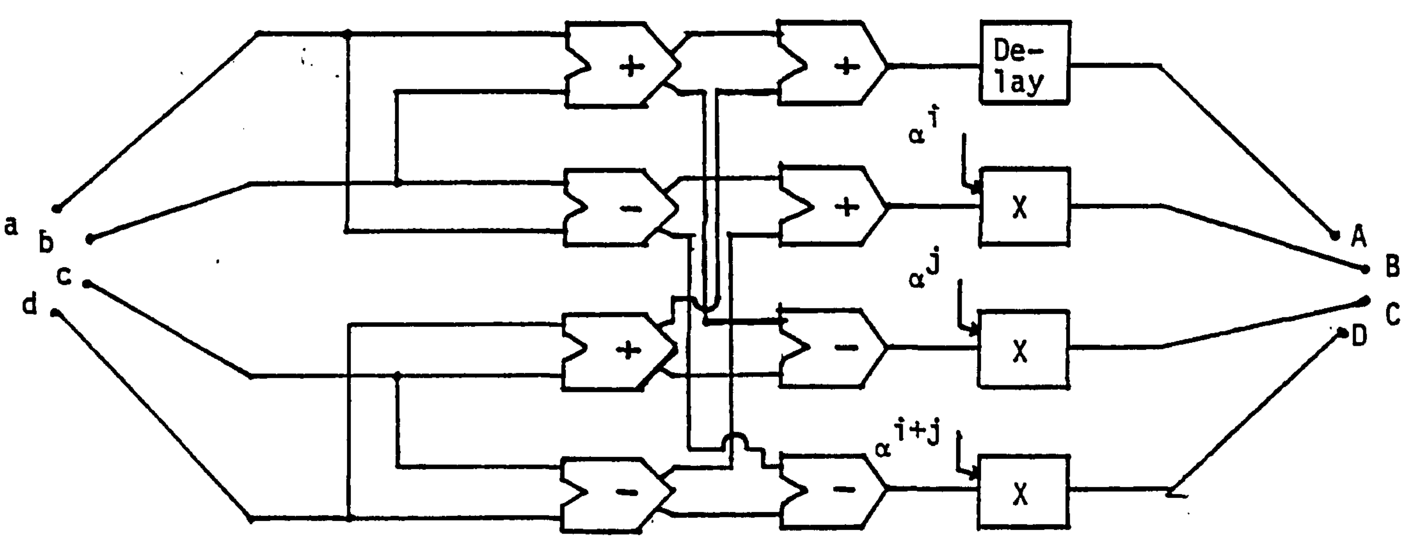
As noted earlier, the total time required for the input and output of an image (through the use of a HSD interface) is approximately 20 msec., and the actual processing time for an image of size (128x128) is 68.8 msec.. Through the use of three-memory structure outlined above, it is possible to have 100 % over-lap of I/O time with the processing time (Fig.3-8). This implies that if images are processed in succession, then the processing time can be reduced to about 48 msec.. This is an improvement in speed by 28 %.

3.6 USING A COMPLETE BUTTERFLY FOR FASTER SPEED

The convolver implements a 2-D-radix-2 butterfly (OIOO-algorithm) for the computation of transforms. In hardware, the processing is performed in a serial sequential fashion. It was observed in sec-(3.3) that we use a multi-



(a)



(b)

Fig. (3.9) Interconnections in the use of a "complete" Butterfly.

plexed "quarter" butterfly to compute the transform of the sequence. We can, instead, use a "complete" butterfly structure to obtain a four-fold increase in speed. The possible structure or a scheme to use a complete butterfly is shown in Fig-(3.9).

The implementation of a 2-D-radix-2 butterfly requires the implementation of eqn.(3.2). The (preweighting) operations of addition and subtraction can be obtained through the use of eight adder/subtracters, each with two data as input. The method will involve two levels of operation to avoid the use of 4 inputs adders. The implementation also requires 3 multipliers as shown in Fig.(3.7). This is because the multiplication by unity may not be performed. The data flows back and forth between the two memory buffers. Once the input sequence has been permuted, the addresses required for the input and the corresponding output are the same. This means that the same address generation logic circuitary can be used for the control of data flow by the use of a delay equal to the time required for processing of one data point.

This is an arrangement where there is a simultaneous processing of 4 data points together in a pipeline flow. The time of convolution by this method is given by,

$$T(conv) = 2.n \,[\, (N.N/r.r).t + D \,] \qquad (3.17)$$

where D is the delay between the input and the corresponding output and $r=2$, the radix of operation. The time required to compute convolution by this method (for an image of 128x128 pixels) is 18.45 msec.. This supports the video rate.

It is interesting to note that the hardware requirement for the butterfly increases by a factor of more than 3 (nearly 4). However, at the same time it is possible to use the rest of logic circuitary with little modifications. This implies that the total cost of the entire filter structure does not increase by a factor of 4, while the speed gain is four-fold.

3.7 A BRIEF COMPARISON OF FFT AND FNTT BUTTERFLY IMPLEMENTATIONS

A comparison of FFT and FNTT butterfly implementation involves several variables. Here we make a brief comparison between a FFT and a FNTT butterfly implementations taking into account the following variables only:

1. Speed
2. Hardware complexity
3. Accuracy

There are two distinct operations in a butterfly computation: addition (or subtraction) of the data points and multiplication by the twiddle factors. A FFT buttefly requires complex arithmetic. When the arithmeitc unit is of general purpose nature, a complex audition takes twice the time that of a real addition and, a complex multiplication takes four times the time required by a real multiplication . Use can be made of the fact that the input data are real.

This results in a saving of 50 percent if one neglects the overhead involved. Thus, the additions in the butterfly in the two implementations can be made equivalent in terms of speed. The multiplication by the twiddle factors through the use of a FNTT butterfly will be efficient on a general purpose arithmetic unit. However, the mathematical computations in the use of FNTT require mod M operations. A mod M operation on a general purpose computer is computed by performing integer division, which is a time consuming process. Thus the efficiency of a FNTT implementation over a FFT implementation is doomed.

Again, it is possible to use special purpose hardwares [19] to handle the butterfly computations. The hardware complexity increases in case of FFT butterfly because of the complex nature of the arithmetic involved in the computation. The need of binary to residue and residue to binary converters in the use of the FNTT butterfly increase the hardware requirement. Also, it is possible to use a table look-up approach to implement multiplication for a faster processing. The look-up tables entries required by the FFT butterflies are subjected to error due to the finite precision representation of the transcedental multiplier function. In general, these tables require more memory [25] than the corresponding look-up tables in the FNTT butterfly implementation.

Further, the results obtained through the use of a FNTT algorithm are exact if the dynamic range of the machine is large enough so that no number in the final result is greater than the the machine dynamic range. But in case there is an overflow, the resulting sequence does not represent the transform. Because of the error due to finite precision representation of the transcedental multiplier functions, the results obtained through the use of a FFT algorithm are approximate. Thus it is difficult to establish the superiority of one implementation over other. Several other factors, such as the type of data to be processed etc., must also be considered to compare the two implementations.

3.8 COMPARISON OF 1-D-RADIX-2 AND 2-D-RADIX-2 BUTTERFLIES

For this comparison, we will define the complexity based on the number of multiplications required in the implementation of a 1-D-radix-2 and a 2-D-radix-2 butteflies to process a matrix. The multiplications involved in the implementation of butterflies are the multiplications by the twiddle factors.

For an m-dimensional-radix-r butterfly, it was shown [10] that the total number of multiplications with twiddle factors is

$$[(r)^{m} - 1]$$

as it combines $[m.(r)^{(m-1)}]$ composite twiddle factor multiplications. If a one dimensional radix-r transform operation were performed along all m dimensions in a sequential fashion, then the number of twiddle factor multiplications would be equal to {m.[(r)**m-(r)**(m-1)]}. Thus the total number of multiplications is given by,

$$M = [(r)**m-1].(n).(M.M/r.r)$$

Where 'n' is the number of stages. Hence the ratio of multiplications (RM) between 1-D-radix-r and m-D-radix-r butterfly implementation is

$$RM = \frac{m.[(r)**m - (r)**(m-1)]}{(r)**m - 1} \qquad (3.15a)$$

and the percentage saving in computation is equal to

$$\frac{m.[(r)**m-(r)**(m-1)] - [r**m-1]}{m.[(r)**m-(r)**(m-1)]} \times 100 \ (\%)$$

(3.15b)

To compare a 1-D-radix-2 and a 2-D-radix-2, we substitue m=2 and r=2 in eqn-(3.15). We obtain

$$RM = (4/3) = 1.3333$$

and a saving in computation of 25 % is obtained, Table-(3.3).

It is interesting to note a 2-D-radix-2 structure is computationaly the same as a 1-D-radix-4 structure, which has been described as optimum [1,2].

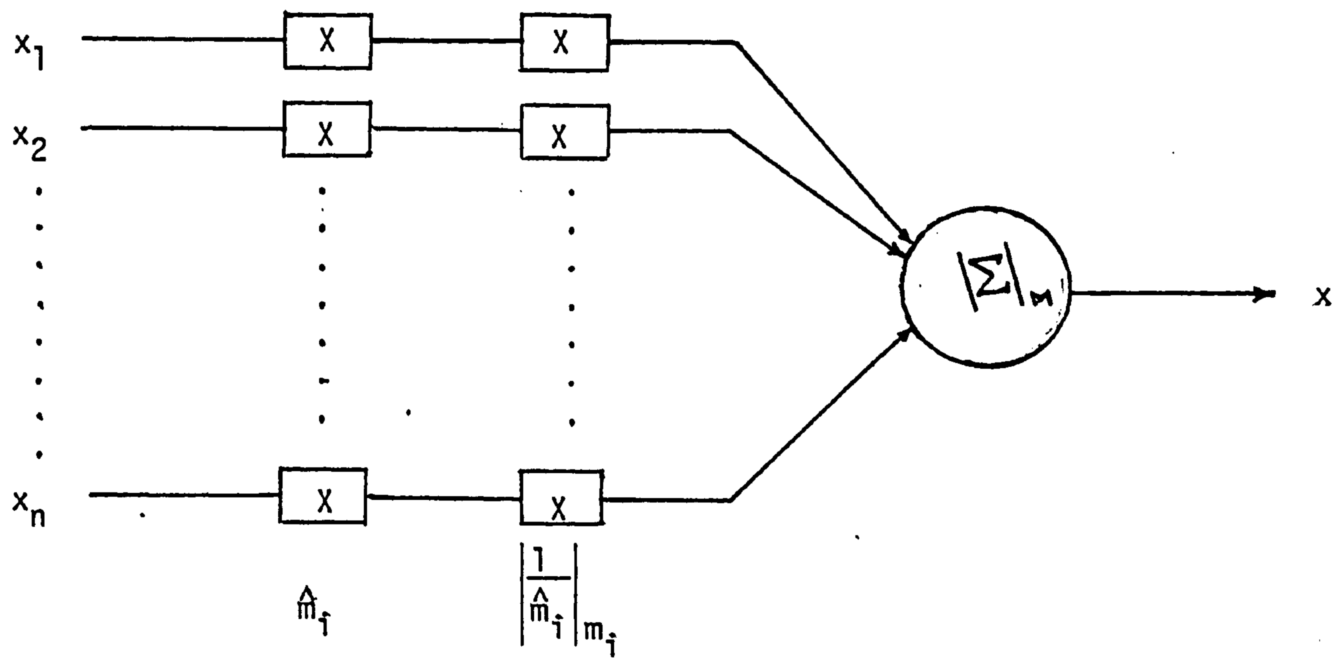


Fig. (3.10) Implementation of the Chinese Remainder Theorem.

Table-(3.3) Comparison of butterfly Computational Requirements

Method	no. of complex multiplications (in thousands)		
Pic.Size, N=	64	128	256
1-D-radix-2	24.57	114.68	524.28
2-D-radix-2	18.43	86.01	393.21

3.9 COMPARISON OF R/B CONVERSION METHODS

Residue to Binary (R/B) conversion is implemented by either the Chinese Remainder Theorem (CRT) or by the use of the Mixed Radix conversion (MRC) method, sec-(2.7). The Chinese Remainder Theorem expresses the relationships between the number and its RNS representation as,

$$x = \left| \sum x_i m_i \left| \frac{1}{\hat{m}_i} \right|_{m_i} \right|_M$$

where xi are the residues (mod pi) and $M = \prod_i pi$. On the other hand, the Mixed-Radix conversion method requires two steps. In the first step, the residues are transferred to Mixed-Radix (weighted) digits. The second step converts these digits into a fixed radix form (e.g. binary). Repeating the relations from sec-(2.7),

$$\langle rn,\ldots,r3,r2,r1\rangle = x = \langle an,\ldots,a3,a2,a1\rangle$$

$$= an.\prod_{i=1}^{n-1} pi + \ldots + a3.p2.p1 + a2.p1 + a1$$

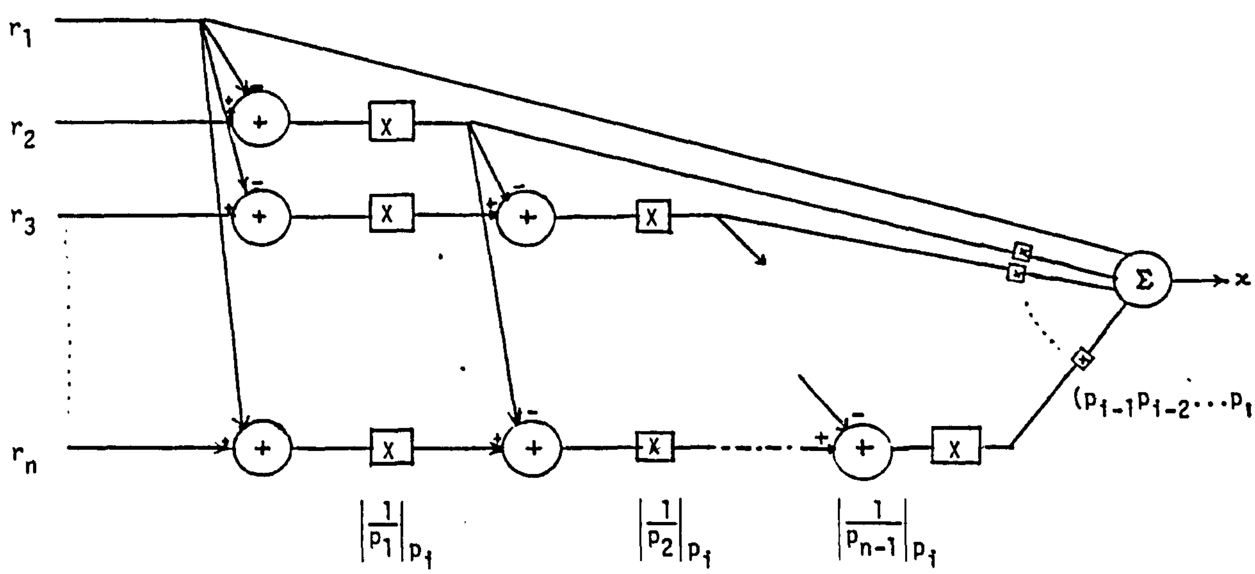


Fig. (3.11) Implementation of Mixed-Radix Conversion Method.

(3.16)

where ri's are the residues (mod pi), ai's are the mixed digits and pi's are the prime moduli. The block diagrams for the methods are shown Fig-(3.10) and Fig-(3.11). The number of computational elements required by the two methods are can be compared. Let a(n) represent the number of adder/subtracters, and b(n) represent the number of multipliers. Then

1. For a(n) 'n' moduli representation, the CRT requires 2n multipliers and an adder (mod M).
2. For a(n) 'n' moduli representation, the MRC requires

$$a(n) = (n-1).(n/2)+1 \quad \text{adder/subtracters}$$

$$b(n) = b(n-1) + b(n-2) \quad \text{multipliers}$$

where b(i) is an operator, b(0)=1 and b(1)=1 and n > 1. The requirement that a modulo M adder be used in the CRT algorithm, restricts its use in hardware implementations. Table-(3.4) shows the requirements of adders and multipliers for various values of 'n'.

Table-(3.4) Comparison of CRT and MRC implementation
(a) Number of adders (b) Number of multipliers

	no. of moduli							
	n=2		n=3		n=4		n=8	
Method	(a)	(b)	(a)	(b)	(a)	(b)	(a)	(b)
Chinese Rem.Theo.	1 mod M	4 (2)	1 modM	6 (3)	1 mod M	8 (4)	1 modM	16 (8)
Mixed Radix-Conv	2	2 (1)	4	5 (3)	7	9 (6)	29	35 (28)

It will be noticed that for small n $(n<4)$, it is beneficial to use the Mixed-Radix-conversion method to compute the results of Residue to Binary operation. Also, it is possible to combine various fixed arithmetic operation for a particular implementation. The requirements after such combination is indicated by putting the numbers in brackets in the table.

3.10 CONCLUSION

In this chapter we have discussed various aspects of the convolver. The main focus was the butterfly unit. First we discussed the efficient implementation of butterfly and then we evaluated the performance of the filter with respect to speed, memory requirements and computation cost. This was followed by a comparison between the implemented filter and other techniques of digital filtering, in terms of computational efficiency. The timing considerations in the pipe-line structure was evaluated, and the through-put rate determined. Two design improvements for faster processing were suggested. Finally, the two methods of implementation of Residue to Binary conversion were compared.

Chapter IV

HARDWARE AND FUNCTIONAL DETAILS OF THE CONVOLVER

4.1 INTRODUCTION

For several problems in Image Processing, linear filtering of images prior to the applications of other algorithm is extremely important. Filtering of discrete signals can be implemented on a general purpose digital computer of reasonable size. This is a time consuming process, and when speed of filtering is more important a hardware implementation is the only viable solution. The speed and cost considerations are basic to filter hardware design and a compromise has to be achieved between the two depending on the particular application in mind. For example, the speed of filtering is the main consideration when real-time processing of signals is desired [4].

In this chapter we describe the hardware and functional details of the convolver. It is suggested that the user of the convolver refers to reference [29] for specific details about the hardware circuitary.

4.2 OVERVIEW OF THE HARDWARE

The Convolver hardware has been assembled on four Augat Wire Wrap boards. One of the boards contains the interface betweem the HSD and the convolver, and the control logic; two boards support the memory buffers and the 2-D-radix-2 Butterfly unit; and the fourth board has the Residue to Binary (R/D) conversion unit. A conceptual diagram of the Convolver organization is given in Fig-(4.1). The main components of the hardware are described in detail in the following sections. An overview is presented here.

The convolver implements a 2-D-radix-2 Fast Number Theoretic Transform (FNTT) using an Ordered-Input-Ordered-Output algorithm [6] where the multiplications in the butterfly are performed using the sub-modular approach [5]. The host computer sends the image input and the indices of the transformed coefficients and the necessary control signals to start the filtering operation. The data stored in the memory buffers are processed with the butterfly unit for a 7-stage sequential implementation. When the input to the butterfly is from the memory buffer-1 then the output is written to the memory buffer-2 and vice versa. At the final stage of the forward transform, the output is multiplied by the transform of the filter coefficients using the twiddle factor multiplier. At the end of the inverse NTT stage the data is processed by a Residue to Binary converter, which then transmits the data back to the host computer.

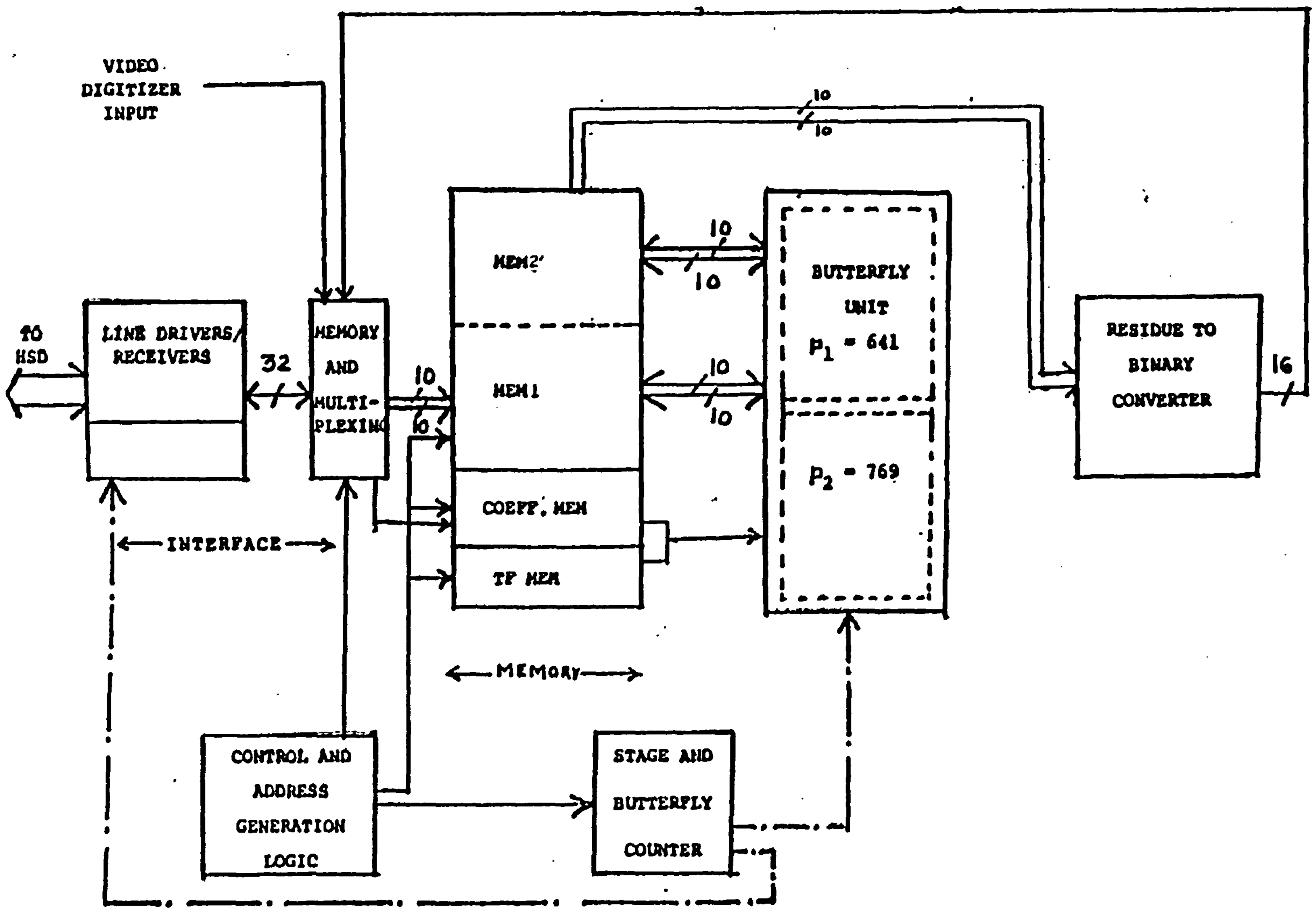


Fig. (4.1) Overview of the Convolver.

TABLE (4.1)

THE IMAGE PROCESSOR (BASIC BLOCK)

INPUT DATA ORGANIZATION:	128 x 128 (BYTES) FOR IMAGE 128 x 128 x 24 BITS FOR COEFFS.
MODULI:	641, 769
ALGORITHM FOR BUTTERFLY:	2D-RADIX-2 0 I 0 0 - NTT ALGORITHM DECIMATION IN FREQUENCY
OUTPUT:	128 x 128 x 16 BIT-BLOCK OF FILTERED SECTION OR IMAGE
TIMING:	BASIC CLOCK 300 ns I/O TIME = 5 msec + 10 msec = 15 msec. BUTTERFLY COMPUTATION TIME = 68.8 msec TOTAL PROCESSING TIME = 85.0 msec
TECHNOLOGY:	MSI and SSI Standard TTL and MOS Static Memory, EPROMs
PHYSICAL:	PACKAGED ON 4-AUGAT WIRE WRAP BOARDS 9 x 16 x 1.5 INCH POWER CONSUMPTION: 5V DC

TABLE 4.2 Main Integrated Circuits Used in the Convolver

IC TYPE	MANUFACTURER	FUNCTION/USAGE
74S124	Texas Instrument (TI)	Dual Voltage Controlled Oscillator (1 Hz-60MHz); Generation of Clock
74S138	TI	Decoder/Demultiplexer (1 of 8); Stage Decoder
74S139	TI	2 to 4 line Decoders/Demultiplexer; Stage Decoder
75138	TI	Quadruple Bus Trans-receiver (8-line); Trans-receiver driver.
74S74	TI	Dual D-Type Positive Edge Triggered Flip-Flop; Delay
74S175	TI	Quadruple D-Type Flip-flop with clear delay.
LM311	National	Voltage Comparator; In control of circuitary for Interface; Reset Logic.
74S163	TI	Synchronous 4-bit counter; Butterfly and stage counter
74S253	TI	Dual 4 to line Data Selectors/ Multiplexer with 3-state output; In control circuitary for address generation.
74S257	TI	Quadruple 2 line to 1 line Data Selectors; Multiplexing of Data at output of residue to binary converter.
74S163	TI	4-bit Sync. counter; Memory address generation for data input
74S283	TI	4-bit Full Adder with Fast carry; Adder-subtractor

IC TYPE	MANUFACTURER	FUNCTION/USAGE
2118	Intel	RAM Memory (16384 x 1 bit) 16KX1; Memory Buffers; Coefficient Memory
AM2964B	AMD	Dynamic Memory Controller (for 16 K and 64K MOS RAM) R/W address control
AM 2966	AMD	Octal Dynamic Memory Drivers with 3-state output; Tristate buffer for memory
2732A	Intel	EPROM (4Kx8 bit) Twiddle Mem/Look-up tables
AM2517	AMD	Arithmatic Logic Unit Butterfly - ALU, Function Generator A±B, A+B, A·B, etc.
9319	Fairchild	Decode Sequencer (1 of 10 sequential output) Generation of different clock from the main clock
96S02	TI	Dual retriggerable monostable multivibrator; In correct sequencing.

The basic block for filtering by this method is an image of (128x128) bytes. The various components of the filter are described below. The main features of the filter are summerised in Table-(4.1). The main Integrated Circuits (IC's) used in the convolver and their functions are explained in Table-(4.2).

4.3 SYSTEM CLOCKS AND RESET LOGIC

The filter design is based on a synchronous architecture and is driven by the system master clock. A dual voltage controlled oscillator IC-74S124 has been used to generate a 20-MHz clock which is divided by the Decade Sequencer IC-9319 to generate six non-overlapping clocks with a period of 300 nsec. These clocks are inverted to generate two sets of system clocks: One set of the system clocks is used by the Interface Board and the other set is used to generate memory and pipe-lining timing signals.

The Voltage Comparator IC-LM311, driving a Mono-stable circuit has been used to generate a power-on reset pulse. This pulse is logically OR-ed with the I/O-reset pulse [selior] from the HSD to generate three different Resets [filreset, filreset1, filreset2] which are used to clear various registers and to initialize the interface logic.

4.4 HSD/CONVOLVER INTERFACE

The HSD/convolver interface is an essential component of the convolver as well as the Image Processing system. The High Speed Device (HSD) interface logic links the HSD handler (in SEL-32/27 computer) and the filter. The HSD interface logic is very general in nature and apart from convolver, devices such as the video digitizer can use the same HSD-interface logic. The HSD-interface (Fig-4.2) has the following main components:

1. Line Drivers/Receivers
2. Multiplexers (Accumulators)
3. I/O data counter (NTT stage/ butterfly counter)
4. control logic for I/O operations
5. Control logic for data input from Video-digitizer

The interface logic is controlled by the signals from the HSD handler. The Filter function register is loaded from the HSD to specify the next filter function (sec-4.5). When an I/O filter-function is specified the data transfers at a rate of 1.2 us (1200 nsec) per 32-bit word. Four pixels are transferred at a time from the SEL to the convolver as input data, and these are demultiplexed at the basic system clock rate of 300 nsec before writing the pixels byte into the memory buffer. This ensures a continuous transfer of data. The I/O data counter keeps the record of the number of data transfer and once the specified number of data-transfer has taken place, it resets and is then available for other counting functions.

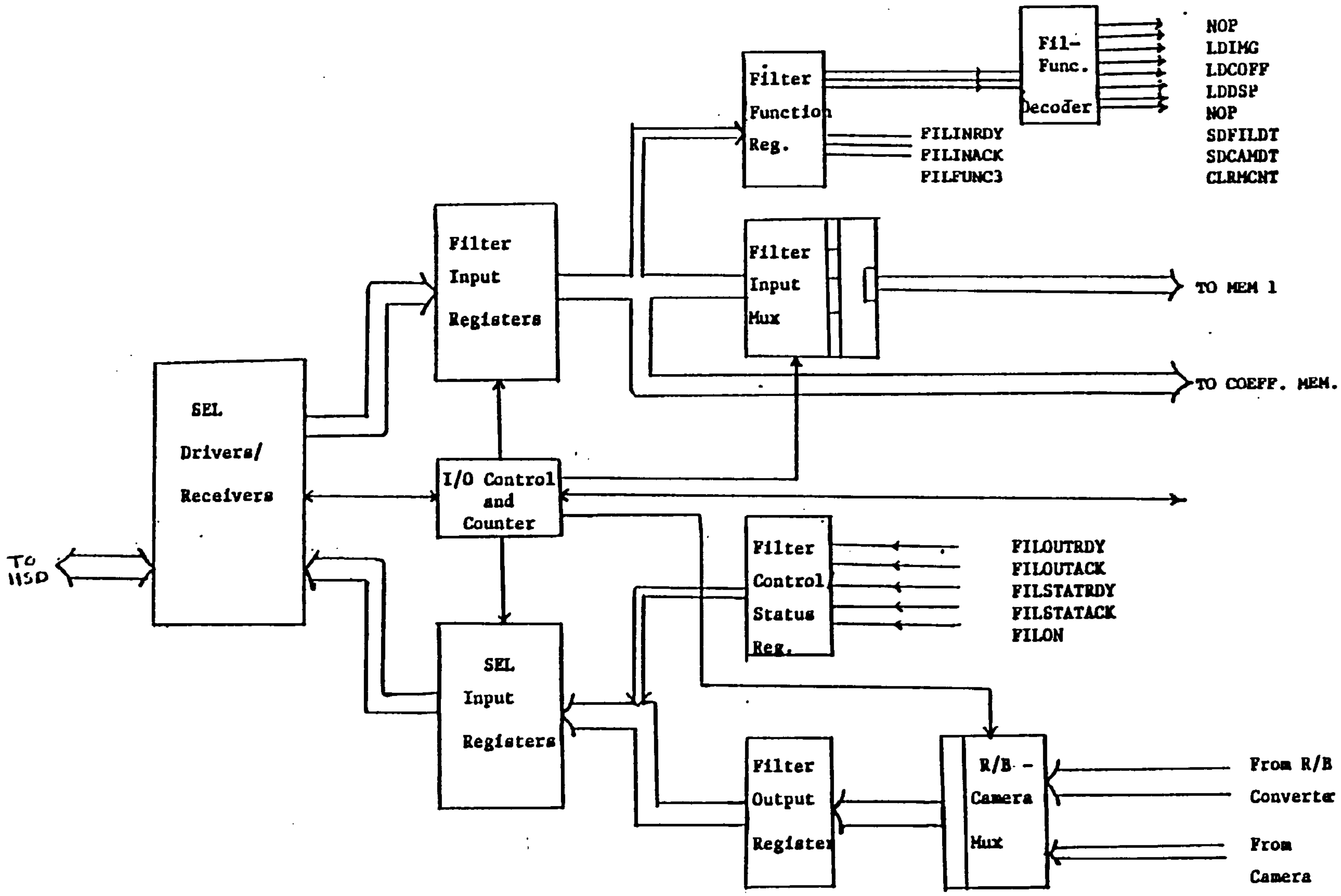


Fig. (4.2) The Filter Interface and I/O Control Unit.

4.4.1 Line Drivers/Receivers

The Line Drivers/Receivers consist of 12 TI-75138 IC's and are connected to the HSD via two 50 pin flat cables. The driver part is controlled by the strobe signal [selinstb], and is enabled whenever the data-status is to be sent to the HSD.

4.4.2 NTT stage/ butterfly counter

The NTT Stage/ Buttefly Counter (NSBC) serves a dual purpose. During the data-transfer between convolver and HSD, it counts the number of pixels and during the convolution operation it serves as a stage and butterfly counter. This is also referred to as the I/O Data Counter.

This 18-bit NSBC is a synchronous counter and is made up of five IC-74S163. During an I/O transfer this counter is controlled by the interface logic and indicates when the specified number of data have transferred. When the filter is set to perform the convolution, i.e., when [filon=1], the butterfly counter is used to generate the count for the 14-stages (7 for forward and 7 for inverse) a of 128x128 transform.

When used as the NSBC, it generates a 6-bit column address, a 6-bit row address and a 2-bit word address. These address bits are used to compute the Read/Write addresses of data items from and to MEM1 and MEM2 for the current butterfly operation. The remaining 4-bits are used for the stage count (1-14).

The NSBC forms a modulo (2**14+9) counter, and generates the addresses in a serial fashion such that data are always (N/r**2) apart as required by the OIOO-NTT algorithm. The additional count of 9 is required to complete the Read/Write operation at each stage. Out of the 4-bits used for the stage count the MSB determines whether the forward (0) or the inverse (1) operation is being performed. At the end of the filtering operation, a signal [clrrunff] is generated to clear the [filstart] flip-flop and to terminate filtering [filon=0]. At this time the filtered image is available in MEM1,to be transferred to the SEL via the HSD.

During the transfer of either Image data or Coefficient data between the HSD and the convolver, the counter generates the address of MEM1/ TCOFMEM to store or retrieve the data from MEM1/TCOFMEM. This process proceeds in a sequential fashion and at the end of the data transfer of (128x128) points a signal [filindatacomp] is generated which inhibits further transfer operations between the HSD and the convolver. The NBSC is cleared at the begining of each transfer operation from the signal supplied by HSD.

4.5 INTERFACE CONTROL LOGIC

The interface control logic generates the signals required for data transfer between the HSD and the convolver. Two kinds of signals are recognized: FILFUNCRDY, FILFUNCACK, FILINRDY and FILINACK control the data transfer from the HSD to convolver while FILOUTRDY, FILOUTACK, FILSTATRDY and FILSTATACK control the transfer of data from the convolver to HSD.

Whenever a new filter function is specified, the [filfuncrdy] goes high and with the next [mclk50] the six-bit function is loaded in the filter-function register. Three of the six bits specify one of eight possible functions:

1. NOP (000) No Operation
2. LDIMG (001) Load Image data in (Buffer 1)
3. LDCOF (010) Load transfer of filter coeff in (Tcoeff Mem)
4. LDDSP (011) Load memory for displaying image
5. SDFILDT (100) Send filtered image to HSD
6. SDCAMDT (101) Send digitized image from camera to HSD
7. CLRMCNT (110) Clear butterfly Counter and send Filter status to HSD
8. NOP (111) left for future use

The fourth bit [filfunc3] determines whether or not the convolution process has been requested and, at the end of the filtering operation, clears [filstart] at which a new filter-function may be loaded to the fil-func-register. There is no provison for external Interrupts, and once the filtering begins, there is no data transfer between the HSD and the convolver until the end of the filtering operation.

This ,however, does not affect LDDSP and SDCAMDT since the operations they control can be performed independently.

4.5.1 LDIMG operation

When LDIMG is true, the sequential loading of the image data is enabled. After the [filinrdy] signal is active, and if no previous function is being served, then a 40 nsec pulse is generated which enables asynchronous data transfer. The data from the SEL-bus are transfered to the filin-registers (32-bit) and from there the data is demultiplexed and written into MEM1. The counter starts counting at the begining of the data transfer and is sychronised with the rate of data transfer. The demultiplexing and writing into memory is completed before the next data arrives. The 4-byte data is always written to memory in the sequence Most Significant Byte to Least Significant Byte (bits 31-24 to bits 07-00). The signal [filinack] is sent to the HSD to acknowledge the completion ofdata transfer.

4.5.2 LDCOF operation

When LDCOF is true, the least significant 24-bits of the data are transfered in the same fashion as above for LDIMG. The 24-bit word is formed by reducing the indices of the transform of coefficients corresponding to the set of sub-moduli 62 and 63, as indicated in sec-(4.6.6).

4.5.3 CLRMCNT operation

This is a special type of function performed by the filter to inform the HSD status of the filter. Whenever status information is requested the CLRMCNT signal is made true by sending the signals from the HSD. The filter then transmits a 5-bit data pack on 32-bit lines indicating FILON, NTTR,ISTAGR0,ISTAGR1,ISTAGR2. This informs the HSD as to whether or not an I/O operation is to performed.

4.5.4 SDFILDT operation

When SDFILDT is active low, the data from the MEM1 is sent to the HSD. The filtered output for each data point is a 16-bit word and is multiplexed to form a 32-bit word for fast I/O operation. The counter keeps track of the number of pixels sent to the HSD.

4.5.5 SDCAMDT operation

This function implies a transfer of data from the filter to the HSD when the data is obtained from a camera (Video-digitizer). The camera output is a 8-bit word. The camera multiplexer [rocammux] multiplexes four 8-bit outputs to form a 32-bit word for fast I/O operation through the HSD.

4.6 MEMORY BUFFERS

The block diagram of the memory buffer arrangement used in the convolver is shown in Fig-(4.3). The figure shows the main memory buffers (MEM1 and MEM2), The Twiddle Factor Memory (TFMEM) EPROMs, the transform of the coefficients memory (TCOFMEM), and the associated logic for the generation of Read/write addresses.

1. MEM1 and MEM2 are organized in 16Kx20 bits each and store the input data and the intermediate results of the butterfly operations.
2. TFMEM EPROMs store the indices of 128 residues of (δ**k) for k=0,1,2,...... 127 corresponding to the set of sub-moduli, 62 and 63.
3. TCOFMEM is organized as 16Kx24 bit and stores the 128x128 indices of NTT of the given filter kernel corresponding to the set of sub-moduli 62 and 63.

4.6.1 Memory Write Address Generator

The output of the Butterfly counter is also used to generate addresses for Read/Write operation from and to MEM1, MEM2 and TCOFMEM. Because of the pipelining structure of the Butterfly unit, there is delay of 9 pulses between the input and the corresponding output. This means that output of the butterfly counter must subtract 9 to generate correct memory addresses. As required by the OICO-NTT algorithm, data are always written in a block of 64x64 with a

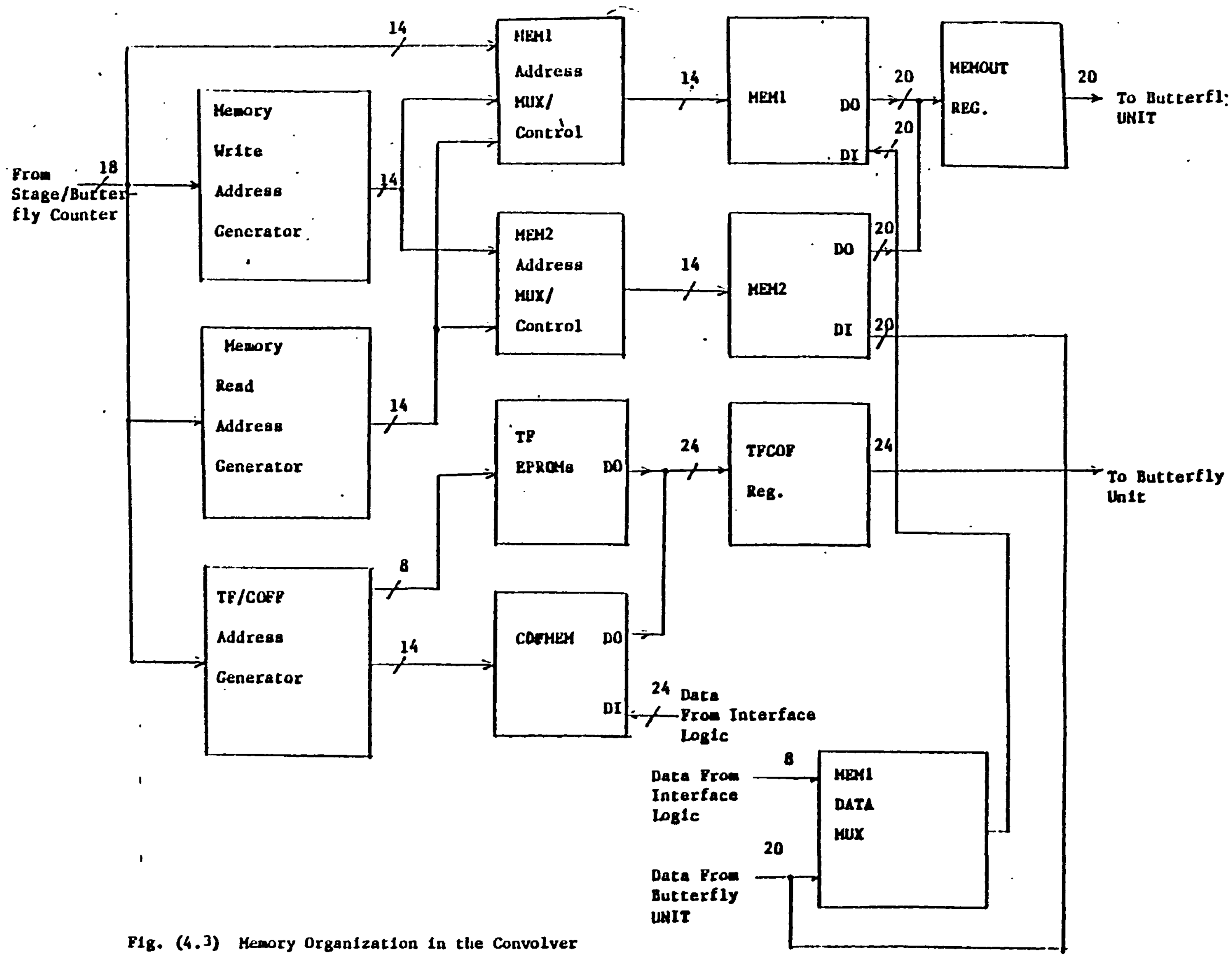


Fig. (4.3) Memory Organization in the Convolver configuration [{(0,0),(0,64),(64,0),(64,64)}, {(0,1),(0,65),(64,1),(64,65)}.......] and this configuration is obtained by using the signals [IROW5-IROW0,ICOL5-ICOL0,IWORD1-IWORD0].

4.6.2 Memory Read Address Generator

The addresses of the data points participating in a butterfly computation depends on the block address within the image (there are 4 blocks) and a sub-block address within the specified block. The stage-decoder and the butterfly counter are used to control the Memory-address generator. The [mem-ad-gen] multiplexes the output of the Butterfly Counter in such a way that a correct memory read address is generated during each stage of the NTT and the INTT operations.

4.6.3 Twiddle Factor/Trans. of Filter Coeff. Addr. Genrator

The TF/TFCOF Address generator generates three types of addresses:

1. TCOFMEM address required for storing the NTT of the filter coefficients into TCOFMEM
2. Address required for accessing the twiddle factors from the memory during the NTT and the INTT operations
3. Address required for accessing the transform of the filter coefficients from TCOFMEM during the final stage of the forward NTT computation

Due to the pipeline implementation of the Butterfly unit, a delay of 6 clock cycles between the output of the counter and the output required by the address generator for TCOFMEM must be introduced. Also, due to the the access time of the TF EPROMS, the TF address must be generated one cycle ahead of the TCOFMEM address. The transform of the coefficients is pre-shuffled and is loaded in TCOFMEM in such a fashion that they can be addressed sequentially for multiplications by the transform of the image.

The OIOO-NTT algorithm requires three different values of the Twiddle factors for each butterfly operation depending on the row index i, the column index j, and the sum (i+j) and the stage of the butterfly. These indices are generated by masking the output of the stage decoder and are then multiplexed to generate the correct addresses for the Twiddle Factors.

4.6.4 Memory Address Multiplexer

While MEM1 stores the input image, the filtered output and the intermediate stage results, MEM2 only stores the results from intermediate stages. The Memory-add-multiplexer selects the memory buffer and the operation (Read or Write) itself.

4.6.5 Memory Buffer MEM1 and MEM2

The memory buffers MEM1 and MEM2 are organized as 16Kx20 bits where bits (19-10) are reserved for residues corressponding to the modulus 641 and bits (9-0) are reserved for residues corresponding to the modulus 769. The Dynamic Memory Controller (DMC1) generates the row/column addresses for MEM1 and maintains the proper timing signal during read/write and refresh operations. Since only one of the two memories, MEM1 or MEM2, gives the output at any time, the two outputs are wire ORed to [mem-req]. The input data to MEM1 is written byte by byte through the [filin-mux] from the HSD at the begining of the convolution process. MEM2 is controlled by DMC2 and the necessary addresses for this buffer are generated by the [mem-add-mux].

4.6.6 Memory Buffer TCOFMEM

The memory buffer TCOFMEM is organized as 16Kx24 bits and stores the indices of the residues of the NTT of the coefficients. The indices of each residue are computed with respect to sub-moduli 62 and 63 in software and combined to form a 24-bit word as shown below:

The TCOFMEM is also controlled by the dynamic memory controller. The data input to this buffer is through [filin-mux] from the HSD. The output of this buffer is connected to the TFCOF Register and is wire ORed with the TF EPROMS, since only one of the two is enabled at any time.

residue of 769		residue of 641	
23 18	17 12	11 6	5 0
index with respect to 63	index w.r.t. 62	index w.r.t. 63	index w.r.t. 62

4.6.7 Twiddle Factor EPROMs

The TF EPROMs store the indices of the powers of the generator for the set of moduli 641 and 769. The format for storing these indices is same as that for indices in TCOFMEM. The EPROMs used are IC-2732A. The addresses 0-127 and 128-255 store the indices corresponding to the various values of δ**k and δ**-k respectively.

4.7 THE BUTTERFLY UNIT

The Butterfly Unit (BU), Fig-(4.4), consists of binary adders/substracters (Am2517), Look-up tables (EPROMs), and standard 8-bit Schottky TTL registers. The transform operations, corresponding to moduli 641 and 769, are performed in parallel. The residue representation and the operations of addition and subtraction are performed in 2's complement binary. Thus, a residue between {0, (pi-1)/2} has the same representation as binary and the values between {(pi+1)/2} and {(pi-1)} are represented as (2**10-Xi) where Xi is the residue.

As discussed in sec-(3.2), the implementation of the butterfly requires the implementation of the following equations:

A(i,i)= [a(i,i)+a(i,j)+a(j,i)+a(j,j)]

A(i,j)= [a(i,i)-a(i,j)+a(j,i)-a(j,j)].(δ**m)

A(j,i)= [a(i,i)+a(i,j)-a(j,i)-a(j,j)].(δ**n)

A(j,j)= [a(i,i)-a(i,j)-a(j,i)+a(j,j)].{δ**(m+n)}

i=0,1,2,...63; j=64,65,...127

where m and n depend on the stage and the location of the butterfly being performed.

The butterfly operation is performed in 8 stages and these can be reduced to three basic steps:

1. Pipeline stage 1 and 2

   Stage one (register R11) is used to buffer the input to the Ist adder/subtracter and stage two (register R12) delays the first data by a clock pulse while the second data is stored in register R22. The result of addition/substraction is computed (11-bits) and is stored into 3rd stage registers R13 and R23.

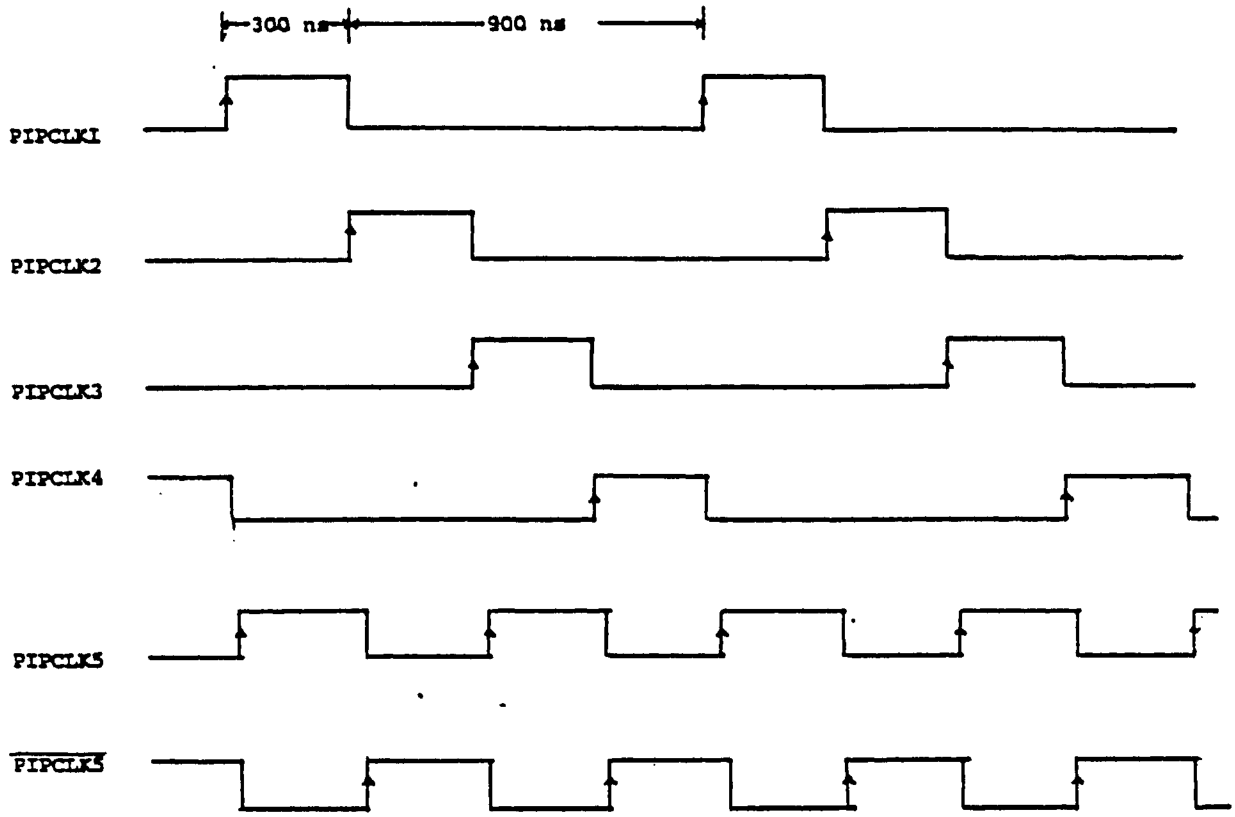
2. Pipeline stage 3, 4 and 5

   These pipeline stages provide buffering of the data between the Ist and 2nd adder/subtractor. Stage 3 and 4 store the result of the pair of additions and subtractions. The registers (R14-R44) in stage 4 act as buffer and the desired output is obtained by connecting the inputs at this stage to two inputs of the 2nd adder/subtractor unit. The output of these operations is obtained in stage-5 register (R15) which, in turn, feeds stage 6.

3. Pipeline stage 6,7 and 8

These stages form the main computation unit (multiplier) using the submodular look up approach. In stage 6, the EPROMS perform the error correction and compute the index corresponding to the sub-moduli 62 and 63. The data for these EPROMs have been generated using the software on the SEL -32/27.These indices are added to the indices of either the Twiddle Factors or of the Transform of the coefficients in stage 7 through the use of Index-add-look-up tables. The output of these EPROMs is stored in registers (R17 and R27) at stage 7 and is fed to stage 8 to obtain the reconstruction through the use of the Reconstruction/correction table. The EPROMs at these stages have been programmed using the software programs (ECILUT,TABLE, and RECON1).

At the rising edge of the MCLK00 the BU output becomes available at stage-8 register (R18) which is then stored in either MEM1 or MEM2.



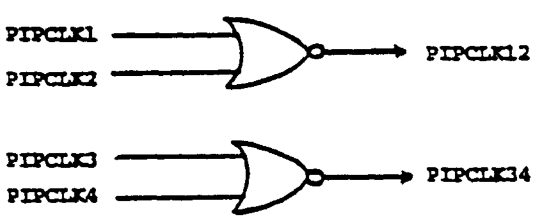


Fig. (4.5) The Pipeline Timing Diagram

4.7.1 Pipeline Timing

The basic timing signals and the pipeline clocks are derived from the system clock MCLK00. The MCLK00 is buffered and the delayed clocks are obtained as MCLK010 through MCLK050. These clocks are used to stobe the data into pipeline stages 8,7,6, and 5. The MCLK050 is used to generate 4 non-overlapping clocks PIPCLK1,PIPCLK2,PIPCLK3 and PIPCLK4 having a period of 1200 nsec through the use of a decade sequencer and delay units. The PIPCLK's are also used to generate two other clocks PIPCLK5 and $\overline{\text{PIPCLK5}}$ which are used to select different pipeline registers in stages 3 and 4, and also to control the 1st and 2nd adder/subtractor stages. A timing diagram of these clocks is shown in Fig.-(4.5).

4.8 RESIDUE TO BINARY CONVERTER

The output obtained from the butterfly unit is in the RNS representation (mod 641 and mod 769) with each residue represented in 2's complement form. A Residue to Binary (R/B) converter, Fig-(4.6), is used to obtain the final results. It utilizes the Mixed-Radix conversion method [3] for R/B conversion (sec-2.7). The design uses a pipelining approach such that the final result can be transferred to the HSD at the maximum possible rate (1.2 usec/32-bit word). The adders/ subtractors used are binary adders and the fixed multiplication is performed via look-up tables. The output of the R/B is a 16-bit number.

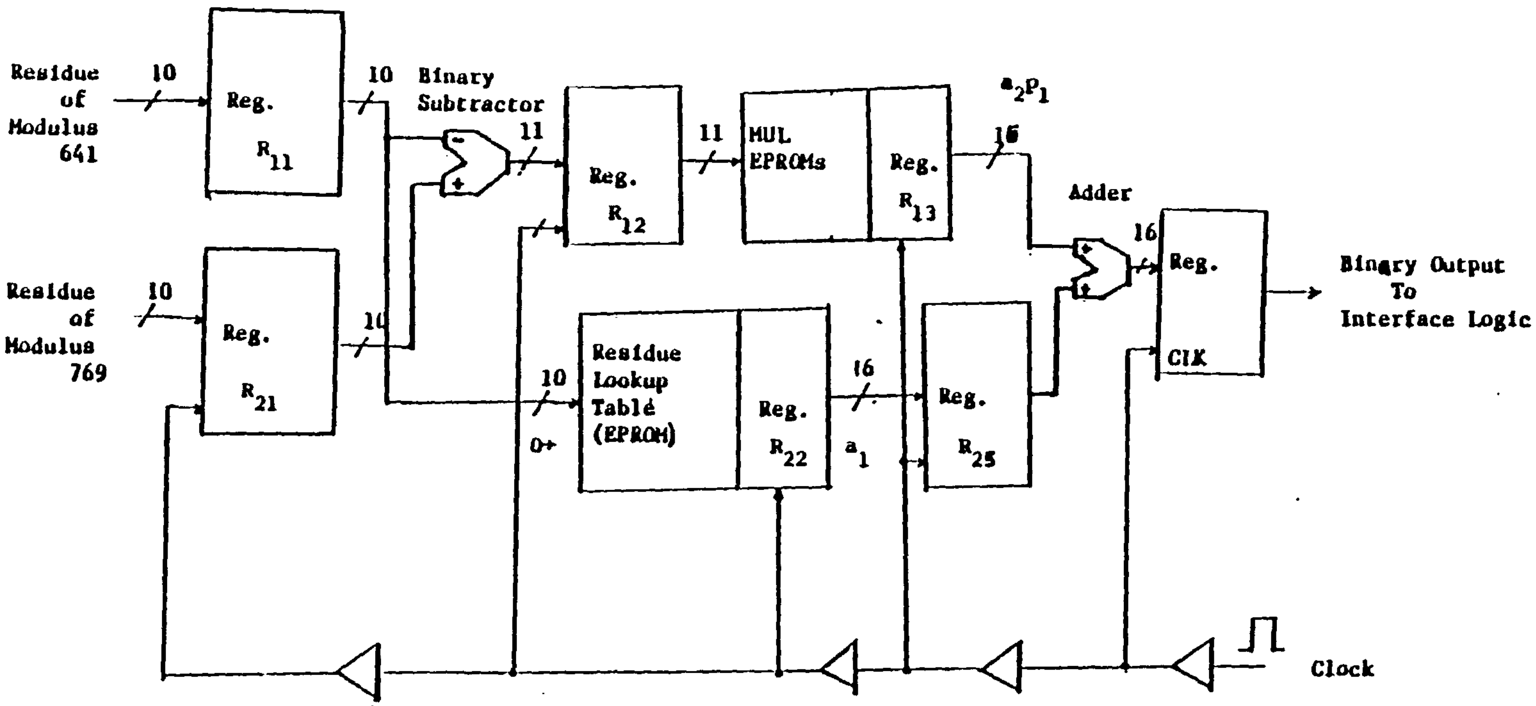
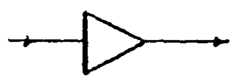


Fig. (4.6) Residue to Binary Convertor



10 ns delay element

The R/B converter unit is made up of 4 pipeline stages and it receives input from the butterfly unit output stored in MEM1. The pipeline timing pulses are generated from the delayed clock MCLK00. The stage 1 registers (R11,R21) are used to buffer the output of MEM1. In stage 2, the residue mod 641 is substracted from the residue mod 769. The result is then stored in stage 2 register (R12). Also an EPROM is used to convert the 2's comlement representation of the residue, a1, for modulus 641 into a positive binary representation (16-bit). The result is stored in register (R22). In stage 3, a look-up table is used to convert the 2's complement output of the previous stages into a positive residue with respect to the modulus 769. This residue, a2, is then multiplied by 641 to generate a 19-bit product. The most significant 15-bits are taken as the result (in register R13). These stage 3 steps are combined together and the result is obtained from the EPROMs. The EPROMs are programmed using the software program RSBCN on the SEL computer. Also, the content of register R22 is transferred to the register R23 in stage 3.

Stage 4 of the R/B converter computes the value of (a2*p1+a1) by adding the output of stage 3 via the adders. The addition is performed by adding the 16-bit output for a1 to the 15-bit output from a2*p1. The result is then sent to [rbcam-mux] to multiplex two consecutive 16-bit words into a 32-bit word and is finally sent to the HSD. The final result is a positive number in the range {0, 769*641/4} and a correction is applied in software to obtain the true 2's complement representation.

4.9 USING THE CONVOLVER

We have described the hardware and functional details of the convolver in this chapter. This section describes the steps in using the filter. The explaination of the various software available is given in Appendix-(C)&(D).The following steps must be carried out:

1. Design a two-dimensional Finite Impulse Response (FIR) filter ,with a relatively smaller kernel size than the dimensions of the image to be filtered. [It was mentioned in sec-(2.2) that to avoid the wrap-around error, the transform length must be chosen to be $M \geqslant N+L-1$, where N and L are dimensions of the input sequence and the filter kernel respectively. In the use of the convolver, which performs a 128-point transform, the wrap-around error can be completely eliminated if and only if, the sum of non-zero sequence lengths at the input is less than or equal to 129. However, if the use of the convolver is intended on an image of size (128x128), then it is advantageous to have a filter kernel as small as possible (to have smaller wrap-around error). Alternatively, the actual image must be reduced to a smaller size by

substituting appropiate number of rows and columns by zeros.] Store these coefficients in a file. Let us call this file as COEFF. Note that the available software is written for an image size of (128x128) and an filter size of (17x17). The software can be modified to include other sizes of the filter kernel.

2. Use the program SCLCOF to achieve a proper scaling factor and multiply the coefficients with this scaling factor so that the NTT of the coefficients is represented by 12-bits or less.
3. Use the program NTCOEF to find the NTT of the coefficients using the moduli 62 and 63 and store the output in a file, say NTTCOF.
4. Use the program CONFIL to do the following:
   a. Transmit the image to the filter
   b. Transmit the NTT of coefficients to the filter
   c. Receive the filtered image after processing
5. Use the program DSPSIMG to display the original and the filtered image on the Aydin Graphic terminal.

We have combined all these steps in a single Command File "FILTER" and a new user can simply type the following on the console (when in FILTER directory) to use the convolver:

```
TSM> FILTER IMAGE COEFF OUTPUT  <cr>
```

where IMAGE is the name of the image to be filtered, COEFF is the coefficient-file name, and OUTPUT is the output file name in which the filtered image is stored. The processing takes place as described above. The software referred above is available in directory 'FILTER' on the SEL hard-disk, and the hardcopy of the programs is available in [29].

Chapter V

FILTERING APPLICATIONS AND FILTERING OF LARGE MATRICES

5.1 INTRODUCTION

The filtering of Images and other inherently two-dimensional signals is an important part of image processing and pattern recognition. In most applications, filtering is used for pre-processing of the images from which certain features have to be extracted. For example, by the use of a boundary enhancement filter, certain regions in an image can be isolated and the features such as area, centroid, etc. can be calculated for that particular region. Similarly, in the case of detection of a faulty part in an auto-assembly line, a template matching algorithm is applied to the filtered image. The filtering of the images, thus, plays an important role in image processing and pattern recognition.

In this chapter, we illustrate the application of filtering on test images. We present a simple and approximate frequency domain designing technique for Finite Impulse Response filters. We also obtain filtered images using the same filter kernel by three different software algorithms, namely, direct convolution, filtering using the FFT and filtering using the FNTT. Next, we describe the filtering of images of dimensions larger than the basic block size in a limited memory system. This block-mode filtering algorithm is discussed in detail. The choice of the basic block size affects the filtering speed and theoretical comparisons for this trade-off are presented.

5.2 FILTERING USING TRANSFORM TECHNIQUES

The use of transform domain techniques in digital filtering is attractive only when a fast algorithm is employed to compute the transform. The transform must have the Cyclic Convolution Property (CCP). The use of transform technique for filtering involves the following steps:

1. Taking the transform of the given sequence

$$T:\ X(k1,k2) = \sum\sum x(n1,n2).\{W^{**}(n1k1+n2k2)\} \qquad (5.1)$$

2. Taking the transform of the given filter-coefficients, h(n1,n2) -----> H(k1,k2).

3. Multiplying the two transform domain representations point by point to obtain

$$Y(k1,k2) = X(k1,k2).H(k1,k2) \qquad (5.2)$$

4. Taking the inverse transform of the result Y(k1,k2) to obtain the filtered output

$$T^{-1}:\ y(n1,n2) = \frac{1}{N^2}\sum\sum Y(k1,k2).\{W^{**}(-n1k1-n2k2)\} \qquad (5.3)$$

The sequence lengths in above calculations must be large enough to avoid wrap around error (sec-2.1). We employ a filter kernel of (17x17) size on the pre-stored images or (128x128) and obtain the results of processing through the convolver.

5.3 A SIMPLE TECHNIQUE TO DESIGN 2D-FIR FILTERS

In this section we consider a technique to design simple FIR filters in the frequency domain. The design assumes "brick-wall" type filters with a cut-off based upon the requirement of transmitted or rejected signal energy [17]. Whenever a time domain representation is required we find this by inverting the frequency domain design. We stress here that this would only give an approximate filter kernel (truncated to a certain size). We choose such an appoach because the purpose in this thesis is to show the application of filtering rather than design techniques.

5.3.1 determination of cut-off based on energy transmissions [16]

It is possible to compute the Energy content of an image by taking its fourier transform, and computing the magnitude square of the coefficients,

$$E(k1,k2) = |X(k1,k2)|**2 \qquad (5.4)$$

$$= [Re\{X\}]**2 + [Im\{X\}]**2$$

so that the total energy (Parseval's theorem) would be proportional to

$$Et = \sum_{k_1} \sum_{k_2} E(k1,k2) \qquad (5.5)$$

Assuming that the transform has been centered, a circle of radius r with origin at the center of the frequency square encloses q(%) of total energy, where

$$q(\%) = 100*[\sum_{k_1} \sum_{k_2} E(k1,k2)/Et] \qquad (5.6)$$

and the summation is taken over the values of (k1,k2) which lie inside, or on the boundary of the circle.

Conversely, if the desired amount of filtered energy (pass or stop) is specified, it is possible to determine the radius r which encompasses that amount of energy. This radius can be used as the cut-off frequency to design a standard filter. We define a standard filter such as a Butterworth High Pass filter (sec-5.3.3) through an explicit mathematical expression. The time domain representation is obtained by taking the inverse Fourier transform of the coefficients. This filter is truncated to a reasonable size, which introduces some error (Gibb's oscillation). For the purpose of illustrating the use of the filter, this error is acceptable.

Also, in all cases described below, the filters are functions which affect the corresponding real and imaginary components of the Fourier transform in exactly the same manner. Such filters are referred to as Zero Phase Shift filters because they do not alter the phase of the transform.

5.3.2 Low Pass Filtering

Edges and other sharp transitions (such as noise) in the grey levels of an contribute heavily to the high frequency content of its Fourier transform. It follows, therefore, that blurring can be achieved via the frequency domain by attenuating a specified range of high-frequency components in the transform of a given image (passing the low frequency components). The filter coefficients can be obtained by specifying the cut-off and the nature of the curve (Fig.5-1) to fit in one of the following standard filters:

1. Ideal Low Pass

$$H(k1,k2) = \begin{cases} 1 & \text{if } D(k1,k2) \leq Do \\ 0 & \text{if } D(k1,k2) > Do \end{cases} \quad (5.7)$$

where Do is the distance from the center to cut-off frequency locus, D(k1,k2) is the distance from point (k1,k2) to the origin of the frequency plane, i.e.,

$$D(k1,k2) = \sqrt{\{k1**2 + k2**2\}}$$

This is a filter with sharp cut-off.

The sharp cut-off frequencies of an ideal low-pass filter cannot be realized with electronic components, although they can be simulated The choice of a small Do results in pronounced blurring and ringing.

2. Butterworth Low Pass

$$H(k1,k2) = \frac{1}{1 + 0.414 * [D(k1,k2)/Do]**2n} \quad (5.8)$$

where n is the order of the filter, and the cut-off is defined at $(1/\sqrt{2})$ of the maximum value of H(k1,k2). This is a smooth filter.

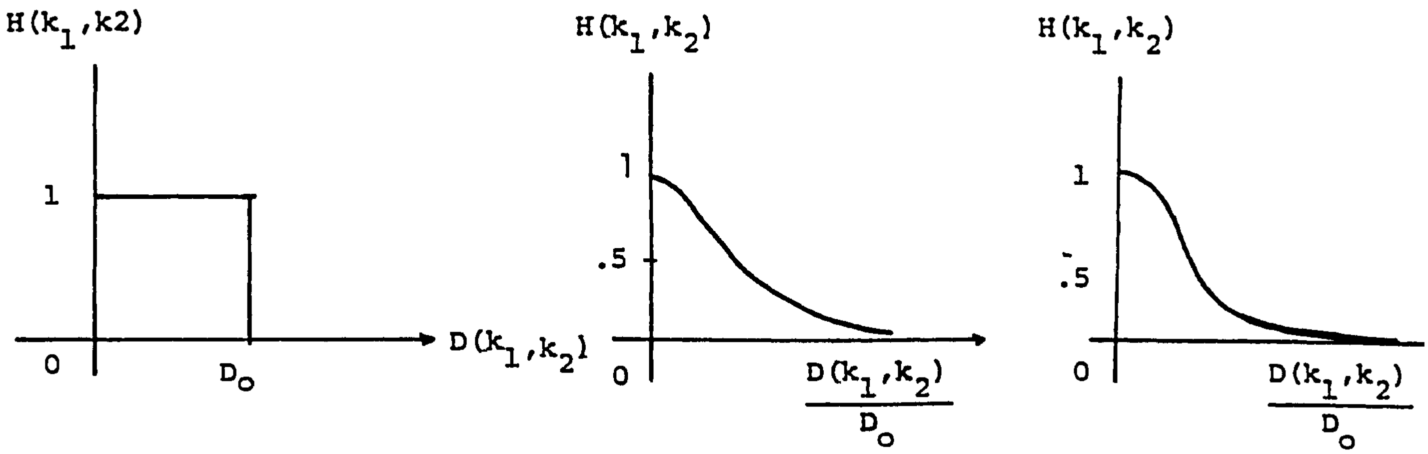


Fig. (5.1) Radial Cross-section of Low Pass Filters
(a) Ideal (b) Butterworth (c) Exponential

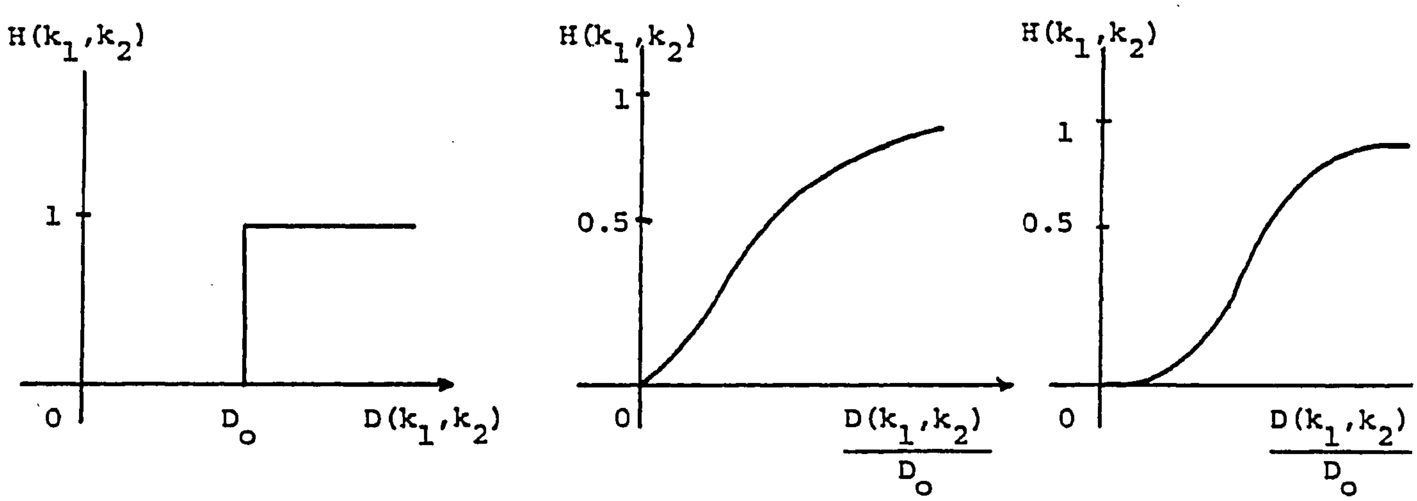


Fig. (5.2) Radial Cross-Section of High Pass Filters
(a) Ideal (b) Butterworth (c) Exponential

3. Exponential Low Pass

$$H(k1,k2) = \exp[-\{(D(k1,k2)/Do\} ** n] \quad (5.8a)$$

This is smooth filter and the n controls the rate of decay of the exponential function.

5.3.3 High Pass Filtering

Since edges and other abrupt changes in gray levels are associated with high-frequency components, image sharpening can be achieved in the frequency domain by a high pass filtering (Fig.5-2). The method of generating the coefficients for a high pass filter is same as in the case of low-pass filtering:

1. Ideal High Pass

$$H(k1,k2) = \begin{cases} 0 & \text{if } D(k1,k2) \leqslant Do \\ 1 & \text{if } D(k1,k2) > Do \end{cases} \quad (5.9)$$

2. Butterworth High Pass

$$H(k1,k2) = \frac{1}{1 + 0.414 * [Do/D(k1,k2)]**2n} \quad (5.10)$$

3. Exponential High Pass

$$H(k1,k2) = \exp[-\{Do/D(k1,k2)\} ** n] \quad (5.10a)$$

where the symbols have the same meaning as in the last section.

5.3.4 Homo-morphic Filtering

The illumination-reflectance model [16] of an image can be made the basis for a frequency-domain procedure that is useful for improving the appearance of an image by simultaneous brightness range compression and contrast enhancement. The illumination component of an image is generally characterized by slow spatial variations. The reflectance component, on the other hand, tends to vary abruptly, particularly at the junctions of very dissimilar objects. These characteristics lead to associate [17] the low frequencies of the Fourier transform of the algorithm of an image with illumination, and the high frequencies with reflectance. Although this is a rough approximation, it can be used to advantage in image enhancement.

The cross-section of the filter function for use in homomorphic filtering is shown in Fig-(5.3). The mathematical characterization of the algorithm is as follows:

1. Find the natural logarithm of the input image
2. Compute the transform of the image
3. Multiply the transform with the filter coefficients
4. Compute the inverse transform
5. Find the exponential of the result.

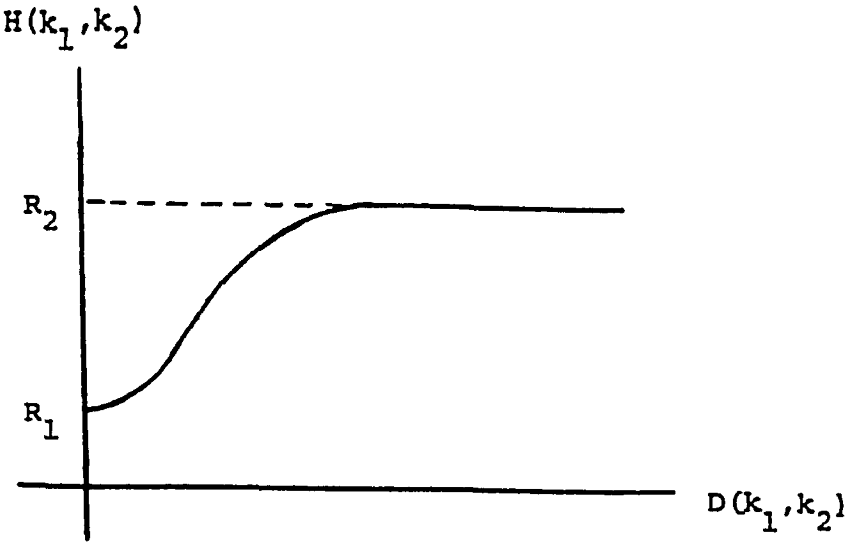


Fig. (5.3) Cross-section of a circularly symmetric filter function for use in homomorphic filtering.

Fig. (5.4) The convolver hardware unit.

5.3.5 Examples of the Image Filtering

The examples of Image filtering included in this thesis are obtained by the application of the following filters on the original images shown in Fig-(5.5) to Fig-(5.8):

1. Fig-(5.9) to Fig-(5.11) have been obtained by the use of a Butterworth high pass filter.
2. Fig-(5.12) has been obtained using the filter coefficients of a high pass filter available in the department [29].
3. Fig-(5.13) to Fig-(5.15) show the result obtained by applying the same filter coefficients through the use of the convolver, the FNTT algoritnm in software and the FFT algorithm in software.
4. Fig-(5.16) to Fig-(5.18) show the result of low-pass filtering by the application of a Butterworth low pass filter. These figures also include the filtering obtained through the use of other algorithms in software.
5. Fig-(5.19) shows the result obtained using a Homomorphic filter whose transfer function is shown in Fig-(5.3); $R1= 1.2$, $R2= 0.5$.

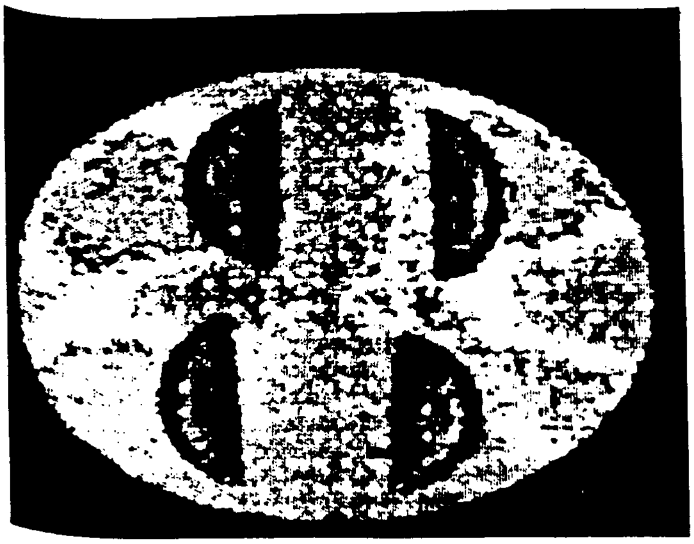


Fig. (5.5) Image I.

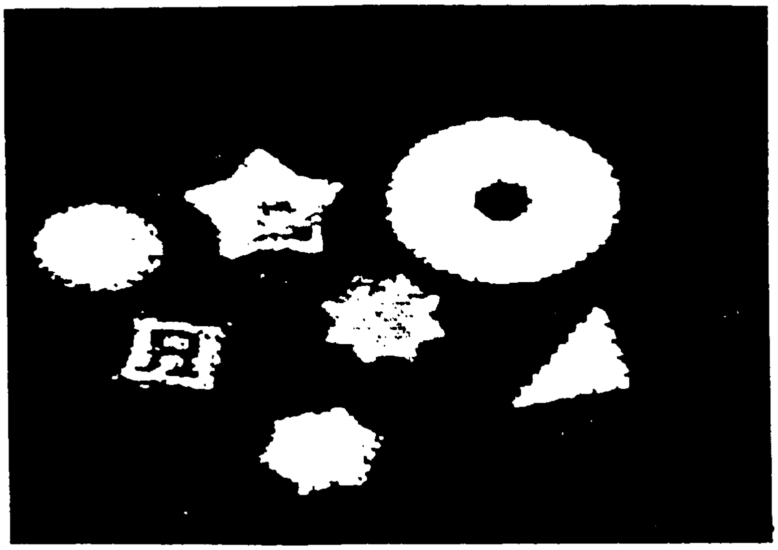


Fig. (5.6) Image 2.

Fig. (5.7) Image 3.

Fig. (5.8) Image 4

Fig. (5.5) to Fig. (5.8) Original Computer Images.

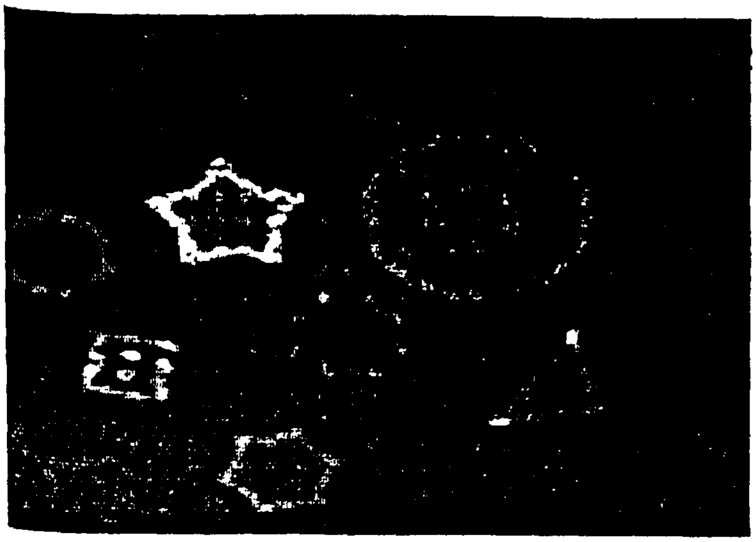


Fig. (5.9)

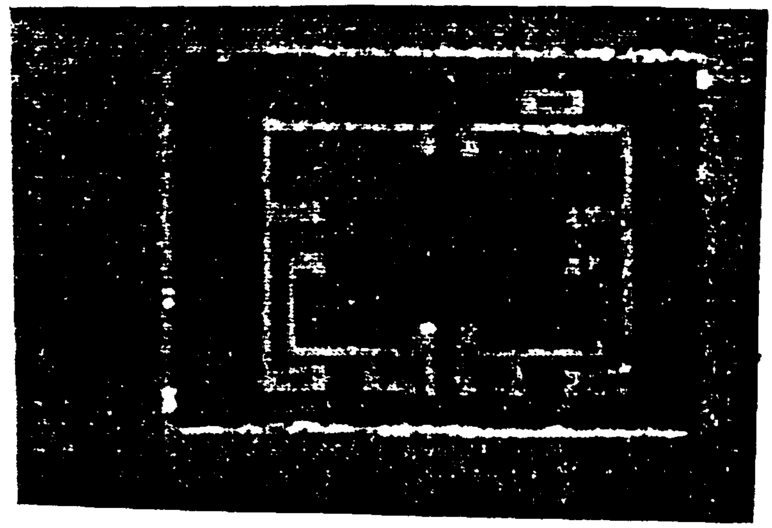


Fig. (5.10)

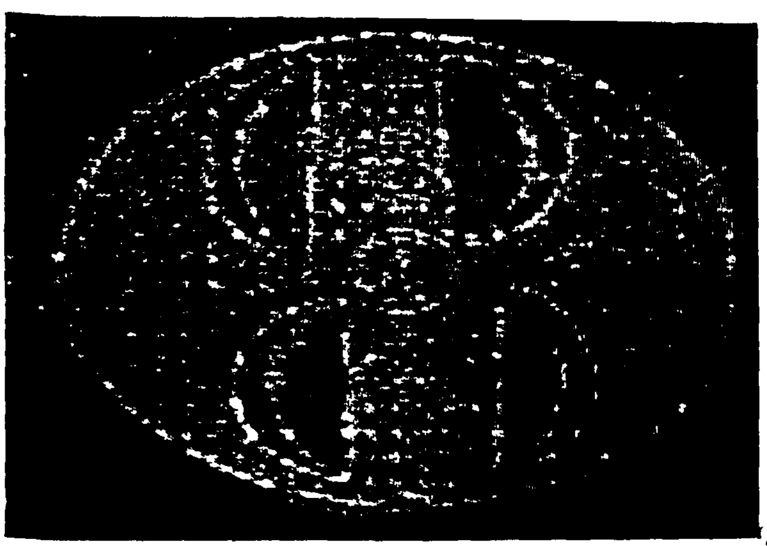


Fig. (5.11)

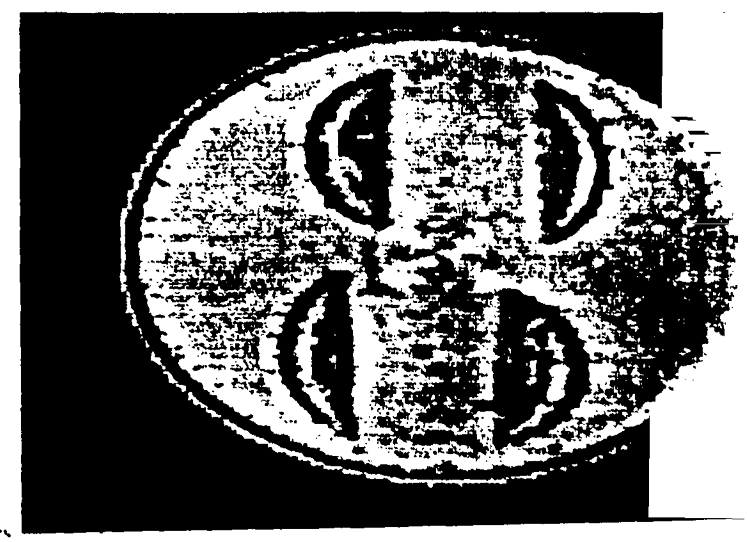


Fig. (5.12)

Fig. (5.9) to Fig. (5.12) Processed Images (High Pass Filtering)

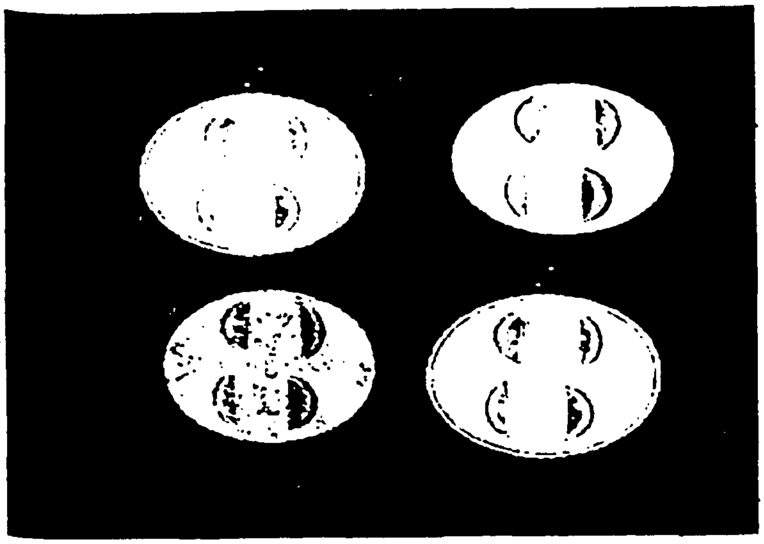


Fig. (5.13)

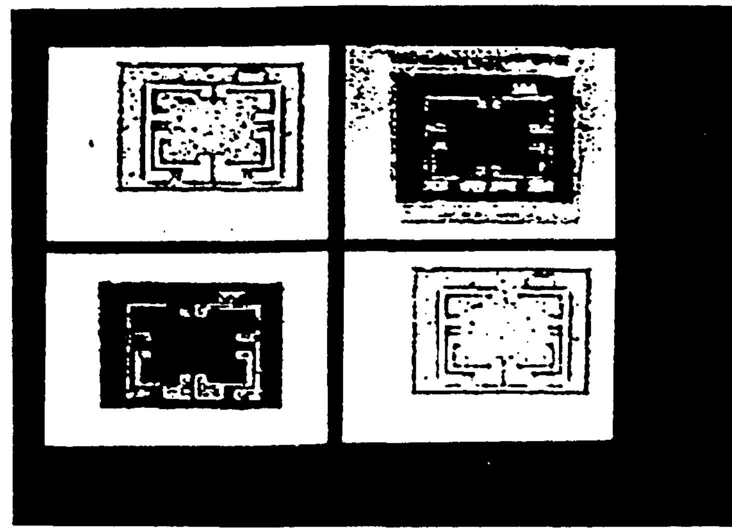


Fig. (5.14)

Fig. (5.15)

Fig. (5.13) to Fig. (5.15) Original [Right top] and Processed Images (High Pass Filtering) through the use of the convolver [Left top], FNTT algorithm in software [Right bottom] and FFT algorithm in software [Left bottom].

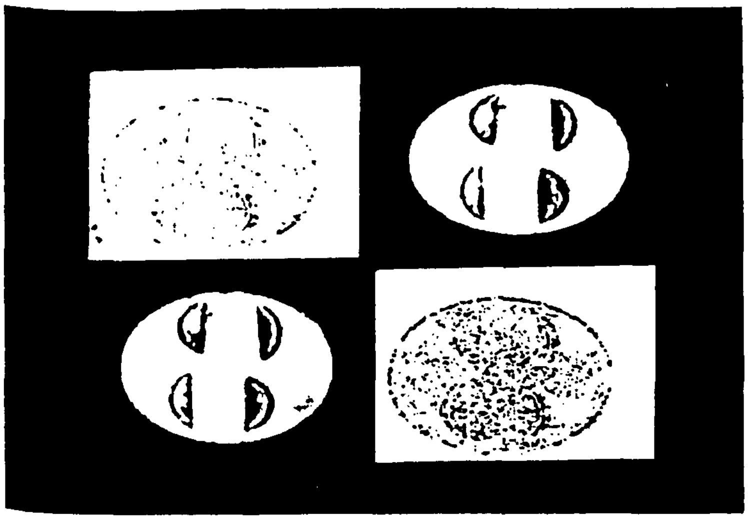


Fig. (5.16)

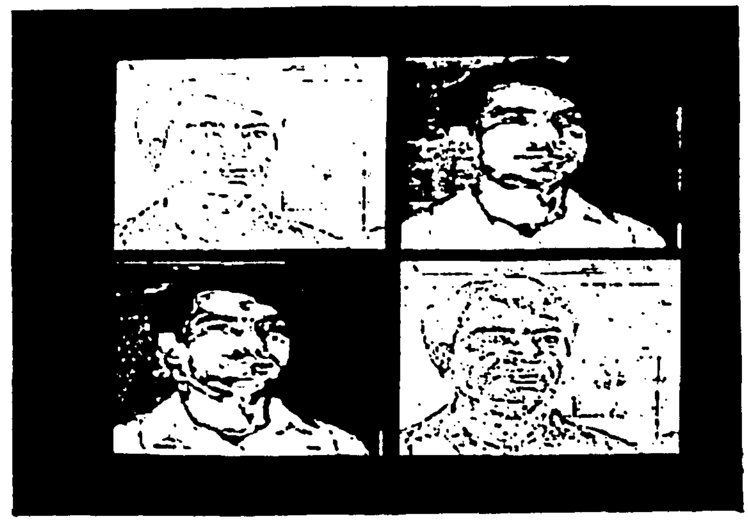


Fig. (5.17)

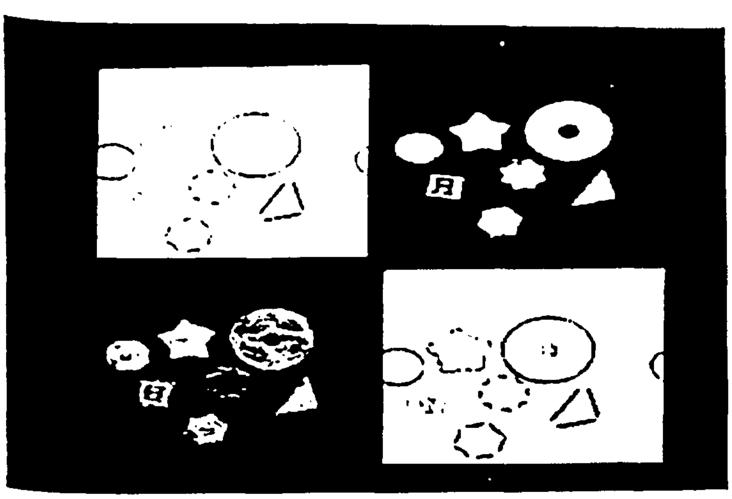


Fig. (5.18)

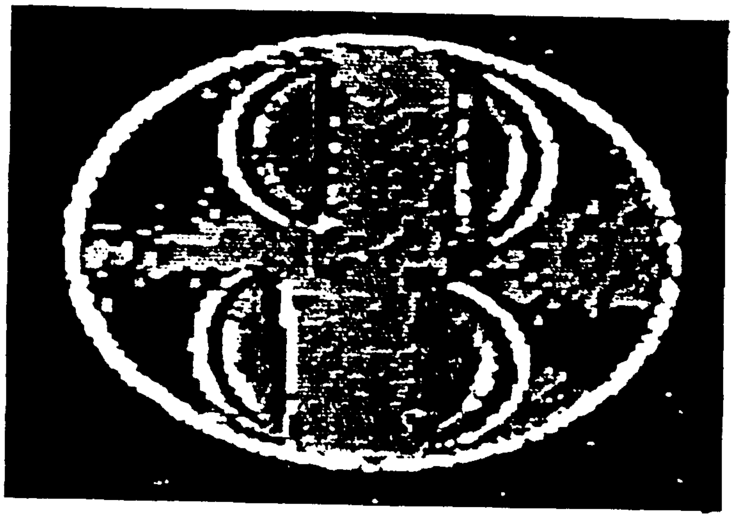


Fig. (5.19)

Fig. (5.16) to Fig. (5.18) Original [Right top] and Processed Images (Low Pass Filtering) through the use of the convolver [Left top], FNTT algorithm in software [Right bottom], and FFT algorithm in software [Left bottom].

Fig. (5.19) Application of a filter function for use in homomorphic filtering as specified in Fig. (5.3).

The time of processing of an image of size (128x128) with a filter kernel of size (17x17) by three different methods (applied through software) is given in Table-(5.1).

Table-(5.1) Approximate Computation Time *
using different methods of convolution in software
for image-size of (128x128), 256 levels.

Method	Processing Time, sec
direct convolution	1500
convolution using FFT	70
convolution using FNTT (in software)	193

* Reported using SEL computer (Programming in FORTRAN-77 and excluding File-I/O time)

5.4 FILTERING OF IMAGES OF LARGER DIMENSIONS

Next we consider the filtering of larger images. A common problem in signal filtering is that of filtering a signal of very long or indefinite length by an impulse response that is of a short length. For image processing in a limited main memory system, such as a mini-computer, it is necessary to employ special techniques to improve the computaional efficiency when the whole image to be transformed can not be accomodated in the main memory. A straight forward method is to store the data on a disk, find the transform of rows, transpose the result and then find the transform of columns using the same computational element. Since the processing is performed taking one array at a time from the disk such a method is I/O bound and is very slow owing to the relatively slow disk access time. A more efficient method is to section the input image into smaller blocks that can be accomodated in the memory of the computational element and then either use the overlap-save or overlap-add technique. This method, referred to as block-mode filtering [18], provides a very efficient means to compute two-dimensional convolutions when the dimensions of the filter kernel are small. An alternate method has recently been proposed by Kraats and Venetsanopoulos [19] that is computationally more efficient in certain instances, specially when the filter kernel size is comparable to the basic block of the input data. Since almost all cases of two-dimensional filtering employ a much smaller filter kernel (N>>L), block-mode filtering is preferred.

5.5. BLOCK-MODE FILTERING

The two-dimensional convolution is defined (eqn.2-1) as

$$y(j,\ k) = \sum_{n=0}^{j} \sum_{m=0}^{j} h(j-m\ ,\ k-n)\ f(m,\ n) \tag{5.12}$$

We begin by proving a fundamental property of two-dimensional transforms and convolutions.

LEMMA. Given two matrices $f(m,\ n)$ and $h(j,\ k)$, both of dimensions D_1 by D_2 let the transforms of these matrices be

$$F(r,\ s) = \sum_{n=0}^{D_2-1} \sum_{m=0}^{D_1-1} f(m,\ n)\ \exp\left[-\,i2\pi\left(\frac{rm}{D_1} + \frac{sn}{D_2}\right)\right],$$

$$H(r,\ s) = \sum_{k=0}^{D_2-1} \sum_{j=0}^{D_1-1} h(j,\ k)\ \exp\left[-\,i2\pi\left(\frac{rj}{D_1} + \frac{sk}{D_2}\right)\right].$$

Then the inverse transform of the product of these transforms, that is

$$y(j,\ k) = \frac{1}{D_1\,D_2} \sum_{s=0}^{D_2-1} \sum_{j=0}^{D_1-1} H(r,\ s)\ F(r,s)\ \exp\left[i2\pi\left(\frac{rj}{D_1} + \frac{sk}{D_2}\right)\right],$$

is equivalent to the convolution form

$$y(j,\ k) = \sum_{n=0}^{k} \sum_{m=0}^{j} h(j-m,\ k-n)\ f(m,\ n) + \sum_{n=k+1}^{D_2-1} \sum_{m=j+1}^{D_1-1} h(j+D_1-m,\ k+D_2-n)\ f(m,\ n) \tag{5.13}$$

PROOF. The proof is similar to the proof for 1-D cases . We substitute for $H(r,\ s)$ and $F(r,\ s)$ in the relation defining $y(j,\ k)$ and have

$$y(j,k) = \sum_{q=0}^{D_2-1} \sum_{p=0}^{D_1-1} \sum_{n=0}^{D_2-1} \sum_{m=0}^{D_1-1} h(p,1)\, f(m,n)$$

$$\times \frac{1}{D_1 D_2} \sum_{s=0}^{D_2-1} \sum_{r=0}^{D_1-1} \exp\left[i2\pi \left(\frac{r(j-m-p)}{\quad} \quad \frac{s(k-n-q)}{\quad} \right) \right] . \tag{5.14}$$

However, the double sum on the exponential can be written as

$$\frac{1}{D_1 D_2} \sum_{r=0}^{D_1-1} \exp\left(i2\pi \, \frac{r(j-m-p)}{D_1} \right) \sum_{s=0}^{D_2-1} \exp\left(i2\pi \, \frac{s(k-n-q)}{D_2} \right) .$$

This is a product of two finite sums. Both sums have the orthogonality property demonstrated by Helms [22], that is,

$$\sum_{r=0}^{D_1-1} \exp\left(i2\pi \, \frac{r(j-m-p)}{D_1} \right) = \begin{cases} D_1 & \text{if } j-m-p = tD_1 \\ 0 & \text{otherwise,} \end{cases}$$

$$\sum_{s=0}^{D_2-1} \exp\left(i2\pi \, \frac{s(k-h-q)}{D_2} \right) = \begin{cases} D_2 & \text{if } k-n-q = tD_2 \\ 0 & \text{otherwise.} \end{cases}$$

All the indices in the first summation above are defined over $[0,D1-1]$; all the indices in the second summation are defined over $[0,D2-1]$. As a result, in the two summations we see that $t=0$ or $t=-1$ are the only possible values of t. As a result we can use $(j-m-p=0)$ or $(j-m-p)=-D1$ to eliminate the summation on p in Eqn.(5-14); we can also use $(k-n-q)=0$ or $(k-n-q)=-D2$ to eliminate the summation on q in Eqn.(5-14). Eqn.(5-13) is the result, and the theorem is proved.

Eqn.(5-13) shows a convolution summation with two terms. The first term on the right-hand side is the desired convolution in the form of Eqn.(5-12). The second term is the so-called wraparound-error term, the term that results from the inherent periodicity in the use of discrete Fourier transforms. The problem of convolution with discrete Fourier transforms is to force the wraparound error to be zero. The problem of block-mode filtering is to decompose the convolution of Eqn.(5-13) into a large number of smaller convolutions.

5.6 BLOCK-MODE FILTERING ALGORITHM

The following is the algorithm for block mode filtering.

1. Choose two numbers $D1 > J$ and $D2 > K$.
2. The matrix $h(j,k)$ is extended by the addition of rows and columns of zeros to form a matrix $hc(j,k)$, defined as

$$hc(j,k) = \begin{cases} h(j,k) & 0<j,k<J-1,K-1 \\ 0 & J,K<j,k<D1-1,D2-1 \end{cases}$$

3. Let b(m,n) be a block that is composed of the first D1 rows and D2 columns of f(m,n), that is,

   $$b(m,n) = f(m,n) \quad \text{for } 0<m,n<D1-1,D2-1$$

4. Compute the transforms of size D1 by D2 of hc(j,k) and b(m,n). Form the product of the transforms and then compute the inverse transform of the products. Call this inverse transform a matrix c(j,k) of size D1 by D2.

5. Part of c(j,k) contains the wrapround error. We save as valid data the submatrix of c(j,k) defined by the range of indices

   $j = J-1, J, \ldots, D1-1.$

   $k = K-1, K, \ldots, D2-1.$ This is a submatrix of size D1-J+1 by D2-K+1. The rest of c(j,k) is discarded.

6. The procedure now splits into two alternative choices for the construction of the next block.

   Form a new block b(m,n) from f(m,n) of size D1 by D2 and such that the first J-1 rows of the new block are the same as the last J-1 rows of the old block. That is, row D1-J+1 of the old block is row 0 of the new block, row D1-J+2 of the old block is row 1 of the new block,

   and so on; or,

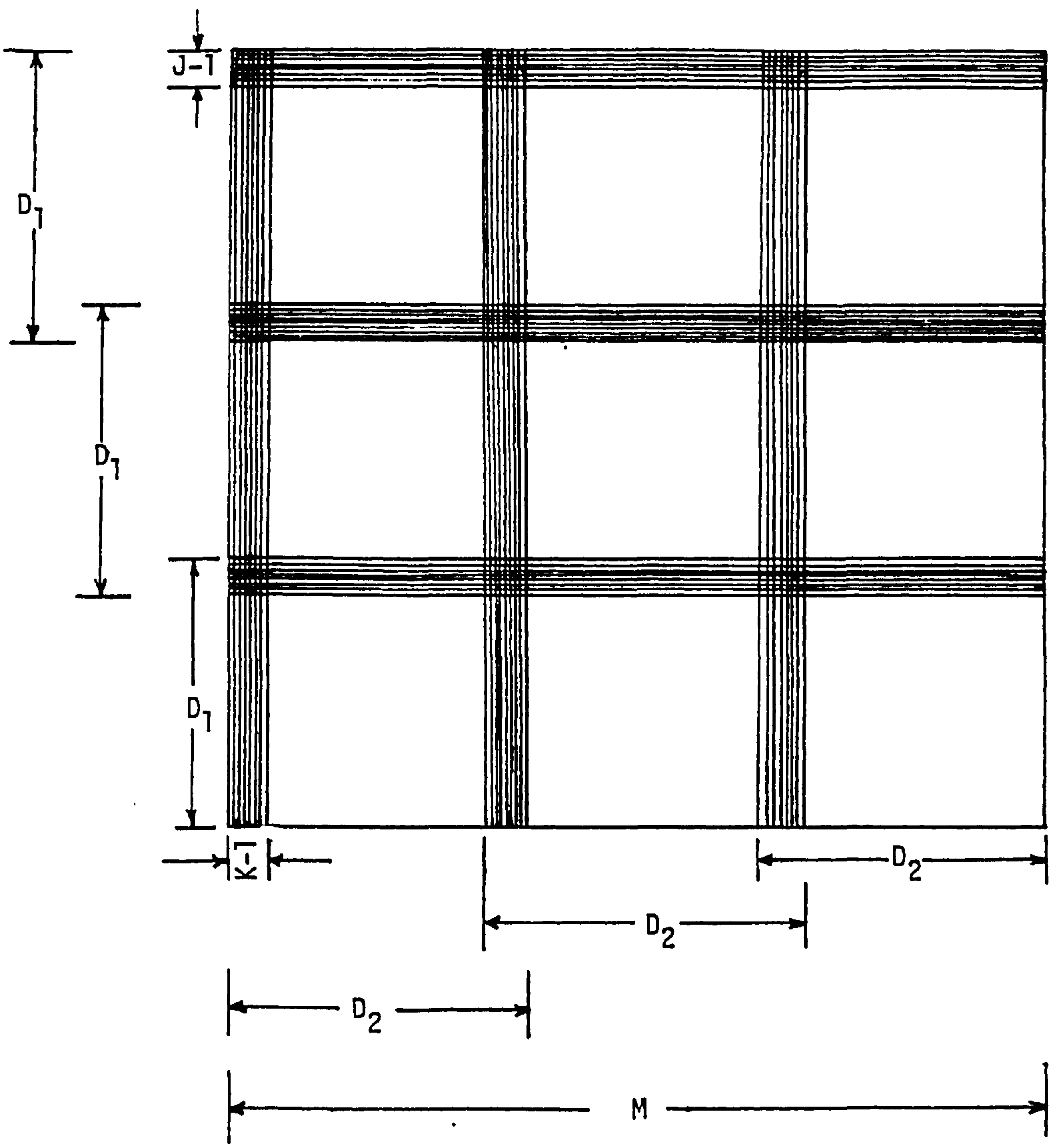


Fig. (5.21) Construction of Blocks in Block-Mode Filtering Algorithm.

Form a new block b(m,n) from f(m,n) of size D1 by D2 and such that the first K-1 columns of the new block are the same as the last K-1 columns of the old block. That is, column D2-K+1 of the old block is column 0 of the new block, column D2-K+2 of the old block is column 1 of the new block, and so on.

7. Repeat steps 4 through 6 until either one of the following conditions arises.

Row M-1 of the picture f(m,n) is included in the new block. Add rows of zeros to the new block, if necessary, to make it of size D1 by D2 and repeat steps 4 through 6. Now discard the first D2-K+1 columns of the picture f(m,n) and redefine the column indices of the picture by subtracting D2-K+1 from the column index n of f(m,n). This defines a new picture of size M by N-D2+K-1. Go back to step 3 and proceed. Or,

Column N-1 of the picture f(m,n) is included in the new block. Add columns of zeros to the new block, if necessary, to make it of size D1 by D2 and repeat steps 4 through 6. Now discard the first D1-J+1 rows of the picture f(m,n) and redefine the row indices of the picture by subtracting D1-J+1 from the row index m of f(m,n). This defines a new picture of size M-D1+J-1 by N. Go back to step 3 and proceed.

8. Steps 3 through 7 are repeated as directed until the entire picture has been processed with the overlapping blocks. The juxtaposition of all the submatrices saved in step 5, where the juxtaposition is formed in the same sequence as the construction of the overlapping blocks, constitutes the filtered picture.

5.6.1 Wrap-around Error Consideration for First Block

If the first J-1 rows and the first K-1 columns of the picture contain important information that must be filtered, then the picture must be enlarged by 'padding' so that the discard operations of step 5 would not result in discarding information that it was desired to filter. For example, a new picture can be constructed by the following procedure. Let f (m,n) be the extended (or padded) picture of size J-1+M by K-1+N. The picture fc(m,n) is constructed as

$$fc(m,n) = \begin{cases} C & 0<m,n<J-2,K-2 \\ f(m,n) & J-1,K-1<m,n<J-1+M,K-1+N \end{cases}$$

where C is any constant. This extended picture is now used in the eight-step procedure above, that is, whenever f(m,n) is stated above, we replace it by fc(m,n). Similarly, we would replace M and N in the eight steps above by M'=J-1+M and N'=K-1+N, respectively.

5.6.2 Number of Blocks to be Processed

Because the blocks must be overlapped, the picture is not processed in blocks of size D1 by D2 . Successive blocks are overlapped in both rows and columns. From the construction in step 6a, 6b, and 7a, 7b, we can see that the total subset of the picture that is processed on any one iteration through the eight steps is a submatrix of size D1-J+1 by D2-K+1. Therefore, the total number of blocks to be processed is given by

$$\lceil a \rceil = \frac{M*N}{(D1-J+1)*(D2-K+1)} \qquad (5.15)$$

The symbol $\lceil a \rceil$ stands for the smallest integer greater than a. We use the smallest integer greater than the expression shown because a fraction of a block must be processed as one full block by padding out the fractional block (step 7a or 7b). If we had used an extended picture, as discussed in the previous section, then we would replace M and N in Eq. (5) by M'=D1-J+1+M and N'=D2-K+1+N.

5.7 TIMING CONSIDERATION IN PROCESSING LARGER IMAGES

The number of arithmetic operations required in the processing of each block is proportional to the number of computation involved in the 2-D transform domain technique of filtering. The total time for processing depends upon the number of blocks to be processed multiplied by time required for each block-processing. Also the number of operations required depends upon the radix of the algorithm and the nature of the butterfly.

We derived relations between a 2-D-radix-2 and a 1-D-radix-2 transform algorithm in sec-(3.8) and it was shown that a saving of approximately 25% occurs when the former implementation is used. The derivation in this section are for a 1-D-radix-2 structure and the correction factor can be applied to obtain the computational efficiency with respect to other butterfly structures.

Assuming a 1-D-radix-2 algorithm for fast transforms, a sequence of D1 points can be transformed with a total of

$$N(M) = (D1/2).\log D1 \qquad \text{multiplications}$$

$$N(A) = (3.D1/2).\log D1 \qquad \text{additions}$$

where multiplications and additions are complex when the FFT structure is considered and are of real type when the FNTT structure is considered for a single modulus. Use can be made of the fact that the data are real, and thus in the use of the FFT structure a 50 percent saving in time can be obtained. The transform of an image of size D1xD2 requires,

$$N(M) = (D1.D2/2).\{\log D1 + \log D2\}$$

$$N(A) = (3.D1.D2/2).\{\log D1 + \log D2\}$$

Since for convolution results, there are two transforms (n-stage) and one pairwise product of matrices (of transform of coeffs with image transform), total number of computation would be

$$Nc(M) = D1.D2.\{\log D1 + \log D2 + 1\}$$

$$Nc(A) = 3.D1.D2.\{\log D1 + \log D2 + 1/3\}$$

If (t1) is the time for an addition and (t2) is the time for a multiplication then the total time for processing is,

$$t(b) = Oc(M).(t2) + Oc(A).(t1)$$

and since there are B such blocks, the total convolution time is,

$$\begin{aligned} T(conv) &= B.t(b) \\ &= D1.D2.[(\log D1+\log D2+1).(t2) \\ &\qquad +3.(\log D1+\log D2+1/3).(t1)] \\ &\qquad *Intg[(M.N)/(D1-J+1).(D2-K+1)] \end{aligned} \tag{5.16}$$

This estimation only considers the computation time. The time required in fetching the data is not taken into account in this calculation.

It can be seen from the relation that it is possible to vary D1 and D2 such that total filtering time may be controlled. However D1 and D2 must be integer power of 2. Finally, D1 and D2 must be chosen such that matrix (D1xD2) fits into computer memory as a basic block. The best way to choose D1 and D2 is then to evaluate Eqn. (5.16) for various combinations and to choose (D1xD2) such that T(conv) is minimized subject to the condition that the data fits into some allowable amount of computer memory. Such a tabulation is presented in Table-(5.3) for (t1)= 1 usec., (t2)= 5 usec., M=N=(256 and 1024), and J=K=(17 and 25). It is observed, as expected, that the larger the size (D1xD2), the smaller the time T(conv).

Table-(5.3) Processing Time by the use of different size of Basic Block (time in sec) M=N=1024, J=K=25, t1=1 usec, t2= 5 usec

	D2					
D1	32	64	128	256	512	1024
32	1442	630	526	509	519	541
64	630	274	227	218	221	230
128	526	227	187	179	181	187
256	509	218	179	171	172	177
512	519	221	181	172	173	177
1024	541	230	187	177	178	182

Table-(5.3b) Processing Time by use of Different size of Basic Block (time in sec) M=N=256, J=K=17, t1= 1 usec, t2= 5 usec

	D2			
D1	32	64	128	256
32	22.5	16.4	15.3	15.4
64	16.4	11.9	11.0	11.0
128	15.3	11.0	10.1	10.1
256	15.4	11.0	10.1	10.0

5.8 CONCLUSIONS

In this chapter, we have first described a simple and approximate method of 2-D Finite Impulse Response filter design. The test images have been processed by four different methods to obtain image smoothing and image enhancemnt. The steps in the use of block- mode filtering algorithm, used for filtering of large images, have been described and the processing time for typical cases have been tabulated with reference to different sizes of the basic processing block.

Chapter VI

CONCLUSIONS

The objective of the research work described in this thesis was to present an elaborate explaination of the theory, hardware implementation, use and analysis of a two-dimensional digital filter. The filter hardware utilizes a fast number theoretic transform algorithm for high speed processing of two-dimensional signals. The main area of application of this filter is in Image Processing.

The conclusions of this study are summarised below:

1. Design considerations of a two-dimensional convolution filter has been described.

   This includes the following:

   a) The theoretical background necessary to understand the architecture of the convolver (topics of interests from the Residue Number System, the Number Theoretic Transform, Fast algorithm for the NTT, 2D-Ordered-Input- Ordered-Output NTT algorithm for butterfly implementation) has been described in detail.

   b) The hardware implementation of the filter has been described. Particular attention has been paid to the implementation of the butterfly unit.

c) The functional detail of each of the units in the convolver has been described.

d) A systematic way to write the interfacing software has been described taking the example of the High Speed Device (HSD) interface to the mini-computer SEL-32/27. Also, the user has been referred to the available software to obtain the processing through the filter.

2. The steps in the use of the convolver filter has been described. A new user can design his own 2-D FIR filter, store the coefficients in a file and can operate this filter over any image of size (128x128) pre-stored in a file. Further all the steps in use of the convolver have been combined together so that a new user can use the filter through a simple command in the format (in FILTER directory):

TSM> FILTER IMAGE COEFF OUTPUT <cr>

where IMAGE is input image file, COEFF is the coefficient file and OUTPUT is the output file.

The software has been written and modified to make the convolver a user friendly device. A directory of the available software is included in the appendix. The use of the convolver has been illustrated through various examples.

3. Various efficiency analysis on the convolver have been presented. This includes the following:

(a) 1-D-radix-2 and 2-D-radix-2 butterflies have been compared in terms of their computation requirements

(b) The memory requirement in multiplication by the sub-modular look-up table approach has been calculated.

(c) The Chinese Remainder Theorem and the Mixed Radix Conversion method for implementation of a Residue to Binary Converter have been compared in terms of their hardware requirements.

4.

(a) Timing diagrams to illustrate the working of the butterfly in a pipeline implementation have been prepared.

(b) Throughput rates obtainable by the use of serial sequential and cascade processing (OICO-algorithm) were compared. The processing speed of the convolver has also been calculated.

(c) Two design schemes to improve the speed of filtering have been proposed.

(d) A table of main IC's and their usage in the convolver has been prepared.

5.

(a) A simple and approximate technique for design of a 2-D Finite Impulse Response filter has been discussed.

(b) The block-mode filtering algorithm used for processing of large matrixes has been described in detail. Theoretical comparisons are made to illustrate the trade-off between speed and the size of a basic block when large matrixes are filtered through the use of block-mode algorithm.

TWO-DIMENSIONAL OIOO-NTT ALGORITHM

In [10] a multi-dimensional algorithm for computing a class of unitary transforms is derived, following the development suggested in [14, 12]. Since tee NTT has the same structure as the DFT, the general derivation [10] can be applied when the transforms are defined in a ring Z(M) of integers modulo M for a two (or multi-) dimensional array for (n_1, n_2) of size N in each dimension.

$$F(k_1, k_2) = \left| \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} f(n_1, n_2)\, \alpha^{\sum_{i=1}^{2} n_i k_i} \right|_M \tag{A.1}$$

where α is the primitive N^{th} root of unity in Z(M). When the elements of input and output arrays f and F are arranged in a lexicographical order, the NTT of eqn. (A.1) can be written as

$$F = |T \cdot f|_M \tag{A.2}$$

where T is a $(N^2 \times N^2)$ matrix performing a Number Theoretic Transformation on the $(N^2 \times 1)$ input vector f and yielding a $(N^2 \times 1)$ output vector F.

The transformation matrix T can be factorized into Kronecker products of one-dimensional transformation matrix T_N [10] and eqn. (A.2) could be written as

$$F = \left| (T_N \otimes T_N) \cdot f \right|_M \tag{A.3}$$

where $\otimes$ represents the Kronecker product and T_N is of size $(N \times N)$ with elements $t_{ij} = \alpha^{ij}$.

When $N = r^n$ is a composite number conditions [ ] has shown that the transformation matrix T_N can further be expressed as product of submatrices as,

$$T_N = \prod_{i=1}^{n} \mu_i^{(r)} \ S_i^{(r)} \tag{A.4}$$

where r is the radix of factorization for T_N. When radix-r is implied, we drop the super-script r to write

$$T_N \ \prod_{i=1}^{n} \mu_i \ S_i \tag{A.5}$$

and then eqn. (A.3) is expressed as

$$F = \left| \left(\prod_{i=1}^{n} \psi_i \ R_i \right) \cdot f \right|_M \tag{A.6}$$

where

$$\psi_i = \mu_i \otimes \mu_i$$

$$R_i = S_i \otimes S_i \tag{A.7}$$

and r, the radix of factorization is implied.

To see what eqn. (A.6) implies, let us define [10] a two-dimensional permutation operator δ_i as,

$$\delta_i = q_i \otimes \ q_i$$

and

$$q_i = I_{r^{n-i}} \otimes P_{r^i} \tag{A.8}$$

where I_K denotes the identity matrix of dimension K and P_K is the ideal shuffle base-r permutation matrix operating on a vector of dimension K. In eqn. (A.4) to (A.7) , μ_i is the weighting or twiddle operator specifying multiplications by the twiddle factors and is given by

$$\mu_i^{(r)} = \mu_i = I_{r^{n-i}} \otimes D_{r^i} \quad , \quad i = 2, 3, \ldots, n \qquad (A.9)$$

where

$$D_{N/K} = \text{quasi} \quad \text{diag}\left(I_{N/rk}, L_k, L_{2k}, \ldots, L_{(r-1)k}\right)$$

and

$$L_m = \text{dia}(0, m, 2m, \ldots, (N/_{rk-1})\, m)$$

where an element t of the diagonal matrix L_m represents the actual element, α^t, and $S^{(r)}$ is a pre-weighting r-point transform operator given by

$$S^{(r)} = S = I_{N/r} \otimes T_r \qquad (A.10)$$

$$T_r = \begin{bmatrix} 0 & 0 & 0 & ------ & 0 \\ 0 & N/r & 2N/r & ------ & (r-1)\, N/r \\ 0 & 2n/r & 4N/r & ------ & 2(r-1)\, N/r \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & r-1)\frac{N}{r} & 2(r-1)\frac{N}{r} & --- & (r-1)^2 N/r \end{bmatrix} \qquad (A.11)$$

where an element of t or T_r represents the matrix element, α^t.

now $\quad S_{i-1} = S \cdot q_i \quad i = 2. 3, \ldots, n$

$\quad S_n = S$

and $\quad \mu_1 = q_1 = T_N$

then for $i \neq 1$

$$\therefore\ S_{i-1} = S\, q_i$$

$$= (I_{N/r} \otimes T_r)\, q_i$$

$$= q_i\, q_i^{-1}\, (I_{N/r} \otimes T_r)\, q_i$$

$$= q_i\, (I_{N/r^2} \otimes T_r \otimes T_r) \qquad \text{(A.12)}$$

thus

$$S_{i-1} = q_i\, S' \qquad i = 2, 3, \ldots, n$$

when

$$S' = I_{N/r^2} \otimes T_r \otimes I_r$$

and, $$R_{i-1} = S_{i-1} \otimes S_{i-1} \qquad i = 2, 3, \ldots, n$$

$$= (q_i\, S') \otimes (q_i\, S')$$

$$= (q_i \times q_i)\, (S' \otimes S')$$

$$= \delta_i \cdot (S' \otimes S') \qquad i = 2, 3, \ldots, n \qquad \text{(A.13)}$$

$$R_n = S_n \otimes S_n$$

$$= S \otimes S \qquad \text{(A.14)}$$

which shows that the operator R_{i-1} $(i \neq 1)$ always operates over data which are N/r^2 words apart. In the first iteration, however, the operator R_n operates on data which are N/r words apart.

Appendix B

PROCEDURE TO USE AN HSD DEVICE ON SEL MINI-COMPUTER

EXAMPLE: NTT-Convolver

The two-dimensional NTT-Convolution filter is interfaced with the SEL-32/27 Mini-computer through a High Speed Data (HSD) interface. The data I/O rate through the use of the HSD are considerably high (1.2 usec/ 32-bit word) and the interfacing procedure is different than that of an RS-232 port.

The HSD handler is a software component which provides general device support for user devices connected to MPX-based series-32 computers. The handler design is based on the notion that the HSD (hardware) acts as a controller. The HSD handler provides a software interface between MPX-32 Tasks and the HSD. The I/O requests could be accepted in either of two formats:

a) File Control Block (FCB) Format

b) STARTIO Format

FCB Format

The FCB interface is designed to permit a device command and/or a data transfer to be initialized as a result of a user request. An FCB is created with the address of data and the transfer count is mentioned in the FCB. EXPANDED FCB must be used and such could be done using M.DFCBE in-built subroutine. A call is given to IOCS H.EXEC module in form of 'Service Call (SVC)'. The handler constructs first constructs logical IOCL, takes care of crossing of map blocks and then converts them to physical IOCL. An example is illustrated below:

Making an FCB Format I/O request to HSD handler

```
 EXECUTE ASSEMBLE         (in assembly language)
...........
..........
          BOUND 4      , FCB to start at word boundary
          M.DFCBE HSDFCB,NAME,0,XARRAY,,,NWT,,DFI,,,,,,,,,NMNWT,ERNWT
          -------
          -------
          LA 1,HSDFCB   Load address of label in Reg-1
          SVC 1,X'nn'   Service call number
          -------
          -------
where
HSDFCB    is the label given to expanded FCB
NAME      is the Logical File Code (LFC) to which I/O
          is to be performed
XARRAY    is the name of data array
NWT,...etc are the available options for NOWAIT,ERROR PROCESSING,..e
```

STARTIO Format

In this format the user can create his own IOCL by means of data statements. Information can be stored into these IOCL's later by separate instructions. Using his own IOCL, the user creates an FCB and uses EXECUTE CHANNEL PROGRAM format for this FCB. The address of IOCL is given as input in the FCB. A STARTIO has to be issued then to request Service Call. An illustrative example is given below:

```
EXECUTE ASSEMBLE
.................
IOCL   GEN  8/X'A2',8/X'30',16/0    (1010 0010 0011 C000 00...)
       GEN  32/W(DATA)
       DATAD 0
..........
BOUND 4
M.FCBEXP HFCB,NAME,IOCL,0,,,,,,ERROR
..........
..........
LA 1,HFCB       Load add. of HFCB
SVC 1,X'25'     Channel prcqram no.25 is for EXECUTE
                CHANNEL PROGRAM Request of the Handler.
.........
```

These calls also can be made using FORTRAN statements 'CALL H.IOCS,n' where n stands for the operation process number.

A Sample procedure for Data-transfer between

SEL and Convolver

The various possible transfers are :

1. Transfer of NTT of Filter Coefficients to the Convolver
2. Transfer of Image data
3. Transfer of Command
4. Transfer of Filtered image or Camera input from Convolver to SEL

We will describe the above transfer and the handler directives. These interfacing programs have been written in Assembly Language.

1-TRANSFER OF NTT OF FILTER COEFFICIENTS

It may be noted that the filter coefficients are multiplied by a proper scaling factor and are made integers. The NTT of the filter coeff. is taken and is reduced to modulo 62 and 63 (6-bits each) and a 12-bit word for each moduli (total 24-bit) is sent to Coefficient Memory of the Filter through the subroutine LDCOEF. The following is the desciption where a word of 32-bit (with 8-MSBs as zero) is sent to the convolver.

CALL LDCOEF(IDAT,IER1,IER2,IOCM)

where IDAT is name of the Data-array

IER1 is the status of the handler posted in FCB

IER2 is the status of device posted by handler in IOCl

IOCM is the parameter to indicate I/O operation completion

The IOCL used for this purpose is given as follows:

EXECUTE ASSEMBLE

```
IOCL      GEN 8/x'A2',8/x'30',16/0
          GEN 32/W(DATA)
          DATAD 0
          GEN 8/x'02',8/x'20',16/x'4000'
          GEN 32/0
          DATAD 0
          GEN 8/x'A8',8/x'00',16/0
          GEN 32/W(DATA)
          DATAD 0
```

where

```
HSD command 'A2' means INPUT TRANSFER , DEVICE STATUS REQUEST
                      and COMMAND CHAIN
UDD command '30' means CLEAR MAIN COUNTER OF FILTER
GEN 32/W(DATA)   is to generate dummy data address
HSD command '02' means OUTPUT TRANSFER and COMMAND CHAIN
UDD command '20' means LOAD COEFF. INTO FILTER
GEN 16/x'4000'   gives data count in HEX (16K words)
HSD command 'A8' means INPUT TRANSFER, DEVICE STATUS REQUEST
                      and INTERRUPT AFTER COMPLETED PROCESSING IOC
UDD command '00' means NOP
```

2-TRANSFER OF IMAGE TO THE CONVOLVER

The subroutine used for this purpose is LDIMG and is called as follows:

LDIMG(IDAT,IER1,IER2,IOCM)

the parameters have same explaination as above. It may be noticed that the transfer of NTT of Coeffs. and that Image data is very much the same. There are two differences, namely, the number of total words to be transferred is '1000' (HEX) i.e. 4K words, and UDD command is '10' instead of '20' to load the IMAGE in the proper memory location ('20' describe a FILTER FUNCTION to load COEFFs.). Thus the IOCL would be:

```
IOCL    GEN 8/X'A2',8/X'30',16/0
        GEN 32/W(DATA)
        DATAD 0
        GEN 8/X'02',8/X'10',16/X'1000'
        GEN 32/0
        DATAD 0
        GEN 8/X'A8',8/X'00',16/0
        GEN 32/W(DATA)
        DATAD 0
```

3-TRANSFER OF COMMAND AND STATUS

This is achieved through the use of subroutine

CALL CLSRT

which is used for clearing the main counter of the filter and starting the convolver processing and for reading the status of the convolver.

The IOCL posted in this case are similar in part as used for above two data transfer. This time there is no continuous data transfer, rather only the UDD command transfers

4-TRANSFER OF FILTERED IMAGE FROM THE CONVOLVER TO SEL

The Subroutine used for this purpose is called as

CALL STFIMG(IDAT,IER1,IER2,IOCM)

with the parameter description as described above. The IOCL used in this subroutine is as follows:

```
EXECUTE ASSEMBLE
................
IOCL     GEN 8/X'A2',8/X'30',16/0
         GEN 32/W(DATA)
         DATAD 0
         GEN 8/X'82',8/X'10',16/X'2000'
         GEN 32/0
         DATAD 0
         GEN 8/X'A8',8/X'00',16/0
         GEN 32/W(DATA)
         DATAD 0
```

The HSD and UDD commands are similar as in the previous cases with the following differences:

HSD command '82' means READ DATA and COMMAND CHAIN

UDD command '10' means READ FILTERED IMAGE FROM CONVOLVER and the transfer count is '2000' (HEX) which is 8K words.

APPENDIX - (C)

DIRECTORY . . . . . . FILTER

FILE	DESCRIPTION
CONFIL	Program to filter a given image using the NTT convolver.
CONTMP	Program for template matching using the NTT convolver.
ECILUT	Program for error correction and index look-up table.
FILRTD	Program for two dimensional convolution using number theoretic transform. (A simulation of the hardware.)
FILTSK	A separate task for NTT convolution. Can be activated by another task in a multi-task environment.
FLTSK1	A sample task to activate the task for convolution using NTT convolver.
FLRESD	Program to store the input and output to the butterfly unit at each stage.
GTOPRM	Program for reading data from the PROM programmer.
HXTOIN	Program for converting hexadecimal values into integer values used in transfer of data from NOVA TO SEL.
INSBLK	Program to insert a blank in the first column used in transfer of data from SEL to NOVA.
LOGBOOK	This file gives us the name and description of all the programs in the DIR filter.
NORMAL	This program normalizes the output of the convolver between levels 0 and 255.
NTFLCF	Program to find the NTT of filter coeffs.
OULPWORD	This program filters 256 * 256 image using overlap-save method of convolution and the NTT convolver.
SCLCOF	To scale the given filter coeffs. so that no overflow occurs.

APPENDIX - (C) (Continued)

SDOPRM	Program to send data to the PROM-Programmer.
XFERHXNS	Program for transferring HEX data from NOVA to SEL.
XFERNS	Program to transfer ASCII data from NOVA to SEL.
FALFA	To find the cyclic group generator Alpha for a given modulus.
FMINU	To find the multiplicative inverse of a given number with respect to a given modulus.
FFTTMP	Program for template matching using fast fourier transform.
NTTTMP	Program for template matching using number theoretic transform. (Software).
RIDCAL	Program for an iterative thresholding.

APPENDIX (D)

A User Session With the Convolver

A user of the convolver is assumed to be familiar with SEL-32/27 computer and how the programs are executed. An image of size (128 x 128) with 8-bit representation is assumed to be prestored in SEL computer in Directory "IMAGE". Based on the specific requirement, a two-dimensional filter of relatively smaller kernel (say, 17 x 17 size) is designed and stored in a file in Directory "FILTER". Let us call the image file as "IMAGEA" and the coefficients file as "COEFFA".

When SEL computer is logged-on it prompts with Task System Manager (TSM). The user must change the directory by executing

TSM> DIRE = FILTER ⤶ .

Now the user is in "FILTER" Directory. A program named "FILTER" is a command processing file in this directory. If this command processing file is to be used to obtain the filtered image, one must issue the following command - TSM> FILTER IMAGEA COEFFA OUTPUTA ⤶

The processing follows the following steps:

1) The filter coefficients are scaled by execution of program "SCLCOF". The input to this program is file "COEFFA" while the output file is "SCOEFF".

2) The NTT of the filter coefficients is obtained by the execution of program "NTTCOF". The input to this program is the file "SCOEFF" and the output file is "NTCOEFF".

The files "SCOEFF" and "NTCOEFF" are temporary files and are scratched at the end of their use. One may not need to worry about them.

3) The filtering operation is achieved by executing the program "CONFIL". The files "IMAGEA" will be taken as input image and the file "NTCOEFF" will be taken as the file containing NTT of the filter coefficients. The IMAGE and NTT of coefficients are sent to the convolver, and at the end of processing the filtered image is sent back to computer where it is stored in a file named "OUTPUTA".

The above steps can also be executed in a serial fashion without the use of the command processing file "FILTER". In this case one must be in directory "FILTER". This could be done by executing

TSM> DIRE = FILTER

One must then create files "SCOEFF" and "NTCOEFF" by issuing the following command.

TSM> CREATE SCOEFF

TSM> CREATE NTCOEFF

Next the file "COEFF" is assigned as logical file Code = 1 and the program "SCLCOF" is executed as:

TSM> AS 1 = COEFFA

TSM> SCLCOF

Now, the NTT of the coefficients is obtained by issuing the following command:

TSM> NTTCOF

The filtering is obtained by execution of program "CONFIL". The file having the image to be filtered and the output file are assigned and filtering command is issued:

TSM> AS 1 TO @ DP1 (IMAGE) IMAGEA

TSM> AS 2 = OUTPUTA

TSM> CONFIL

Note that since the image was stored in "IMAGE" directory (which is different than the current default directory), it was necessary to represent it by @ DP1 (IMAGE) IMAGEA where "@ DPI (IMAGE)" lets the computer use "(IMAGE)" Directory. The file "OUTPUTA" is stored in "FILTER" Directory. This file is the filtered output.

The filtered image can be displayed by issuing the following command:

TSM> AS 1 = OUTPUT

TSM> DISPIMG

The program "DISPIMG" displays an image of size (128x128) on the Aydin Color Monitor. To display the input image, which was assumed to be "IMAGE" directory, the following commands have to be issued:

TSM> AS 1 TO @DP1(IMAGE)IMAGEA

TSM> DISPIMG

As described earlier, the first command lets the computer search the input file in "(IMAGE)" directory.

REFERENCES

1. L.R.Rabiner and B.Gold 'Theory and Applications of Digital Signal processing' Prentice-Hall, Englewood Cliffs, New Jersey, 1975

2. A.V.Oppenheim and R.W.Schafer 'Digital Signal Processing' Pretice-Hall Inc., Englewood Cliffs, New Jersey, 1974

3. N.S.Szabo and R.I.Tanaka 'Residue Arithmetic and its Applications to Computer Technology' Mc-graw Hill Book Company, New York, 1969

4. A.Peled and B.Liu 'Digital Signal Processing' John-Wiley and Company, New York, 1979

5. G.A.Jullien 'Implementation of Multiplication, Modulo a Prime Number, with Applications to Number Theoretic Transforms' IEEE Trans. on Computers, Vol.C-29, pp.899-905, Oct 1980

6. M.J.Corinthios 'A Fast Fourier Transform for High Speed Signal Procesing' IEEE Trans. on Computers, Vol.C-20, pp.843-847, Aug 1970

7. R.C.Agarwal and C.S.Burrus 'Number Theoretic Transforms to implement Fast Digital Convolution' Proc.IEEE, Vol.63, pp.550-560, April 1975

8. R.C.Agarwal and C.S.Burrus 'Fast Convolution using Fermat Number transforms with applications to digital filtering' IEEE trans.Acous.,Speech and Sign.Proc.,Vol.ASSP-22, pp.88-97,Apr.1974

9. J.H.McClellan and C.M.Radar 'Number Theory in Digital Signal Processing' Prentice-Hall Inc., New Jersey, 1979

10. H.K.Nagpal, G.A.Jullien and W.C.Miller 'Multidimensional Algorithms and Processor Architectures for Computing a class of Unitary Transforms'

11. G.A.Jullien and W.C.Miller 'A Hardware Realization of an NTT Convolver using ROM arrays' Proc.IEEE.ICASSP-Vol.3,pp.788-791,1980

12. M.J.Corinthios 'The Design of a class of Fast Fourier Transform Computers' IEEE Trans. on Computers, Vol.C-20, pp.617-623, June 1971

13. J.M.Pollard 'The Fast Fourier Transform in a Finite Field' Math.Comput., Vol.25, pp.365-374, April 1971

14. C.M.Radar 'On application of the Number Theoretic Transform methods of high speed convolution to 2-D filtering' IEEE Trans. Ciruits and Systems, Vol.CAS-22, June 1975

15. G.A.Jullein and W.C.Miller 'An RNS based FFT processor Hardware Implementation'

16. R.C.Gongalez and P.Wintz 'Digital Image Processing' Addison-Wesley Publishing Co., 1977

17. E.L.Hall 'Computer Imgae Processing and Recognition' Academic Press, 1979

18. B.R.Hunt 'Block-mode Filtering of Pictures' Math.Biosc., Amer.Elsevier Publ. Co., Vol.11, pp.343-354, 1971

19. R.H.Vander Kraats and A.N.Venetsanopoulos 'Hardware for two-dimensional Digital Filtering using Fermat Number Transform' IEEE Trans. on Acous.Speech and Sign.Processing, Vol.ASSP-30, April 1982

20. G.A.Jullien and W.C.Miller 'An RNS Arithmetic unit for 2-D Digital Convolution'

21. A.Baraniecka 'Digital Filtering using Number Theoretic Techniques' Ph.D.Thesis, University of Windsor, Windsor, Ont.,Canada, June 1980

22. H.Helms 'Fast Fourier Transform method of Computing Difference Equations and Simulating Filters' IEEE trans.,Vol.AU-15,pp.85-90,1967

23. J.W.Cooley and J.W.Tukey 'An algorithm for the Machine Calculation of Complex Fourier Series' Math.Comp., Vol.19, pp.297-301, 1965

24. C.S.Joshi, J.P.McDonald and R.H.Steinvorth 'A Video Rate Two Dimensional Fast Fourier Transform Processor' IEEE trans.,1979

25. B.Liu and A.Peled 'A new Hardware Realization of High speed Fast Fourier Transformers' IEEE Trans. on Acous. Speech and Sign.Processing, Vol.ASSP-23, pp.543-547, Dec 1975

26. H.K.Nagpal 'Processor Architectures for Fast Computation of Multi-Dimensional Unitary Transforms' Ph.D.Thesis, University of Windsor, Windsor, Ont.,Canada,1981

27. G.A.Jullien 'Residue number Scaling and other Operations using ROM arrays' IEEE Trans. on Computers, Vol.C-27, pp.325-336, April 1978

28. D.L.Dietmeyer 'Logic Design of Digital Systems' Allyn and Bacon, Boston, 1978

29. ---- 'Project and Research Applicable in Industry (PRAI) Grant Report' University of Windsor, Windsor, Ontario, Canada, 1982 (Project no.-7908, NSERC File no. 612-14/79)

30. ----- 'SEL-32/27 Manuals'

31. Chao-Huan Huang 'On the RNS and its Application to Computer Architectures and Fast Spectral Transforms' Ph.D.Thesis, University of Cincinnati, 1979.

32. R.J.Karwoski 'An Introduction to Digital Analysis including a high speed FFT Processor Design' TRW LSI Product, El Segundo, Calif., June 1980.

Rajendra Prasad Rathi
(B. - 29 Nov. 1959)

VITA AUCTORIS

EDUCATION

1974 Graduated from S.S.N. High School, Biratnagar, Nepal (School Leaving Certificate Examination conducted by the Board of Education, Govt. of Nepal, Kathmandu).

1976 Graduated from M.M.A. College with Merit in Certificate Level in Science Examination conducted by the Tribhuvan University (T.U.), Kathmandu, Nepal.

1982 Graduated with B.Sc. (Engineering) (Hons,) Degree in Electronics and Communication Engineering from the Kurukshetra University, Haryana, India with the First Rank in the University.

1984 Candidate for Master in Applied Science (M.A.Sc.) Degree in Electrical Engineering at the University of Windsor, Windsor, Ontario, Canada.

PROFESSIONAL AFFILIATION

Associate Member of Institution of Engineers (AMIE), India.

Student Member of Institution of Electrical and Electronics Engineers (IEEE), New Jersey, USA.

Member of Association of Students Council, Toronto, Ontario, Canada.

MAJOR HONORS AND AWARDS

1983-1985 University of Windsor Merit Scholarship.

1982-1984 A.N.E. Foundation (Oregon), USA, Scholarship Award.

1983 Associate Member of Institution of Engineers (AMIE) as an Honor to Gold Medalist in B.Sc. (Engineering).

1982 Gold Medal from the Kurukshetra University given to First Rank holder in professional education.

1977-1982 TSC of Colombo Plan Scholarship for undergraduate study in electronics and communication engineering (Government of Nepal).

CO-CURRICULAR ACTIVITIES

1982-1983 Graduate students representative at the University of Windsor.

1981-1982 Secretary of the Students Chapter, Institution of Engineers (Electronics and Telecommunication)

1980-1981 Associate Editor of the Monthly College Magazine "THE VOICE".

1979-1982 Editor of the Annual College Magazine "THE VULCAN".

PROFESSIONAL WORK EXPERIENCE

1982-1984 Teaching and Research Assistant at the University of Windsor, Windsor, Ontario, Canada.

1984- A member of Electrical Engineering Analytical Design Analysis group at Buick Motor Division, GMC, Flint, Michigan, USA.