1982

Implementation of an RNS based sequential NTT convolver.

Pramod B. Modak
University of Windsor

CANADIAN THESES ON MICROFICHE

I.S.B.N.

THESES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

NL-339 (r. 82/08)

Canada

IMPLEMENTATION OF AN RNS BASED

SEQUENTIAL NTT CONVOLVER

by

© Pramod B. Modak

A Thesis
Submited to the Faculty of Graduate Studies
through the Department of Electrical Engineering
in Partial Fulfillment of the requirements
for the Degree of Master of Applied Science at
The University of Windsor

Windsor, Ontario, Canada

1982

ABSTRACT

In this thesis, the design of a sequential NTT processor is proposed. The main elements of the processor are

(i) memory buffers

(ii) computational element

(iii) complex multiplier.

The computational element performs the NTT butterfly calculations. The complex multiplier performs multiplication of the NTT of the input sequence with the NTT of the filter impulse response. The complex multiplier is implemented using arrays of ROM and table look-up techniques resulting in a simple pipeline structure. All arithmetic operations involved in the multiplier are carried out using the RNS in order to achieve multiplication throughput rate of 3.5 MHz. The Ordered Input Ordered Output algorithm has been used to design memory structure of the convolver. The memory buffers store the input and the convolved output of the processor. The memory buffers are implemented using sequentially accessed memories.

The prototype was constructed by using slow speed, off the shelf components and tested.

ACKNOWLEDGEMENT

I would like to express my sincere thanks and appreciation to my supervisor, Dr. W.C. Miller for many valuable discussions and constructive criticism. I am also thankful to Dr. G.A. Jullien for his advice, help and constant encouragement throughout the study period. In addition, the help of Mr. J.M. Novosad, Mr. A. Thibert and Mrs. S.A. Ouellette is sincerely appreciated.

To my parents and brother, I extend my sincerest thanks and gratitude. Without their help and inspiration, though far away, this work would not have started.

I wish to express my sincere thanks to Dr. H.K. Nagpal for many helpful discussions.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

L	Number of moduli
M	Main modulus
m_i	i^{th} sub-modulus
$\|X\|_{m_i}$	Reduction of X modulo m_i
$\left\|\frac{1}{x}\right\|_{m_i}$	Multiplicative inverse of x moduli m_i
$\dot{x}$	Additive inverse of x modulo m_i
ROM	Read Only Memory
IND	Index Look-up Table
IND30, IND31	Index Look-up Table at sub-moduli 30 and 31
INV	Inverse Index Look-up Table
INV30, INV31	Inverse Index Look-up Table at sub-moduli 30 and 31
η	Primitive root of prime m_i
ADD30, ADD31	Addition look-up table at sub-moduli 30 & 31
SUBM1, SUBM2	Sub-moduli
SRi	i^{th} serial shift register
ACC	Accumulator register
α	Cyclic group generator
$GF(m_i)$	Galois Field of first order
$GF(m_i^2)$	Galois Field of second order
T_N	NTT transformation matrix of size (N x N)
P_N	Permutation matrix of size (N x N)
$A \otimes B$	Kronecker product of two matrices A and B
D_N	Quasi-diagonal matrix of size (N x N)

I_N	Unity matrix of size (N x N)
μ_i	Cyclic group generator of i^{th} stage
S_i	Transform operator of i^{th} stage
SR	Shift register
MM, BUF	Memory buffer
SM1, SM2	Sub-memories
REC	Reconstruction table
Re(X)	Real part of X
Im(X)	Imaginary part of X
EPROM	Erasable programmable read only memory
FIFO	First-in First-out memory
DNTT/FNTT	Forward NTT
INTT	Inverse NTT

CHAPTER 1

1. INTRODUCTION

1.1 Preamble

This thesis describes a hardware realization of memory buffers and an external complex multiplier for a sequential NTT processor. The work forms part of a more general development of a digital signal processing facility that is being constructed by the Signal and Systems Laboratory at the University of Windsor. The author's responsibility in this project was to design a memory structure which along with a radix-2 butterfly unit can provide the basis for a digital filtering capability.

1.2 Digital Convolution

Finite digital convolution has many practical applications in digital signal processing. It can be used to implement finite impulse response (FIR) filters, computation of auto and cross correlation and polynomial multiplication. It can be computed directly in time domain, called as a direct convolution or with transform that has the cyclic convolution property (CCP).

The Discrete Fourier Transform (DFT) exhibits the CCP over infinite complex field. The DFT becomes very attractive to use as it can be implemented efficiently using Fast Fourier Transform (FFT) algorithm. The two main disadvantages associated with DFT are multiplication by irrational coefficients and the inherent number growth. Both of

the above factors introduce truncation and round-off errors when DFT is implemented on a finite wordlength machine.

Pollard [7] has shown that the transform defined in a finite ring also exhibit the cyclic convolution property, which is known as Number Theoretic Transform (NTT). The NTT is defined as

$$X(k) = \left| \sum_{n=0}^{N-1} x(n) \cdot \alpha^{nk} \right|_M , \quad k = 0, 1, \ldots, N-1 \qquad (1.1)$$

where α is the cyclic group generator of order N. As the transforms are defined in finite rings, the number growth problem is inherently solved. The M must be large enough. so that the result of convolution is within defined range. The modulo M can be decomposed in number of primes m_i such that

$$M = \prod_{i=0}^{L-1} m_i \qquad (1.2)$$

and transform can be computed modulo several primes and results can be reconstructed at modulo M by using the Chinese Remainder Theorem (CRT) [1]. As NTT has the similar structure as that of DFT, the fast algorithm to compute DFT can be also used to compute NTT and will be called as FNTT.

The proposed implementation of the NTT processor requires a computational unit known as butterfly unit and a supporting memory organization. The computational unit involves arithmetic operation of addition, subtraction and multiplication and can be implemented using arrays of Read Only Memory (ROM) and table look-up techniques.

The memory buffers is an important element of an NTT processor. The memory buffers store the input/output sequences of processor and also the intermediate results of NTT calculations.

1.3 Objective and Outline of the Work

The NTT processor can be implemented with

i) a parallel structure

ii) a cascade structure

iii) a sequential structure

These structures have their own advantage and disadvantages as far as the operational speeds and the hardware requirements are concerned. Table (1.1) gives the computational unit requirements and operational speed for these structures.

	Processor Structure	Butterfly Units	Butterfly Operation Performed In		Execution Time
			Parallel	Sequential	
1	Parallel	N/r	N/r	m	T · m
2	Cascade	m	m	N/r	T · N/r
3	Sequential	1	1	N · m/r	T · N · m/r

N = number of points in input sequence

m = $\log_r N$, i.e., number of stages

r = radix of BCU

T = time required to perform butterfly computation

TABLE (1.1) Comparison of NTT Processor Structures

The parallel or cascade structure call for large number of computational units than sequential structure and would be proved economical in processing large data arrays. In [9]

Pease has described algorithm suitable for parallel structure. For processing of smaller data arrays and throughput rate requirements up to 3.5 MHz, the use of sequential processor is the most economical.

The main objective of work undertaken was to implement the memory buffers for radix-2 sequential NTT processor. The present work was divided into three phases. The first phase of the work consisted of a literature survey to establish theoretical basis for the design of sequential NTT processor. Corinthios in [10] - [13] has proposed a number of algorithms for the implementation of a serial sequential FFT processor, which can be used for NTT processor. The first algorithm, 'Post-Permutation Algorithm', is suitable for the applications where ordered set of NTT coefficients are not required. The second algorithm, 'Ordered Input Ordered Output (OIOO) Algorithm', which yields an ordered set of Output coefficients. The OIOO algorithm is suitable to implement a sequential NTT processor as the additional hardware for shuffling the output data is not required. The OIOO algorithm involves the application of three matrix operators on input data. The first operator computes sum and difference of equally spaced pairs of input data points, the second one performs complex multiplication on a fixed lower half of the data set and the third is an elementary permutation of data. The OIOO algorithm was used to implement proposed NTT processor.

The second phase of work was to implement the memory buffer and to design the complex multiplier, to perform multiplication of NTT of input sequence and filter coefficients. The design of various multiplexing units (a distributor unit, an output multiplexing unit and a BCU input multiplexing unit) was also done. A computer simulation of the hardware implementation of the memory structure, the complex multiplier, the various multiplexing units and the butterfly unit was carried out on IBM 370/3031.

The final phase of the work was to build the hardware for memory buffers, complex multiplier and various multiplexing units. These units along with the butterfly unit were tested by convolving number of input sequences with impulse response of lowpass filter for real time applications.

1.4 Thesis Organization

Chapter 2 provides a review of Residue Number System, Number Theoretic Transforms and different algorithms to be used to implement a sequential NTT processor. The advantage of using the RNS for a look-up table implementation, particularly multiplication, is established. The availability of NTT for error free convolution is also discussed. The use of OIOO algorithm to realize sequential NTT processor is described in detail.

In Chapter 3 hardware realization of various elements of a sequential NTT processor is presented. The simulation details of proposed processor structure on IBM 370/3031 are

also included.

In Chapter 4, the step-by-step design procedure to implement the memory buffers, the complex multiplier and various multiplexing unit is explained. The relevant material about generation of look-up tables for 2708 EPROMs using Intel's 220 development system is also included.

In Chapter 5, the test procedure for the verification of the NTT processor is described. The processor was tested by convolving number of waveforms with the impulse response of lowpass filter. The results obtained are also included.

Chapter 6 presents the conclusions that can be reached regarding this work.

CHAPTER 2

RESIDUE NUMBER THEORY AND NUMBER THEORETIC TRANSFORM

2.1 INTRODUCTION

The fast convolution has a large number of applications in the area of digital signal processing. In this chapter we have discussed the use of Residue Number Theory and Number Transform for fast convolution. The algorithms used to implement this are also described.

2.2 RESIDUE NUMBER SYSTEM

In the past, most of the research work concentrated on the use of the binary number system in the hardware realization of signal processing algorithms. In most of the algorithms, the operation of multiplication is the most time-consuming and many forms of binary multiplier structure have been proposed. The high speed binary multipliers are much more expensive compared to adder and subtractors of the same speed.

The Residue Number System (RNS) has received considerable attention for the hardware realization of digital signal processing elements. The main advantage of the RNS, compared to the binary number system, lies in the fact that the arithmetic operations within the dynamic range, can be performed using independent and parallel paths. The dynamic range can be varied by varying the number of parallel paths. With the current advancements in semiconductor memories and

and the continual decreasing in memory prices, the arithmetic operations of addition, subtraction and multiplication can be performed at very high speeds using look-up table techniques.

Another advantage of the RNS is its adaptability to look-up table implementation. In the binary number system, the look-up table approach is not feasible because of the enormous storage required. For a wordlength of B bits, 2^{2B} entries would be required in the table; however in the RNS with a comparable range, ie, $\prod_{i=1}^{L} m_i \approx 2^B$, each modulus - m_i requires m_i^2 entries in the table, hence a total of $\sum_{i=1}^{L} m_i^2$ entries are needed.

2.2.1 Residue Representation and Residue Arithmetic

A number in the RNS is represented by the L-tuple

$$X = (x_0, x_1, \ldots, x_{L-1}) \qquad (2.1)$$

with respect to the set of moduli $(m_0, m_1, m_2, \ldots, m_{L-1})$, where x_i is the i^{th} residue of X modulo m_i and is written as $x_i = |X|_{m_i}$. If all the moduli are relatively prime then we can have unique representation of each number in the range

$$0 \leq x \leq \prod_{i=1}^{L-1} m_i$$

The binary operation of addition or multiplication of two numbers X and Y, can be performed by independent operations on their respective residues, ie,

$$Z = |X \circledast Y|_M \text{ implies } z_i = |x_i \circledast y_i|_{m_i} \qquad (2.2)$$

where $\circledast$ denotes addition or multiplication. Hence the

operation of addition or multiplication can be implemented in L parallel paths. Addition and subtraction have no interdigit carries or borrows and multiplication does not need the generation of partial products, hence fast operating speeds can be obtained.

A signed integer system can be developed by considering the numbers in the interval $(0, \frac{M}{2} - 1)$ for even M or $(0, \frac{(M-1)}{2})$ for odd M as a positive, and the numbers in the internal $(\frac{M}{2}, M-1)$ for even M or $(\frac{(M+1)}{2}, M-1)$ for odd M as a negative. The additive inverse of number X is given by $\dot{X} = M-X$ and $\dot{x}_i = m_i - x_i$ for individual residues.

If $0 < X < M$, $(X, M) = 1$, then W is called the multiplicative inverse of X if $|W \cdot X|_m = 1$ and is denoted by

$$W = \left| \frac{1}{X} \right|_M \tag{2.3}$$

e.g., if $M = 7$, $X = 4$ then $W = 2$.

The division of two number X & Y at modulo M can be written as

$$\left| \frac{X}{Y} \right|_M = \left| |X|_M \cdot \left| \frac{1}{Y} \right|_M \right|_M \tag{2.4}$$

i.e., the division can be carried out by multipling X with the multiplicative inverse of Y. If $\frac{X}{Y}$ is not an integer in the real number system, then division of X by Y is not defined in the RNS. e.g. if $X = 6$, $Y = 2$ and $M = 7$

then $\left|\frac{1}{2}\right|_7 = 4$ and $\left|\frac{X}{Y}\right|_M = |6.4|_7 = |3|_7$

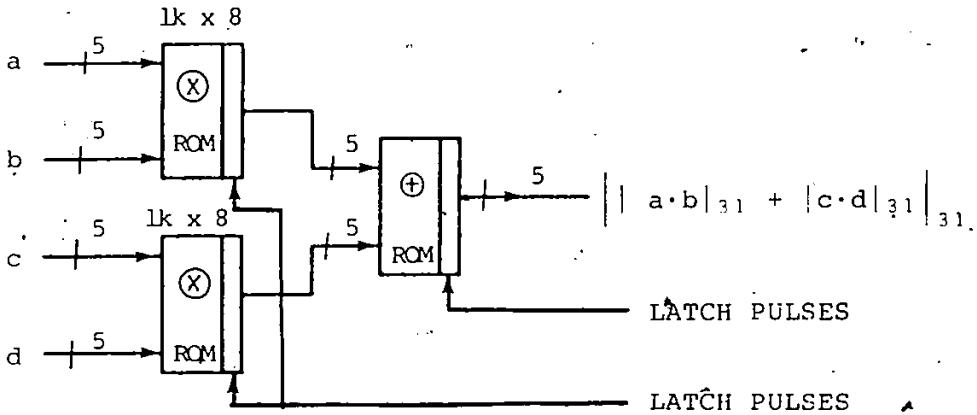


Fig. 2.1 A Pipeline ROM Array for Function $\left||a\cdot b|_{31} + |c\cdot d|_{31}\right|_{31}$

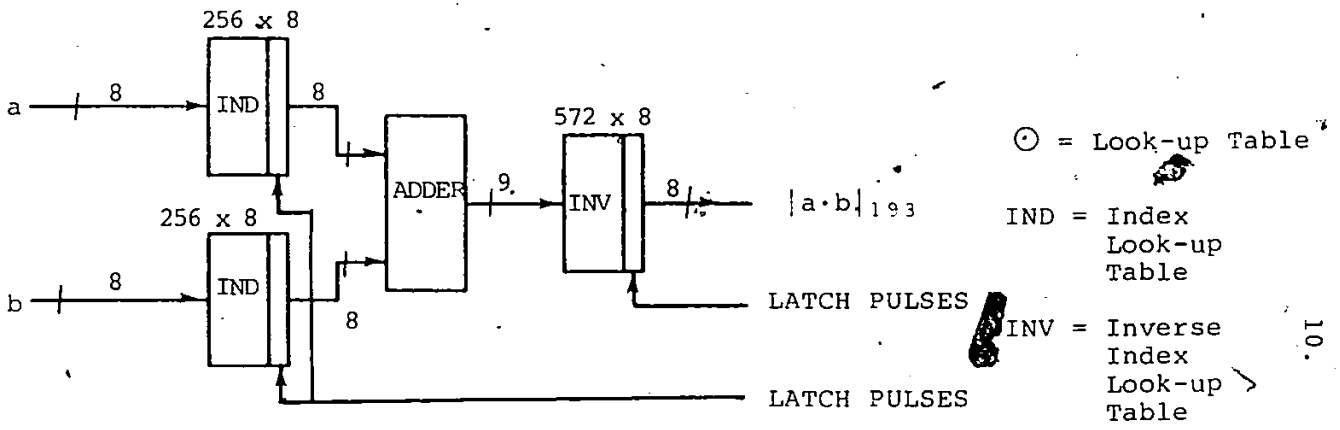


Fig. 2.2 Multiplication Using Isomorphism Technique

2.2.2 Hardware implementation of arithmetic operations in RNS

The basic approach of implementing an arithmetic operation in hardware can be divided into three main catagories as suggested in [1]. The first approach is direct logical implementation of the Boolean function related to the operation, the second requires storing of all the possible outcomes of the operation in look-up tables and the third is storing the software in general purpose computer. The second approach is becoming more and more attractive with the recent advancements in memory technology and the reduction in cost. It also offers the best solution for high speed realization.

A memory size of $(2^{2B} \times B)$ bits is required to store the look-up table for a modulus of B bits. For example, binary operation for moduli $m_i < 32$ can be implemented using the 1k x 8 bit a commercially available ROM package.

Moduli $32 \leq m_i < 64$ would require a storage of 4k x 8 or four 1k x 8 packages and moduli $64 \leq m_i < 128$ would require a storage of 16k x 8 or sixteen 1k x 8 packages.

The look-up table approach allows easy pipelining of an arithmetic operation. Fig.(2.1) shows the implementation of the function $\left| |a \cdot b|_{31} + |c \cdot d|_{31} \right|_{31}$. For every latch pulse, the output of each ROM is stored and becomes part of the address for the next ROM. A new input is thus presented and a result is generated for each ROM in one ROM access time plus the settling time of the latch. The only control signal required to clock the pipeline is a latch pulse.

For moduli $m_i > 128$, the look-up table technique calls for a large amount of memory. In order to increase the implementation efficiency, we use the isomorphism between a multiplicative group, Z, having elements, $\{z_n\} = \{1, 2, \ldots, m_i-1\}$, with multiplication moduli m_i, and the additive group, g, containing elements $\{g_n\} = \{0, 1, \ldots, m_i-2\}$ with addition moduli m_i-1, with m_i prime. The mapping is given by

$$z_n = \eta^{g_n} \tag{2.5}$$

where η is the primitive root of the prime, m_i; z_n is an element of the multiplicative group; and g_n is an element belonging to the additive group. g_n is also called the index of z_n to the base η and is denoted by $\text{IND}_{\eta}^{z_n}$.

Thus the multiplication of the elements in the multiplicative group can be done by adding the indices modulo (m_i-1), and taking the inverse index. Fig.(2.2) shows multiplication using the isomorphism technique. The addition can be done at any modulus, with condition that this modulus must be at least twice the original modulus. Hence standard binary adders can be used. The result is a larger modulus can be converted to the original modulus by using look-up tables.

If we are interested in having an all ROM structure, then the submodular ROM adder [2] can be considered. In this technique the main modulus is broken down into two relative prime submoduli, SUBM1 and SUBM2, with the condition that

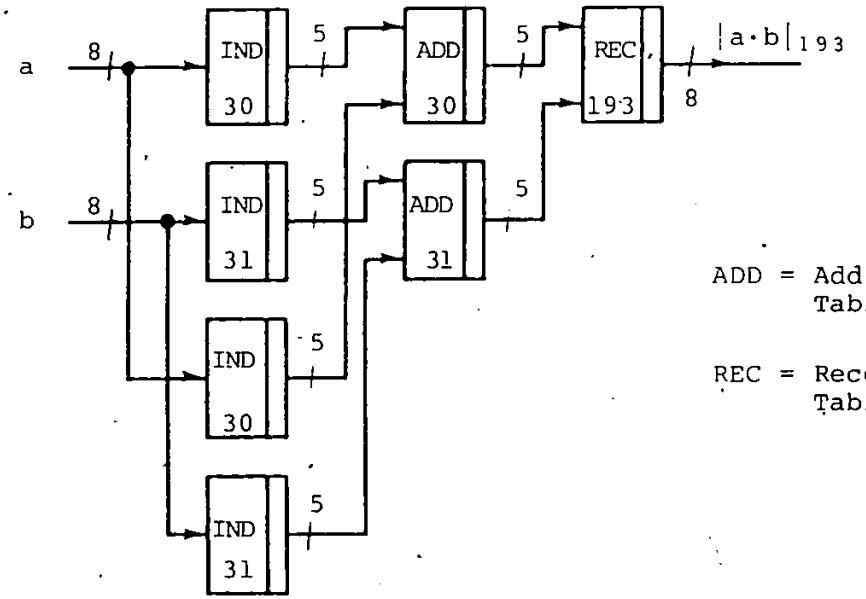


Fig.2.3 Multiplication Using Submoduli Approach

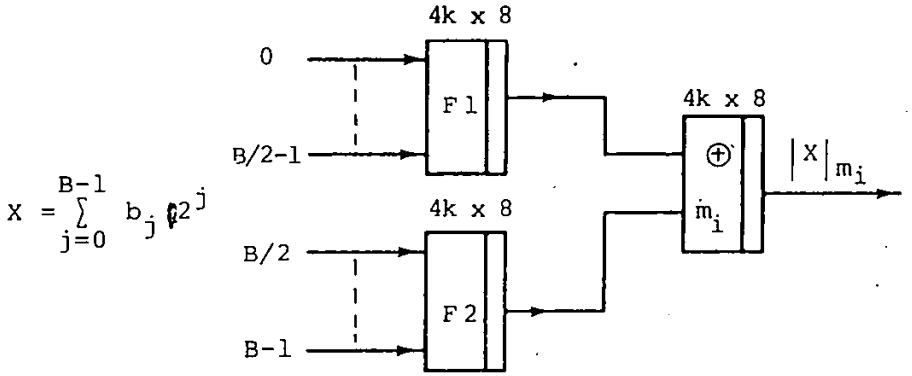


Fig.2.4 Binary to Residue Converter for $m_i \leq 64$ and $B \leq 24$

$$SUBM1 \cdot SUBM2 > 2 \cdot m_i \qquad (2.6)$$

The index, inverse index and addition tables are stored at moduli 30 and 31. The result is converted to the main moduli by using the Chinese Remainder Theorem (CRT). Fig.(2.3) shows the block diagram of the submodular multiplier for m = 193, SUBM1 = 30 and SUBM2 = 31.

2.3 BINARY TO RESIDUE AND RESIDUE TO BINARY CONVERSION

In order to interface digital structures using the RNS to conventional digital systems, it is necessary to have efficient residue input and output converters available.

A binary to residue converter (BRC) can be easily implemented using ROM arrays [3]. For example, in an L-moduli RNS, the i^{th} residue of a number X is given by

$$x_i = \left| \sum_{j=0}^{B-1} b_j \, 2^j \right|_{m_i} \qquad (2.7)$$

where b_j is the j^{th} bit of the binary representation of X. For B ≤ 12 bits and $m_i \leq 256$, equation (2.7) can be implemented using ROM package of 4k x 8 bit. For B > 12, we can split the binary representation of X into number of sections and the sum in each section moduli, m_i, is computed using a single ROM. The equation (2.7) can be written as

$$x_i = \left| \left| \sum_{j=0}^{\frac{B}{2}-1} b_j \, 2^j \right|_{m_i} + \left| \sum_{j=\frac{B}{2}}^{B-1} b_j \cdot 2^j \right|_{m_i} \right|_{m_i} \qquad (2.8)$$

Fig.(2.4) shows the block diagram of BRC for m_i < 64 and B 24 using three 4k x 8 bit ROMs.

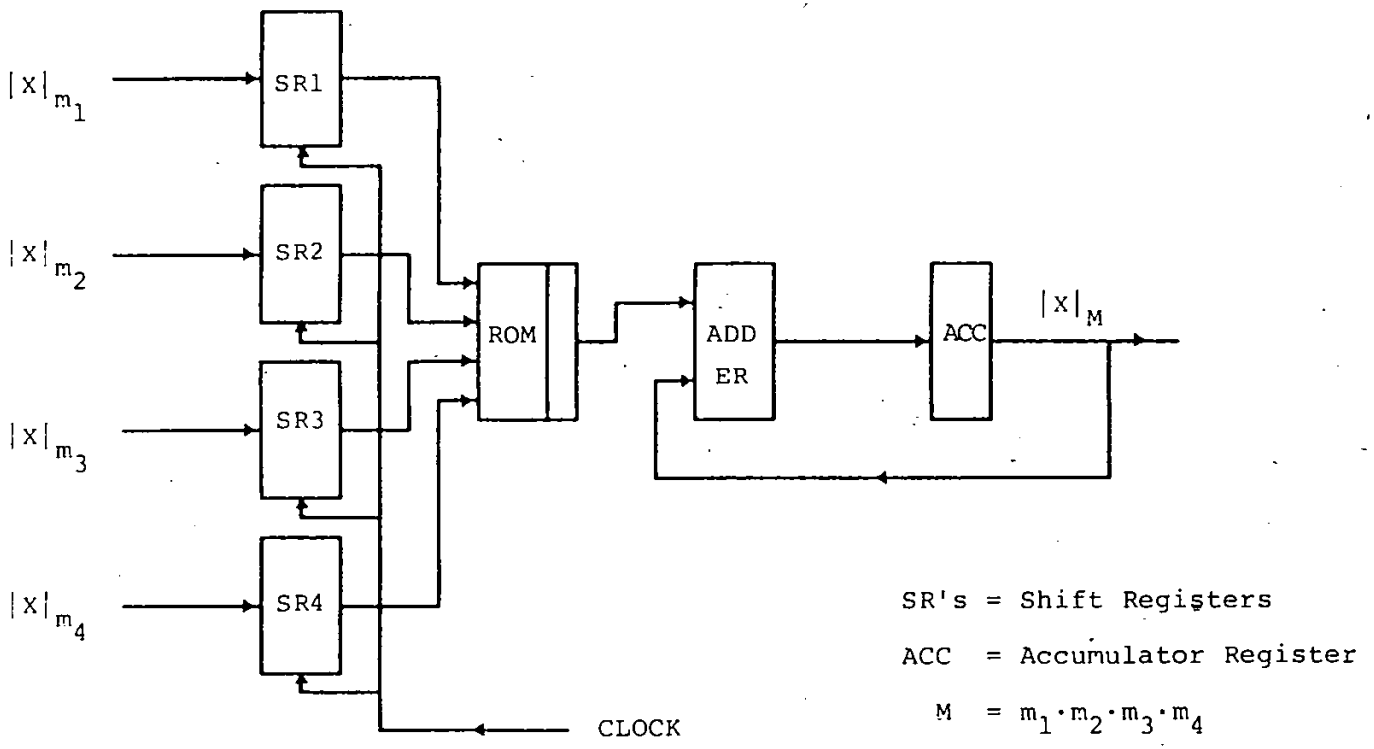


Fig.2.5 A Block Diagram of Implementation of the Chinese Remainder Theorem

The conversion from residue to binary is more difficult than binary to residue. There are two methods for RBC based on the Chinese Remainder Theorem (CRT) and a mix radix conversion. By CRT, a number X in range 0 to M-1 is given by

$$X = \left| \sum_{i=0}^{L-1} \hat{m}_i \left| \frac{X_i}{\hat{m}_i} \right|_{m_i} \right|_M \qquad (2.9)$$

where $\hat{m}_i = \frac{M}{m_i}$, $M = \prod_{i=0}^{L-1} m_i$ and $\left|\frac{1}{\hat{m}_i}\right|$ is the multiplicative inverse of $\hat{m}_i$. The hardware implementation of CRT is based on bit slice technique developed by Peled and Liu [4]. It requires modulo M adder which for a large value of M is difficult to implement. Fig.(2.5) shows the block diagram of CRT.

For high speed implementation of RBC, the mixed-radix technique is normally used. In mixed-radix form, a number in the range 0 to M-1 has the representation

$$X = \sum_{i=0}^{L} a_i \, p_i \qquad (2.10)$$

where $p_o = 1$, $p_i = \prod_{k=0}^{i-1} m_k$ and the $\{a_i\}$ are mixed-radix digits in the range $0 \leq a_i < m_i$. The $\{a_i\}$ can be generated using look-up tables [3]. Fig.(2.6) shows the residue to mixed-radix converter using ROM arrays for a four moduli system. The function stored in the look-up tables are as follows.

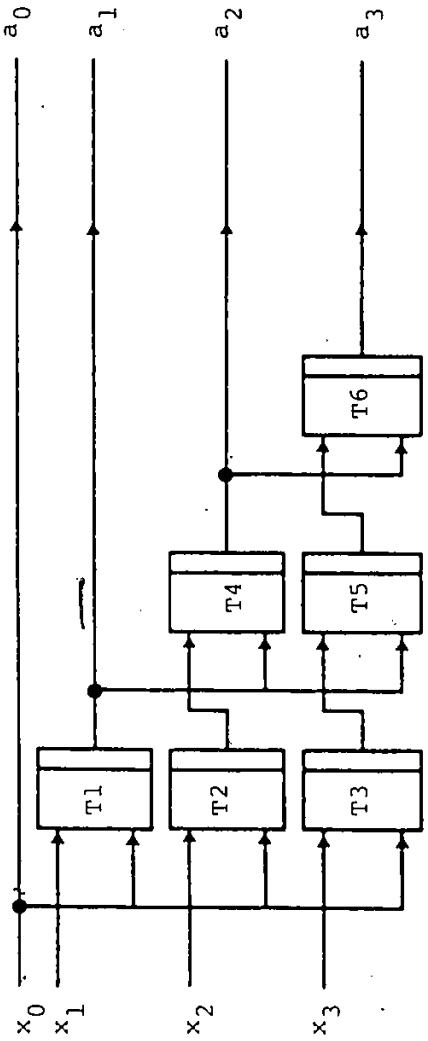


Fig.2.6 A Residue to Mixed-Radix Converter

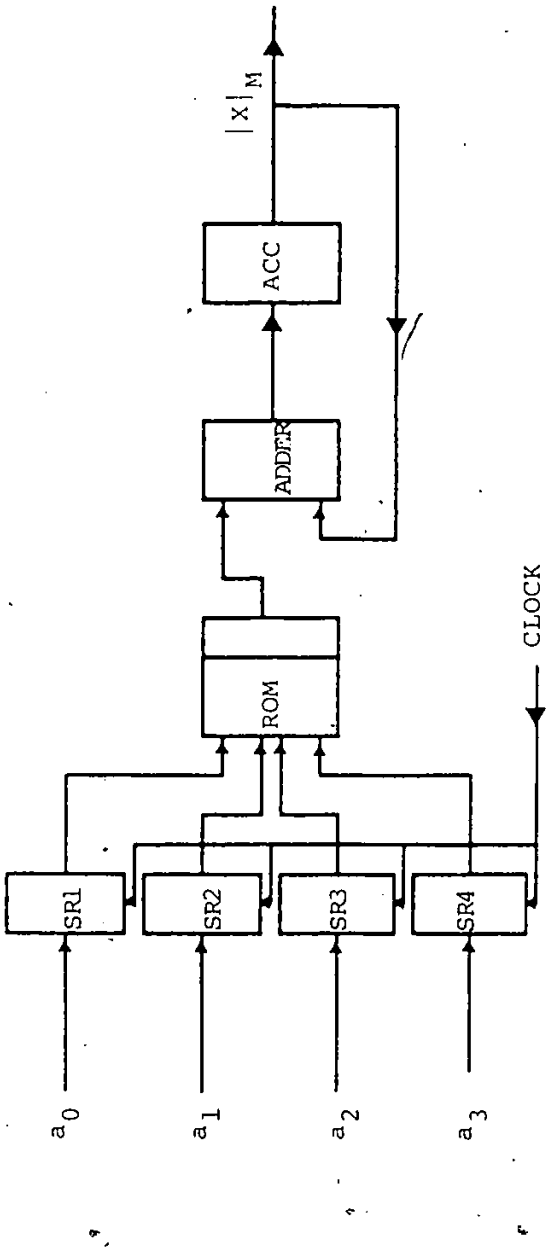


Fig.2.7 A Mixed-Radix to Binary Converter

$$
\begin{aligned}
T_1 &= \left| (x_1 - a_0) \cdot \left| \frac{1}{m_0} \right|_{m_1} \right|_{m_1} \\
T_2 &= \left| (x_2 - a_0) \cdot \left| \frac{1}{m_0} \right|_{m_2} \right|_{m_2} \\
T_3 &= \left| (x_3 - a_0) \cdot \left| \frac{1}{m_0} \right|_{m_3} \right|_{m_3} \\
T_4 &= \left| (T_2 - a_1) \cdot \left| \frac{1}{m_1} \right|_{m_2} \right|_{m_2} \\
T_5 &= \left| (T_3 - a_1) \cdot \left| \frac{1}{m_1} \right|_{m_3} \right|_{m_3} \\
\text{and} \quad T_6 &= \left| (T_5 - a_2) \cdot \left| \frac{1}{m_2} \right|_{m_3} \right|_{m_3}
\end{aligned}
\tag{2.11}
$$

The partial products $\{a_i\ p_i\}$ can be implemented using Peled and Liu's bit slice technique. This scheme does not require a Modulo M adder since the summation can be performed using a binary adder. Fig.(2.7) shows the block diagram of this scheme.

2.4 FAST CONVOLUTION

The computation of finite digital convolution

$$y(n) = \sum_{n=0}^{N-1} h(n-m)\ x(m) \tag{2.12}$$

symbolically denoted as

$$y(n) = h(n) * x(n) \tag{2.13}$$

has extensive applications in digital processing of signals. For example: in implementation of finite impulse response (FIR) filters, computation of auto and cross correlation, and polynomial multiplication. Hence efficient implementa-

tion of the convolution is essential.

The digital convolution can be implemented directly in the time domain, called direct convolution or with transform, T, that has the property of cyclic convolution (CCP).

The CCP can be denoted as

$$T[h(n) * x(n)] = T[h(n)] \cdot T[x(n)] \qquad (2.14)$$

where * denotes circular convolution and n = 0, 1, 2, ..., N-1.

This implies that the N-point convolution can be obtained by an inverse transform of the pointwise product of two vectors in the transform domain as follows

$$y(n) = T^{-1}\{T[h(n)] \cdot T[x(n)]\} \qquad (2.15)$$

The finite convolution can be calculated with the help of the circular convolution as in equation (2.15), by appending zero's in x(n) and h(n) to prevent folding or aliasing [5]. The technique of convolving two finite duration sequences, using transform techniques is known as a fast convolution.

The general form of the transform which maps circular convolution of length N sequences, over ring R, is given by

$$T: X(n) = \sum_{k=0}^{N-1} x(n)\, \alpha^{nk} \quad n = 0, 1, \ldots, N-1 \qquad (2.16)$$

where α is a primitive N^{th} root of unity in R.

The inverse transform is given by

$$T^{-1}: x(k) = N^{-1} \sum_{n=0}^{N-1} X(k)\, \alpha^{-nk} \quad k = 0, 1, \ldots, N-1 \qquad (2.17)$$

with the condition that N^{-1} belongs to R. The Discrete Fourier Transform (DFT) with $\alpha = e^{-j2\pi/N}$ is the only transform

having CCP over an infinite complex field. The Fast Fourier Transform (FFT) algorithm can be used to compute DFT. The efficiency of computation depends upon the degree of composition of the transform length N.

2.5 NUMBER THEORETIC TRANSFORM (NTT)

If we operate over a finite field, with arithmetic operation at modulo M and with special choice of N, α and x, which belong to the finite algebra system, then there exists a Number Theoretic Transform (NTT), which has the property of circular convolution.

The NTT has the following form

$$X(k) = \left| \sum_{n=0}^{N-1} x(n) \cdot \alpha^{nk} \right|_M , \quad k = 0, 1, \ldots, N-1 \qquad (2.18)$$

where M indicates modulo M reduction with prime modulus M. α is known as a primitive root of unity or a cyclic group generator of order N, i.e., N is the least positive integer such that

$$\alpha^N = 1 \qquad (2.19)$$

The inverse transform is given by

$$x(n) = \left| N^{-1} \cdot \sum_{k=0}^{N-1} X(k) \cdot \alpha^{-nk} \right|_M , \quad n = 0, 1, \ldots, N-1 \qquad (2.20)$$

where N^{-1} is known as a multiplicative inverse of N.

The NTT is used only to compute the convolution as the transform domain does not have any known practical value. The X(k) depends on the choice of α.

The important advantage of NTT over DFT is that the

computation of convolution is exact. That is, after the quantization of the input data and the filter coefficients, no additional quantization noise in introduced into the filtering process. In DFT the exact computation of convolution is not possible due to the presence of an irrational coefficient.

The disadvantage of NTT is that in the relationship of the sequence length N to the wordlength, longer wordlength is required for longer sequence length.

2.5.1 Conditions for existence of NTT

In [6], Nicholson has presented an algebraic theory for generalized transform having a DFT structure in a commutative ring with identity and without a zero divisor (i.e., an integral domain). If α and N are the cyclic group generator and transform length respectively, then the necessary and sufficient conditions for NTT to exist and to invertible are

i) α is a primitive root of unity in Ring, ie,
$\alpha^N = 1$ and $\alpha^\ell \neq 1$ for $\ell = 1, 2, \ldots, N-1$

ii) The multiplicative inverse of N exists and N^{-1} belongs to Ring.

If R is commulative ring with identity, the R can be decomposed uniquely as the direct sum of rings, R_i's as

$$R = R_1 \oplus R_2 \oplus R_3 \oplus \cdots \oplus R_L \qquad (2.21)$$

and under this decomposition every element r can be represented as $(r_1, r_2, \ldots, r_i)$. If R_i contains α_i of order N such that $\alpha^N = 1$ and N^{-1} belongs to R_i, then R supports NTT of length N.

2.5.2 NTT using RNS

We have seen from the equation (2.18), that modulo M must be large enough to avoid the overflow, since convolved output must remain in the interval

$$-\frac{M}{2} \leq y < \frac{M}{2} \quad \text{for M even}$$

$$-\frac{(M-1)}{2} \leq y \leq \frac{(M-1)}{2} \quad \text{for M odd}$$

Instead of using a single large modulus transform, the transform can be computed modulo several primes $\{m_i\}$, and the result can be reconstructed at modulo M by using CRT. This is similar to decomposing Z_M into the direct sum of ring Z_{mi}. This indicates that a transform of length N having the CCP in Z_m must also have CCP in Z_{mi}. The conditions for this are [8],

i) α, an integer of order N, must exist in Z_{mi}, such that $\alpha^N = 1$

ii) N^{-1} must exist in Z_{mi}.

The necessary condition for N to be possible transform length in Z_m is

$$N \mid \text{god}\ (m_0-1,\ m_1-1,\ \ldots,\ m_L-1).$$

2.5.3 NTT over the Galois Field of second degree $GF(m_i^2)$

Polard in [7] has derived the necessary and sufficient conditions for the transform to have CCP in Galois Field of m_i^n elements, $GF(m_i^n)$, with m_i prime. In [3] Baraniecka has discussed extensively the advantages and possibilities of NTT's over $GF(m_i^2)$. The NTT can be defined over $GF(m_i^2)$ as

$$T_{GF(m_i^2)}: X_i(\ell) = \sum_{j=0}^{N-1} x_i(j)\, \alpha_i^{j\cdot\ell}, \quad \ell = 0, 1, \ldots, N-1 \qquad (2.22)$$

where N must divide m_i^2-1 and $x(j)$, $X(\ell)$ and $\alpha_i \in GF(m_i^2)$. This increases the possibility of long transform lengths. By using RNS and $GF(m_i^2)$, we can compute the finite convolution in finite ring which is the direct sum of several $GF(m_i^2)$ as

$$R \simeq GF(m_1^2) + GF(m_2^2) + \cdots + GF(m_L^2) \qquad (2.23)$$

In [37] Baraniecka has discussed the method of determining different parameters of NTT. For implementing circular convolution in $GF(m_i^2)$, the following steps must be followed

i) choose the transform length N which is most suitable for the application.

ii) Choose the primes which can support this transform length in $GF(m_i^2)$.

iii) Construct the 2nd order field in which binary operations are simpler.

iv) Find out the generator, α, of order N in $GF(m_i^2)$.

Let $x(n)$, $h(n)$ be the input sequence and impulse response respectively. The circular convolution can be implemented by computing the transform of the block of input data of length N and of the impulse response $h(n)$, multiplying the transforms and taking the inverse transform. As we are using NTT over $GF(m_i^2)$, for convolution of real sequences, two successive blocks of the input sequence can be transformed

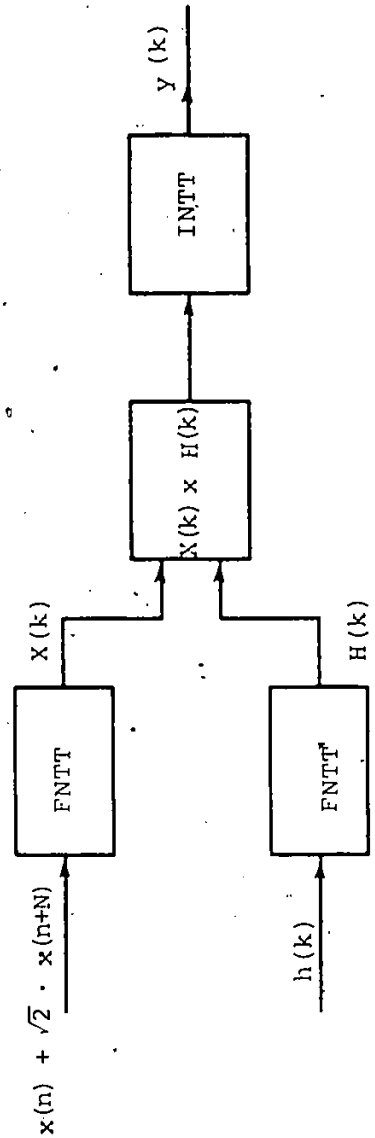


Fig.2.8 Convolution Using NTT Method

simultaneously via the same NTT by making

$$u(n) = x(n) + \sqrt{r} \cdot x(n+N)$$

where $u(n) \in GF(m_i^2)$ and $\alpha = \sqrt{r}$ is the cyclic group generator. Fig.(2.8) shows the block diagram of the convolution of real input sequence by NTT over $GF(m_i^2)$.

To compute the convolution unequivocally, the components of the circular convolution must remain in the interval

$$-\frac{(m_i-1)}{2} \leq y \leq \frac{(m_i-1)}{2} .$$

The absolute upper bound on the input data and the impulse response is

$$\max|x| \cdot \max|h| \leq \frac{m_i-1}{2N} \tag{2.24}$$

This bound on the dynamic range is pessimistic. For practical applications, if impulse response is known and fixed, it is sufficient to have

$$\max|x| \leq \frac{m_i-1}{2 \sum_{i=0}^{N} |h(i)|} \tag{2.25}$$

If input sequence consists of a set of positive numbers, then we can have

$$\max|x| \leq \frac{m_i-1}{\sum_{i=0}^{N} |h(i)|} \tag{2.26}$$

2.6 ONE-DIMENSIONAL NTT ALGORITHMS AND ITS IMPLEMENTATION

In the past, the computation of a DFT of an N-point sequence required N^2 operations and proved to be very time-consuming and uneconomical. Since the development of the

Fast Fourier Transform algorithm by Cooley and Tukey in 1965, a significant amount of research effort has been aimed towards finding efficient algorithms for the implementation of DFT processors. These algorithms can be used to implement an NTT processor.

In [9] Pease described a variation of the Cooley-Tukey FFT algorithm in which he first partitioned the DFT transformation matrix and then further factored it in terms of arithmetic and permutation operators. He has suggested the implementation of these algorithms for realization of a parallel hardwired processor. He has suggested the utilization of relatively slow memories and a number of arithmetic units operating in parallel for a radix-2 FFT processor.

Corinthios in [10] - [13] has proposed a number of algorithms for the implementation of a serial sequential processor. The first one is known as 'Post-Permutation Algorithm', suitable for the applications which do not call for an ordered set of output Fourier coefficients. The second one is 'Ordered Input Ordered Output,(OIOO) Algorithm'. It avoids the shuffling of the output data. Both the algorithms involve the application of three matrix operators on input data. The first operator calculates the sum and difference of equally spaced pairs of data points; the second operator performs complex multiplication on a fixed lower half of the data set; the third one is an elementary permutation of the data. In the fast permutation algorithm, the third operator is identical throughout all iterations. In

the OIOO algorithm the permutation varies with each iteration.

The OIOO algorithm suggested by Corinthois can be used to implement suitable memory structure for the NTT processor using sequentially accessed memories. The use of sequentially accessed memories leads to simple hardware for memory organization and for real time Input/Output control.

2.6.1 OIOO Algorithm

The NTT of an N-point sequence f_t over finite field is defined by

$$F_k = \sum_{t=0}^{N-1} \alpha^{t \cdot k} \cdot f_t \qquad (2.27)$$

where α is a cyclic group generator, F_k is the k^{th} coefficient of the NTT and module M reduction is assumed. The equation (2.27) can be written in matrix form as

$$F = T_N \cdot f \qquad (2.28)$$

where f and F are column vectors and T_N is N x N NTT transformation matrix and elements of T_N are given by

$$[T_N]_{txk} = \alpha^{t \cdot k} \qquad (2.29)$$

The T_N can be denoted by exponents of α to simplify the notation as

$$T_N = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & N-1 \\ 0 & 2 & 4 & 2(N-1) \\ 0 & N-1 & 2(N-1) & (N-1)^2 \end{bmatrix} \qquad (2.30)$$

The matrix T_N can be expressed in terms of a permutation matrix P_N and T'_N as

$$T_N = P_N \quad T'_N \tag{2.31}$$

$$T'_N = [P_N]^{-1} \; T_N \tag{2.32}$$

where P_N is the ideal shuffle permutation matrix, which is defined by its operation on the vector of dimension N, by relation

$$P_N \cdot \text{col}[x_o, x_1, .., x_{N-1}]$$

$$= \text{col}[x_o, x_{N/2}, x_1, x_{\frac{N}{2}+1}, x_2, x_{\frac{N}{2}+2}, \ldots, x_{N-1}] \tag{2.33}$$

and P_N^{-1} is a permutation operator which groups the even-odd elements together after operating on the vector of dimension N as

$$P_N^{-1}[x_0, x_1, x_2, \ldots, x_{N-1}] = [x_0, x_2, x_4, \ldots, x_1, x_3, \ldots] \tag{2.34}$$

when N is a composite number, i.e., $N = r^n$, then T'_N can be partitioned in (r x r) square submatrices, each of the dimension N/r and factored in a product of matrices including a matrix $T_{N/r}$. Furthermore $T_{N/r}$ can be partitioned and factored in terms of T_{N/r^2}. This process can be continued in order to get OIOO algorithm.

If $r = 2$ and $j = 2^i$, $i = 0, 1, 2, \ldots$

we can then write:

$$T_{N/j} = P_{N/j} \; (T_{N/2j} \circledast I_2) \; D_{N/j} \; (I_{N/2j} \circledast T_2) \tag{2.35}$$

where $\circledast$ represents the Kronecker product of matrices and

$$D_{N/j} = \text{quasi-dia} \; (I_{N/2j}, K_j) \tag{2.36}$$

$$\text{and} \quad K_j = \text{dia}(0,\ j,\ 2j,\ 3j,\ \ldots) \tag{2.37}$$

$$T_r = \begin{bmatrix} 0 & 0 & 0 & \vdots & \vdots & 0 \\ 0 & N/r & 2N/r & \vdots & \vdots & (r-1)N/r \\ 0 & 2N/r & 4N/r & \vdots & \vdots & 2(r-1)N/r \\ 0 & (r-1)N/r & 2(r-1)N/r & \vdots & \vdots & (r-1)^2N/r \end{bmatrix} \tag{2.38}$$

Using equation (2.35) and replacing $T_{N/j}$ by its factors in terms of $T_{N/rj}$, we can write the factorization of T_N as

$$T_N = \prod_{i=0}^{n} \mu_i \, S_i$$

where μ_i is a cyclic group generator operator specifies multiplications by α's and they are given by

$$\mu_i = (I_{2n-i} \circledast D_{2i}) \ , \ i = 2,\ 3,\ \ldots,\ n \tag{2.40}$$

and S_i is the transform operator given by

$$S_{i-1} = S \cdot P_i \tag{2.41}$$

with $$S_n = S^1 = (I_{N/2} \circledast T_2) \tag{2.42}$$

where P_i is the permutation operator defined by

$$P_i = (I_{2n-i} \circledast P_{2i}) \tag{2.43}$$

and $$P_1 = \mu_1 = I_N \tag{2.44}$$

also

$$S_{i-1} = PS^2 \tag{2.45}$$

where $$S^2 = (I_{N/4} \circledast T_2 \circledast I_2) \tag{2.46}$$

Substituting (2.45) in (2.39) and if we merge the permutation operations into computation ones, ultimately an ordered set of coeffidients can be obtained at output, which results in OIOO algorithm.

$$T_N = \prod_{i=0}^{n-1} P_i \cdot \mu_i \cdot S \qquad (2.47)$$

By substituting equation (2.47) into equation (2.27) we get

$$F = \left(\prod_{i=0}^{n-1} P_i \cdot \mu_i \cdot S \right) f \qquad (2.48)$$

The equation (2.48) indicates that the computation of F can be divided into n-stages, where each stage performs the computation specified by these operators. The operators for any stage operate on the output of the previous stage. For the first stage they operate on input vector f.

From equation (2.42) and equation (2.46), we can conclude that the transform operates S_i for first stage operates on the data, N/2 words apart and for subsequent stage it operates on the data, N/4 words apart. This data separation can be obtained by using 4 sequential access memories with a storage capacity of N/4 words each.

2.6.2 Example of OIOO algorithm for N = 16 and r = 2.

For N = 16, r = 2 and n = 4, we require four sequential access memories, each with a storage capacity of 4 words. The OIOO can be written as

$$F = P_3 \ \mu_3 S \ P_2 \ \mu_2 S \ P_1 \ \mu_1 S \ P_0 \ \mu_0 S^1 \ f \qquad (2.49)$$

Fig.(2.9) shows the memory organization and computational unit. The memory organization consists of two memories MM1 and MM2, each of them is further divided into two sub-memories SM1 and SM2. The sub-memories are further divided into two shift registers SR1 and SR2 as shown in Fig.(2.9).

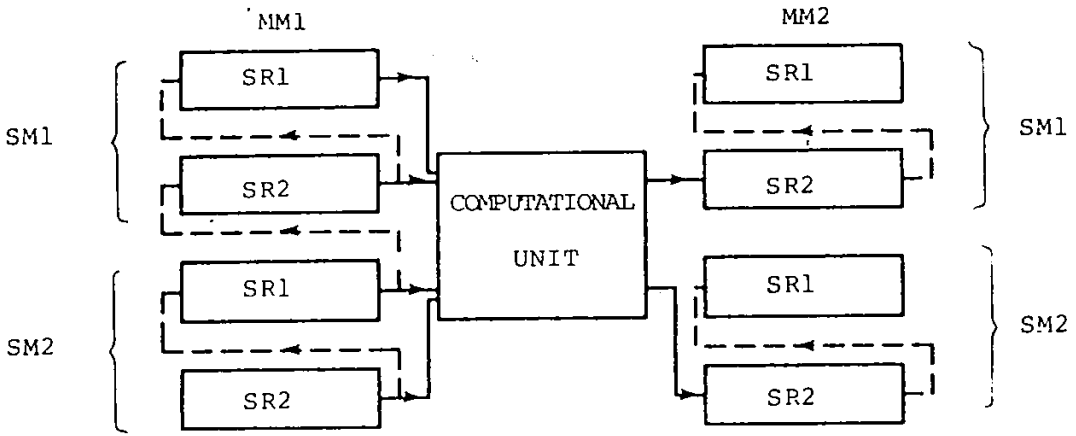


Fig.2.9 A Memory Organization and Computational Unit for N = 16 and r = 2

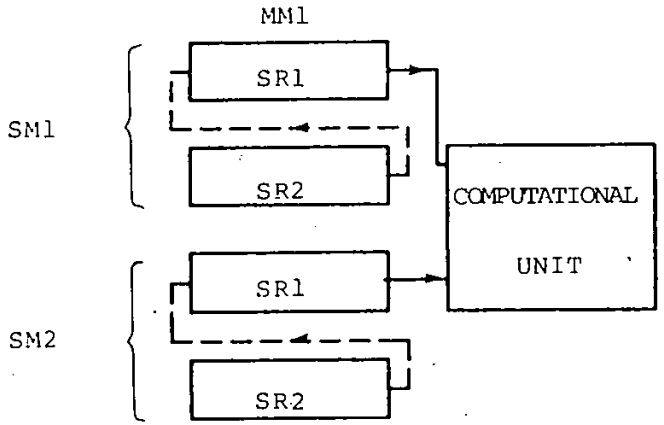


Fig. 2.10 Memory Interconnections for 1st Stage of NTT

The first stage is specified by the operator S^1, which is equal to $I_{N/2} \circledast T_?$. The sub-memory interconnections are as shown in Fig.(2.10). The SR1 and SR2 of sub-memories SM1 and SM2 are serially connected so as to form a shift register with a length of N/2 words. The data at the top of SM1 and SM2 is shifted in a computational unit and at the same time the data in SM1 and SM2 is shifted to the right so as to make it available for the next computation. The arrows shows the direction of data flow.

All other stages are specified by the operator S, which is equal to $I_{N/4} \circledast T_? \circledast I_?$. In each stage the SM1 and SM2 of the selected memory are selected in the following order: SM1, SM2, SM1, SM2, SM1, ... The number of butterflies selected for each sub-memory depends upon the permutation operator P_i and NTT stage. In general we can write the number of selected butterflies as $NB = 2^{j-2}$, j = 2, 3, ..., n-1. Fig.(2.11) shows the necessary sub-memory interconnections. The data at the top of the SR's of selected sub-memory is shifted into the computational unit.

The shuffle operation is to be performed while storing the output data from the computational unit. Fig.(2.12) shows the necessary connections. The SR1 and SR2 of sub-memories SM1 and SM2 are connected serially. The computation unit gives two output words at the end of every computation. Each word is shifted into each sub-memory as shown in Fig. (2.12).

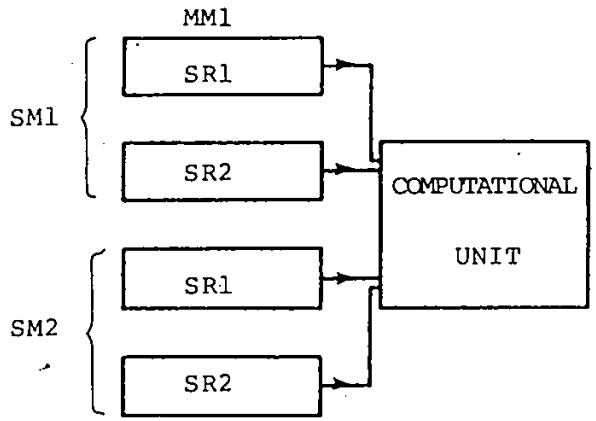


Fig. 2.11 Memory Connections for Stages 2 to N-1 of NTT

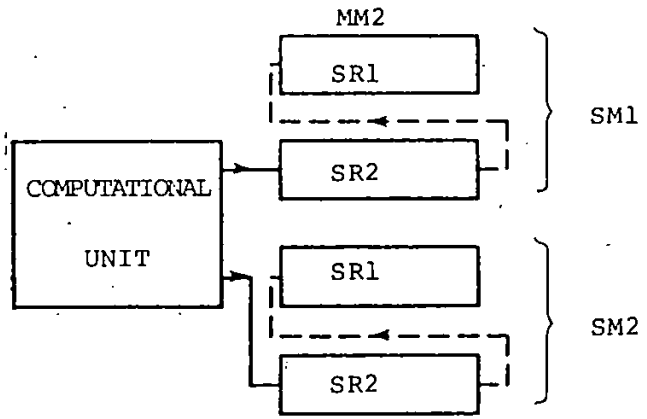


Fig.2.12 Memory Connections While Storing The Output

The operations specified by the transform operator μ_i are performed in the computational unit. The equation (2.40) tells how to store the cyclic group generators. The matrices in the Fig.(2.13a) and Fig.(2.13b) shows a typical order in which α's and α^{-1}'s are stored.

In the case of the OIOO algorithm the multiplication by α is done after a two point transform, hence the butterfly has the same structure as the DIF butterfly. Fig.(2.14) shows the flowgraph of OIOO for N = 16 and r = 2.

2.7 SUMMARY

In this chapter the basic concepts of the Residue Number Theory and the Number Theoretic Transform were discussed. The fast multiplication using the look-up table technique is described.

Finally different algorithms to implement an NTT processor were discussed. The implementation of OIOO algorithm for realization of the sequential processor is discussed in detail.

$$\begin{bmatrix} \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 \\ \alpha^1 & \alpha^0 & \alpha^0 & \alpha^0 \\ \alpha^2 & \alpha^2 & \alpha^0 & \alpha^0 \\ \alpha^3 & \alpha^2 & \alpha^0 & \alpha^0 \\ \alpha^4 & \alpha^4 & \alpha^4 & \alpha^0 \\ \alpha^5 & \alpha^4 & \alpha^4 & \alpha^0 \\ \alpha^6 & \alpha^6 & \alpha^4 & \alpha^0 \\ \alpha^7 & \alpha^6 & \alpha^4 & \alpha^0 \end{bmatrix}$$

(a) Position of α's for DNTT

$$\begin{bmatrix} \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 \\ \alpha^{14} & \alpha^0 & \alpha^0 & \alpha^0 \\ \alpha^{13} & \alpha^{13} & \alpha^0 & \alpha^0 \\ \alpha^{12} & \alpha^{13} & \alpha^0 & \alpha^0 \\ \alpha^{11} & \alpha^{11} & \alpha^{11} & \alpha^0 \\ \alpha^{10} & \alpha^{11} & \alpha^{11} & \alpha^0 \\ \alpha^9 & \alpha^9 & \alpha^{11} & \alpha^0 \\ \alpha^8 & \alpha^9 & \alpha^{11} & \alpha^0 \end{bmatrix}$$

(b) Position of α's for Inverse NTT

Fig.(2.13) The Order of Storing α's in the Computation Unit

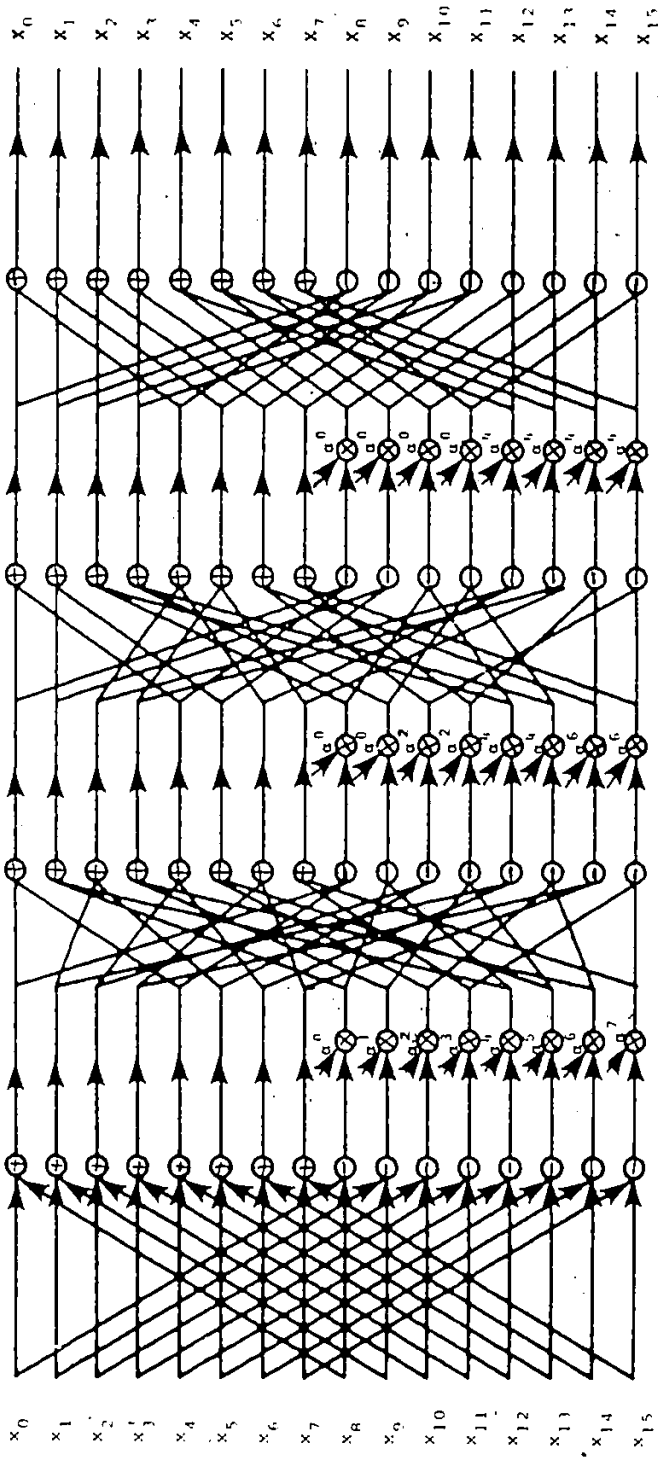


Fig. (2.14) Flowgraph of OIOO for N = 16 and r = 2

CHAPTER 3

REALIZATION OF A REAL TIME SEQUENTIAL NTT PROCESSOR

3.1 INTRODUCTION

The NTT processor mainly consists of a memory structure and a basic computational unit known as Butterfly Unit (BCU). The main aim of the work presented here is to realize the memory organization as well as the supporting structure of an NTT processor for real time applications with the constraint of minimum hardware requirements.

In this chapter, the modular design of an NTT processor is developed. The basic computational unit requirements are discussed but the actual design of the BCU is not undertaken. A memory structure suitable for radix-2 BCU was designed. The supporting structure consists of a Distributor Unit (DU), Input/Output multiplexing unit, Binary to Residue Converter (BRC), Residue to Binary Converter (RBC). These individual blocks of supporting structure were realized. The external serial complex multiplier for transform domain multiplication of two sequences was developed. The organization of a sequential NTT processor for filtering a real-valued input sequence is described.

3.2 NTT PROCESSOR

The NTT has the same structure as the DFT and therfore for highly composite transform length N, the fast algorithm for computing the DFT can also be used for computing NTT. The NTT processor can be implemented with a parallel, a

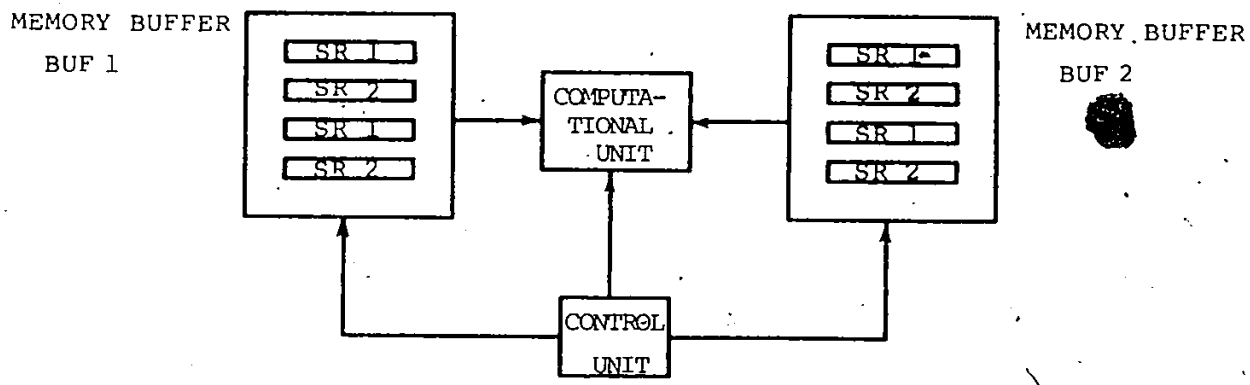


Fig. (3.1a) An NTT Processor Organization for OIOO Algorithm

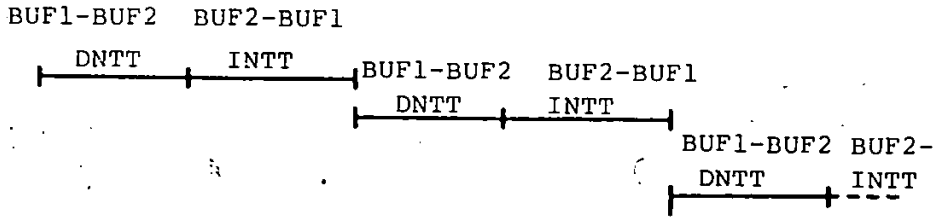


Fig. (3.1b) Timing Diagram for an NTT Processor

cascade or with a sequential structure. These structures have their own advantages and disadvantages as far as the operational speeds and the hardware requirements are concerned. The aim of the work undertaken was to implement the memory structure of an NTT processor with minimal hardware requirements. The OIOO algorithm suggested by Corinthois, as discussed in the previous chapter, is most suitable for the sequential realization of an NTT processor. A multiplexed radix-r butterfly can be used as the computational unit. It is accessed $\left(\frac{N}{r} \log_r N\right)$ times; where r is the radix of FNTT algorithm and N is transform length.

3.2.1 An NTT Processor for Non-realtime Applications

Fig.(3.1) shows a conceptual block diagram of an NTT processor consisting of a memory organization, computational unit and a control unit. The data flow between the memory and the computational unit is governed by the control unit and it also keeps track of the stage of computation. For non-realtime applications the memory organization can be realized with a two-buffer structure, BUF1 and BUF2, as shown in Fig.(3.1). When BUF1 contains the input data of a stage, BUF2 will store the output of that stage and their roles will be reversed as the computation continues in next stage. Ultimately at the end of the convolution process, if BUF1 contains the output, the new data is shifted in and the convolved data is shifted out at the same rate. It is assumed that the multiplication of filter coefficients and NTT coefficients is done within the computational unit.

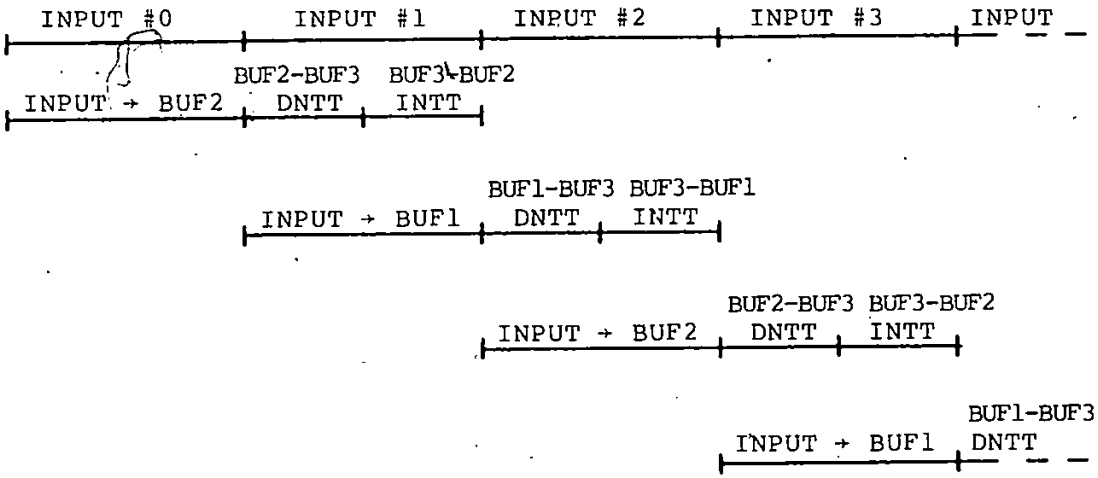


Fig. (3.2) Timing Diagram of an NTT Processor Organization for Real Time OIOO

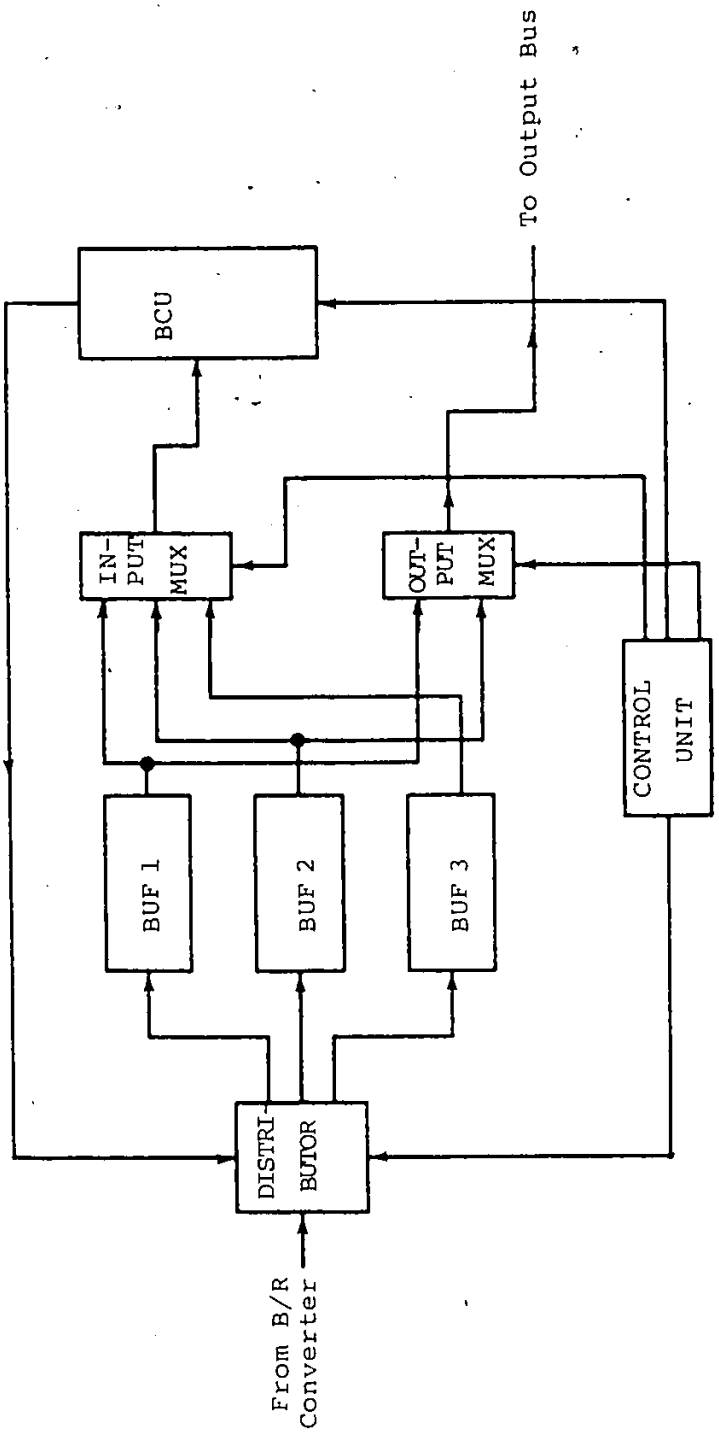


Fig. (3.3) An NTT Processor Organization for Real Time OIOO

3.2.2 Real-time NTT processor

As discussed in [14], for realization at a real-time NTT processor, the optimum memory organization consists of a three-buffer structure; BUF1, BUF2 and BUF3. For the minimum hardware requirement, the rate at which input data is moved in a memory must be twice that of the computational rate of NTT. Fig.(3.2) shows the typical timing diagram.

If BUF2 contains the convolved output of the previous cycle, BUF1 and BUF3 are involved in the computation of the data of the current cycle. New input to the next cycle is shifted in BUF2 and the output of the previous cycle is shifted out at the same rate.

This scheme requires a distributor unit, a BCU multiplexing unit and an output multiplexing unit. The distributor unit routes the input data and the data from BCU to the appropriate buffer. BCU multiplexing unit is required for data flow from the shift registers of the selected buffer to the BCU. The output multiplexing unit selects the buffer containing filtered output and routes it to output bus. A control unit is necessary to regulate the role of each of the above units during the operation of an NTT processor. The Fig.(3.3) shows the block diagram of an NTT processor for real-time applications.

3.2.3 Computational Unit

As discussed in section (2.6.2), the computational unit which is also known as a Butterfly Unit has a structure similar to the DIF butterfly unit. The selection of a

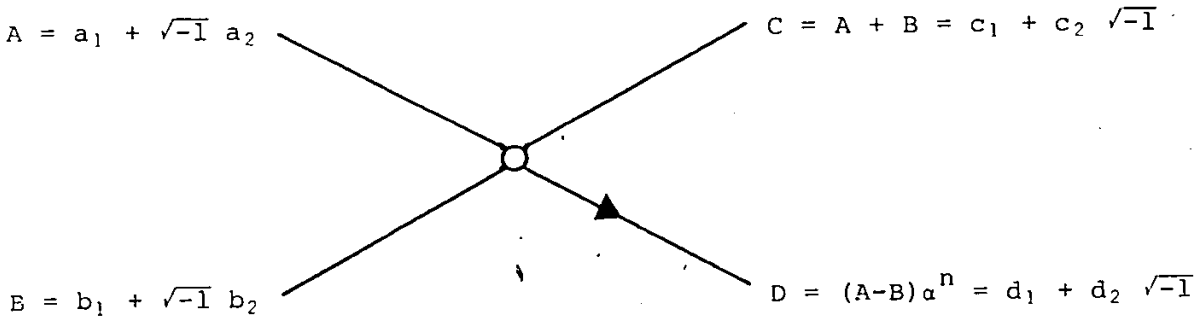


Fig. (3.4a) Radix-2 Butterfly Operations for $m_i = 4n + 3$

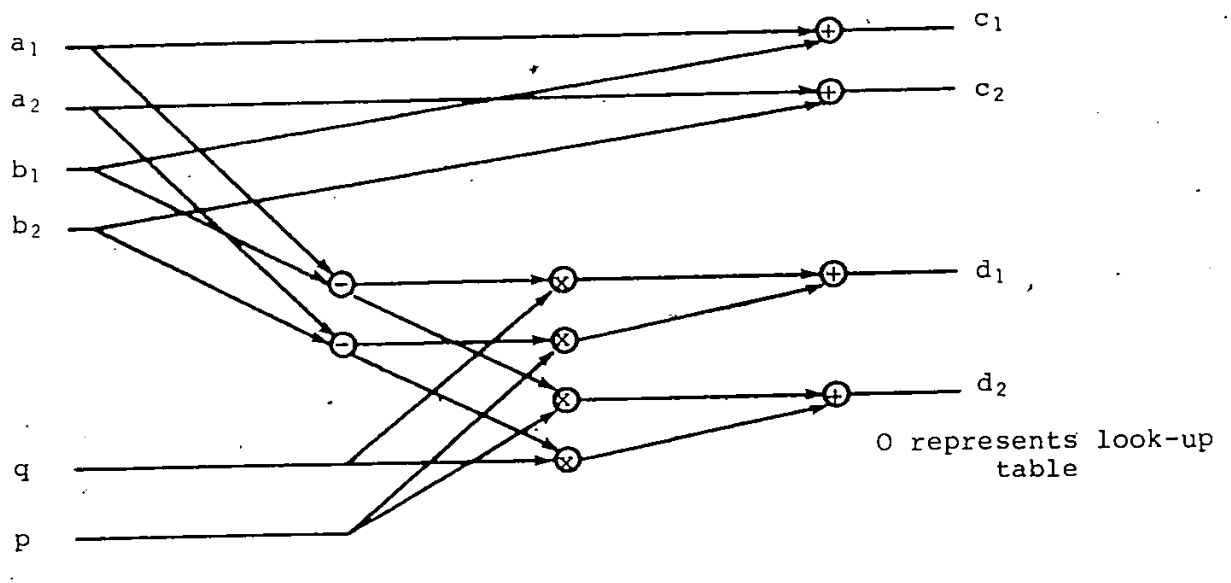


Fig. (3.4b) Look-up Table Implementation of a Radix-2 Butterfly for 4n + 3 Type Primes

modulus for an NTT determines the form of the cyclic group generator and hence the butterfly structure. In order to achieve large transform length, moduli should be the prime numbers. They can be divided into two groups; 4n + 1 type and 4n + 3 type.

3.2.3a Butterfly unit for 4n + 3 type primes

As discussed in [3], for 4n + 3 type prime moduli, the cyclic group generated α is of the form, $\gamma + \beta \sqrt{-1}$, where α and $\beta \in GF(m_i)$ and $x^2 + 1$ is an irreducible polynomial in the first order field.

Fig.(3.4a) shows the butterfly for 4n + 3 type of primes and Fig.(3.4b) shows its look-up table realization. The operations represented by $\odot$ are performed in the look-up tables. The input points are the elements of $GF(m_i^2)$ and can be considered as complex points. The butterfly calculation involves addition and subtraction of input points and multiplication of difference by a cyclic group generator. All binary operations performed are complex. The multiplication by α requires four multiplications, an addition and a subtraction, hence the butterfly for 4n + 3 type prime call for 10 binary operations.

3.2.3b Butterfly unit for 4n + 1 type primes

As discussed in [3], for 4n + 1 type primes, α has the simple form of $\sqrt{r}$, where $r \in GF(m_i)$ and $x^2 - r$ is an irreducible polynomial in $GF(m_i)$. Even the power of α can be considered as a real number and thereby only real multiplication is required. Fig.(3.5a) shows the look-up table

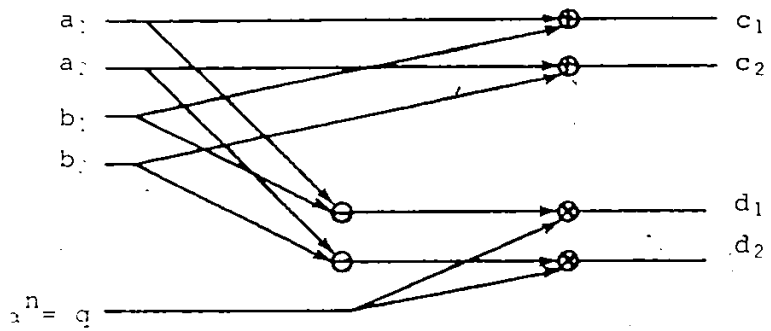


Fig. (3.5a) BCU for 4n + 1 type primes with n is even

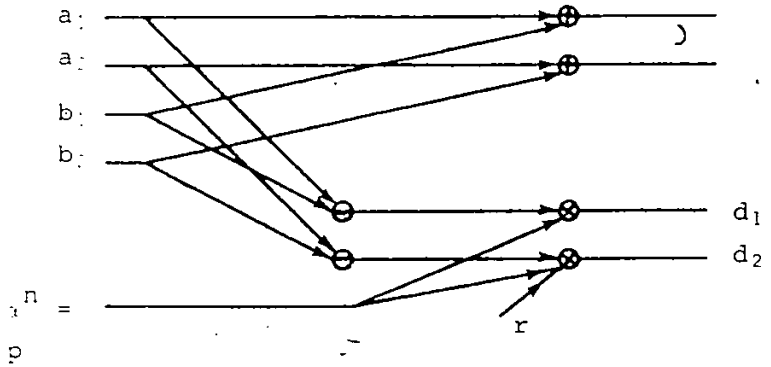


Fig. (3.5b) BCU for 4n + 1 type primes with n is odd

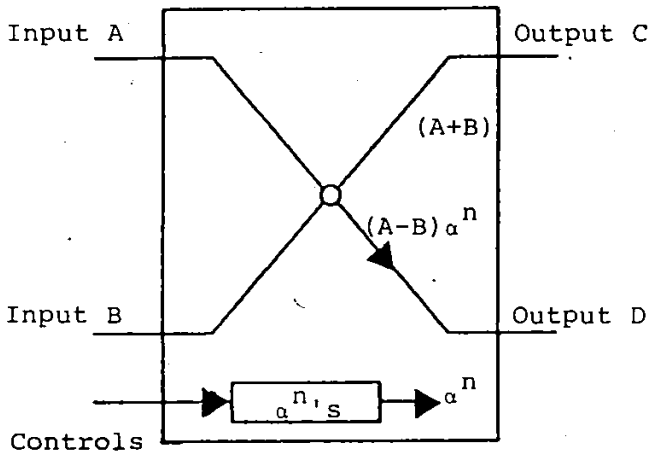


Fig. (3.6) Conceptual Diagram of the BCU

CONTROLS:

FWD/INV: DNTT or INTT

STAG: STAG OF COMPUTATION

POSTN: POSITION OF THE BUTTERFLY

implementation of 4n + 1 type of butterfly for the even power of α. The odd power of α requires an additional multiplication by r as shown in Fig.(3.5b). Hence the butterfly requires 6 binary operations.

It is obvious that the 4n + 1 type prime requires less hardware for BCU implementation than the 4n + 3 type as less binary operations are involved. While implementing a BCU, the primes of 4n + 1 type should be given preference.

3.2.4 Selection of the Prime Moduli for Hardware Implementation

As discussed in the previous chapter, in order to obtain a large dynamic range, the NTT is computed over a ring which is a direct sum of several second order Galois Fields, $GF(m_i^2)$. A transform length of 128 points is quite reasonable for practical applications. The Read Only Memory (ROM) of size 1024 x 8, i.e., 1k x 8 was selected as a building block for look-up tables in an NTT processor. To implement the multiplication efficiently, the sub-moduli technique was employed. The necessary and sufficient condition to select the sub-moduli, SUBM1 and SUBM2, is

$$SUBM1 \cdot SUBM2 \geq 2 \cdot m_i \qquad (3.1)$$

The ROM of size 1k x 8 has ten address lines, to obtain a large modulus m_i; sub-moduli should be represented by 5 bits each. We select 30 and 31 as the sub-moduli SUBM1 and SUBM2 respectively. This imposes the following condition on main modulus m_i:

$$m_i \leq \frac{SUBM1 \cdot SUBM2}{2}$$

i.e. $$m_i < 465 \qquad (3.2)$$

Thus our search for prime moduli has two constraints, it should support the transform length of 128 points and should be less than 465.

4n + 1 type primes can be represented as

$$m = q \cdot 2^p + 1 \qquad (3.3)$$

where q is odd and the maximum transform length is given by $N = 2^{p+1}$. For transform length of 128 points, p is equal to 6.

We consider the following numbers for the selection of primes:

for q = 1, $m = 1 \cdot 2^6 + 1 = 65$, which is not a prime

for q = 3, $m = 3 \cdot 2^6 + 1 = 193$, which is a prime

for q = 5, $m = 5 \cdot 2^6 + 1 = 321$, which is not a prime

for q = 7, $m = 7 \cdot 2^6 + 1 = 449$, which is a prime

for q = 9, $m = 9 \cdot 2^6 + 1 = 577$, which is a prime > 465.

Hence we can have 193, 449 as two prime moduli. The dynamic range associated with these two moduli is

$$\prod_{i=1}^{2} m_i = 193 \times 449 = 86657 = 2^{16.4} \qquad (3.3a)$$

For a reasonable dynamic range, we use one additional modulus of 4n + 3 type.

The moduli of 4n + 3 type can be represented as

$$m_i = r \cdot 2^{\ell} - 1 \qquad (3.4)$$

where r is odd and the maximum transform length is in

$GF(m_i^2)$ is equal to $2^{\ell+1}$. For our transform length $\ell = 6$. Our selection is restricted to following numbers.

For r = 1, $m = 1 \cdot 2^6 - 1 = 63$, which is not a prime

For r = 3, $m = 3 \cdot 2^6 - 1 = 191$, can be selected

For r = 5, $m = 5 \cdot 2^6 - 1 = 319$, which is not a prime

For r = 7, $m = 7 \cdot 2^6 - 1 = 447$, which is not a prime.

We select 191 as prime modulus of 4n + 3 type. The final selection of moduli is $m_1 = 191$, $m_2 = 193$ and $m_3 = 449$, hence the dynamic range is 23.98 bits. This is equivalent to saying that the NTT is computed over a finite ring, which is isomorphic to the direct sum of three Galois Fields of second degree that is

$$R = GF(191^2) \oplus GF(193^2) \oplus GF(449^2)$$

The cyclic group generators and primitive roots of moduli selected are as follows

$m_1 = 191$, $\alpha_1 = 66 + 6\sqrt{-1}$, $g = 19$

$m_2 = 193$, $\alpha_2 = \sqrt{125}$, $g = 5$

$m_3 = 449$, $\alpha_3 = \sqrt{391}$, $g = 3$

The parameter g is used to generate index and inverse index look-up tables.

3.2.5 Implementation of Butterfly Unit

A conceptual block diagram of the butterfly unit is shown in Fig.(3.6). It can be implemented using an all ROM structure which is inherently simple to pipeline. The two-point input is supplied to BCU along with some controls. These controls should provide information to BCU regarding the stage of NTT, position of butterfly to be computed and

whether DNTT or INTT, in order to choose the proper cyclic group generator. The typical throughput rate of BCU will be equal to $\left(\frac{1}{\tau_{BACC}}\right)$ MHz, where τ_{BACC} is the access time of ROM used. If we use the PROMs with latches with an access time of 40 nsec., i.e., PROMs; TBP28R165 by Texas Instruments [16], the throughput rate of 25 MHz can be achieved. The ROM oriented implementation of the BCU is discussed in [15].

3.3 COMPLEX MULTIPLIER

The sequence of operation of an NTT processor is as follows

(DNTT) (MULTIPLICATION BY FILTER COEFFICIENTS) (INTT)

i.e., after taking DNTT of an input sequence, the NTT coefficients are multiplied by filter coefficients and then the inverse NTT is taken.

Corinthois in [10] has suggested that, as in the final stage of DNTT, there is no multiplication and hence the complex multiplier in BCU can be used with an additional complex multiplier for multiplying two sequences in transform domain. Therefore, for this implementation, the butterfly structure consists of two complex multipliers. The typical operation in the final stage of NTT is as shown in Fig.(3.7).

3.3.1 Two Multiplier BCU Structure

Akhtar in [15] has discussed an all ROM implementation of BCU for minimum hardware requirement. He suggested the use of a real multiplier for 4n + 1 type of moduli with

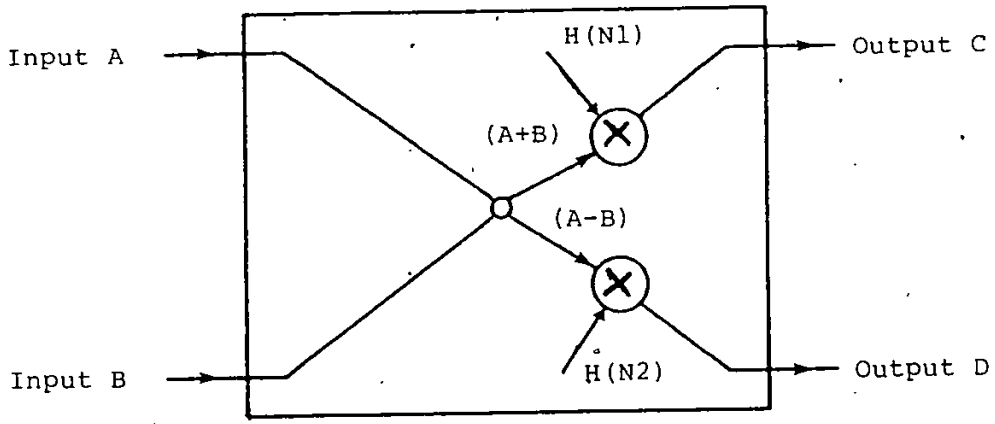


Fig. (3.7) Typical operations in a BCU at the final stage

Ⓧ represent complex multiplier

H(N1) Filter coefficients for $0 \leq N_1 < N/2$

H(N2) Filter coefficients for $N/2 \leq N_2 \leq N-1$

typical ROM count of 32 for modulus 193 and a complex multiplier for 4n + 3 type modulus, 191, with ROM count of 48. If we want to perform multiplication of two sequences in BCU then the implementation of this modified BCU will require 86 PROMs. It also requires some additional control circuitry to perform multiplication in the final stage of DNTT. With the modified BCU, the throughput rate of an NTT processor is $\left(\frac{1}{7 \cdot \tau_{BACC}} \right)$ MHz. This structure calls for a greater number of PROMs with lower access time, thus increasing the hardware cost.

3.3.2 External Multiplier

As the multiplication of two sequences is carried out after every 14 stages, we can use an external multiplier in such a way that the multiplication time is twice the NTT time.

This complex multiplier can be further implemented using an all PROM structure which is easy to pipeline. The PROMs with slower access time can be used, and the access time τ_{MACC} can be calculated as follows

$$\text{Multiplication Time} = 2 \cdot (\text{NTT Time})$$

$$128 \times \tau_{MACC} = 2 \times 64 \times 7 \times \tau_{BACC}$$

$$\tau_{MACC} = 7 \cdot \tau_{BACC} \qquad (3.5)$$

Hence the PROMs used for a multiplier can be 7 times slower than those used in BCU. This complex multiplier can be implemented using 30 PROMs.

The hardware requirement of this external complex

multiplier can be further reduced by using a serial complex multiplier. This multiplier can be implemented using a single real multiplier which can operate in pipeline to perform a complex multiplication. This structure requires thirteen PROMs ten of which have an access time of 70 nsec and three of which have an access time of 140 nsec. The block diagram of this serial complex multiplier is shown in Fig.(3.8)..

3.4 A SEQUENTIAL NTT PROCESSOR WITH FOUR BUFFER MEMORY ORGANIZATION

In order to achieve the throughput rate of $\left(\frac{1}{7 \cdot \tau_{BACC}}\right)$ MHz using an external serial complex multiplier approach, an additional buffer in memory organization and some extra multiplexing circuitry is required. Fig. (3.9) shows the block diagram of a real time sequential NTT processor with an external multiplier, modified memory structure and an additional multiplexing unit. Fig.(3.10) shows the timing diagram of the processor. The rate at which an input data is moved in the memory and at which one complex multiplication done is twice that of NTT.

This structure makes 100% utilization of BCU. In this memory organization, out of four buffers, two store the input and output of the intermediate NTT stages, the third stores the input and output of transform domain multiplication and the fourth stores the sampled input and convolved output.

If BUF3 and BUF2 are performing the INTT operation on the input data of the previous cycle, BUF1 will be involved

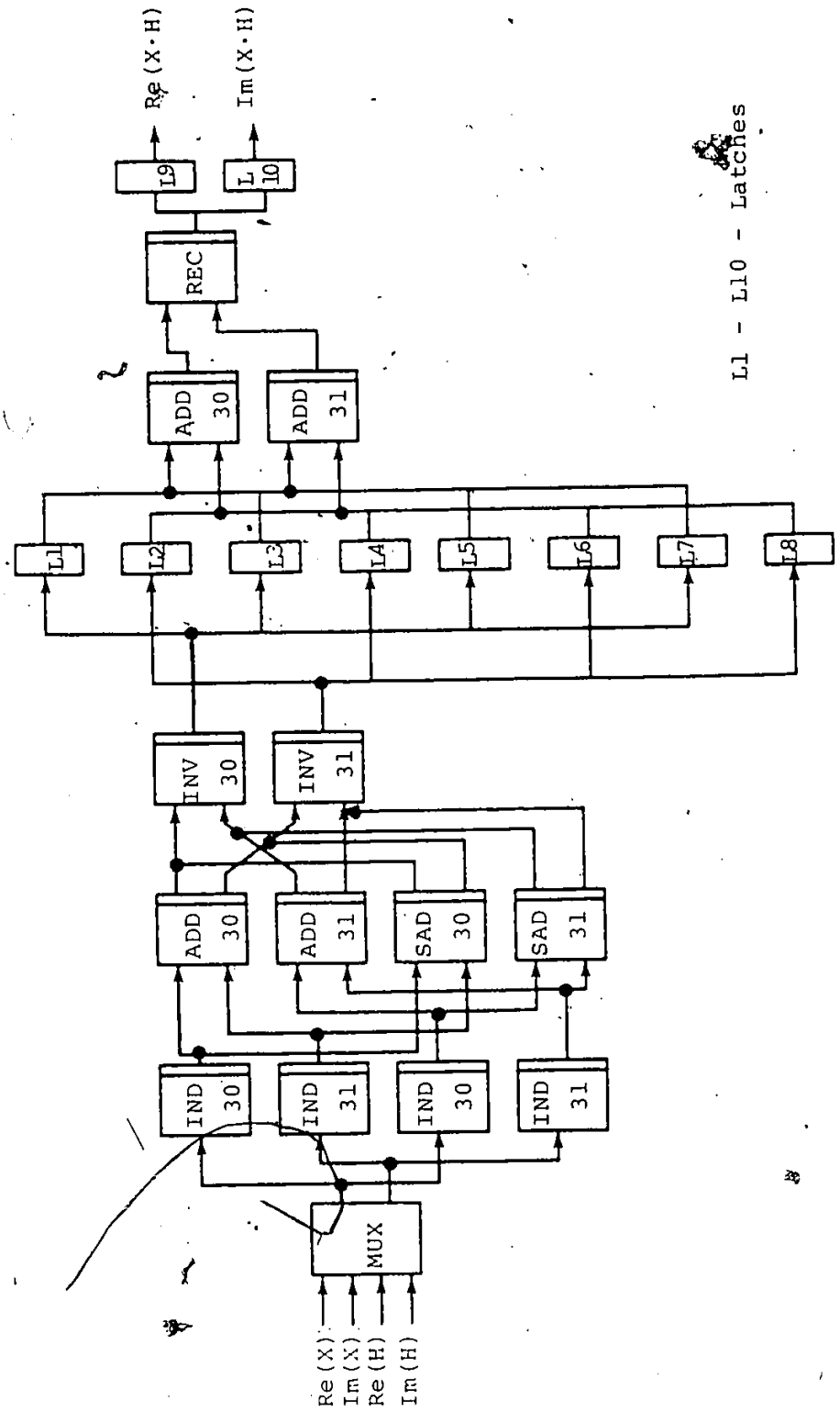


Fig. (3.8) The Serial Complex Multiplier

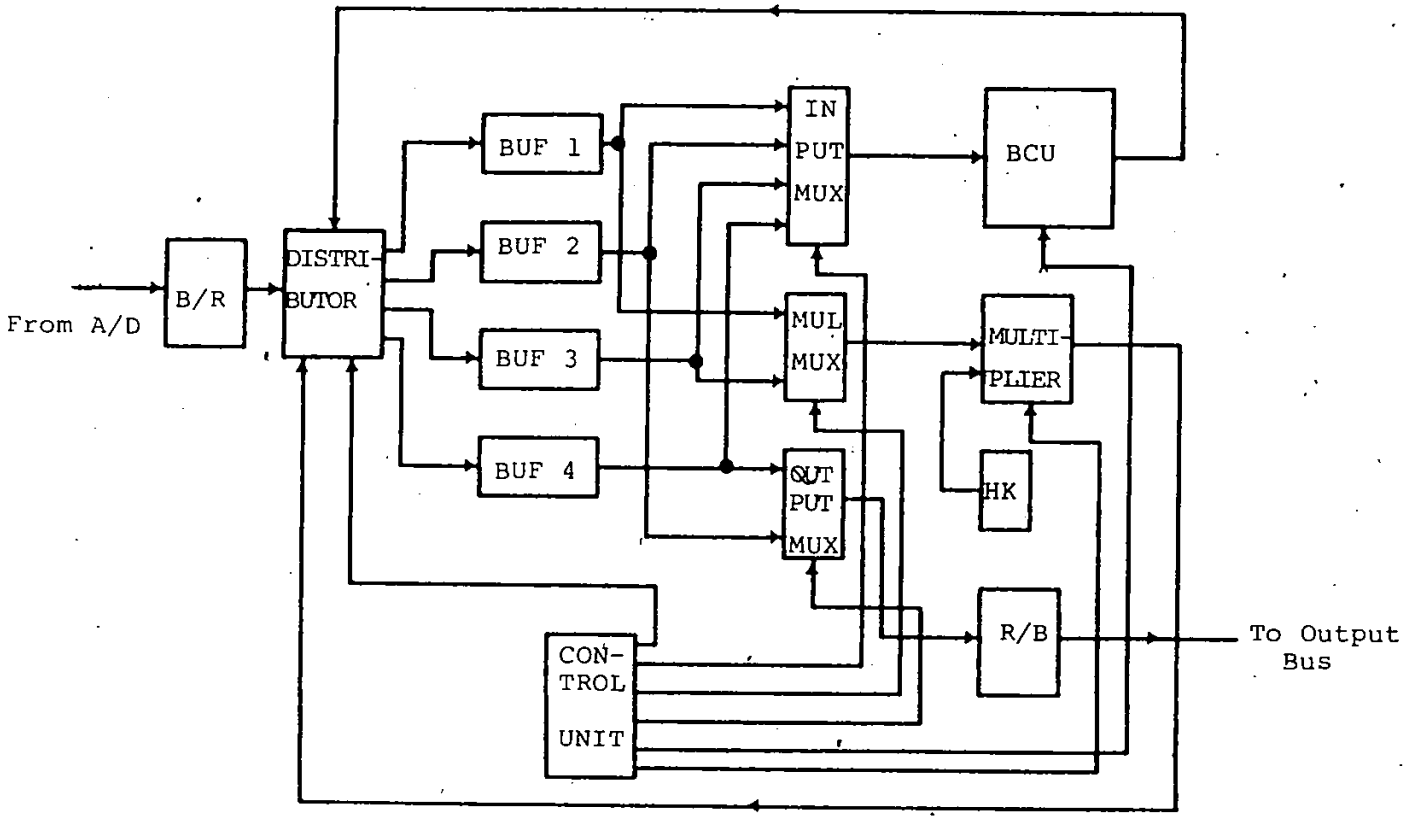


Fig. (3.9) A real time NTT processor organization with 4 Buffer memories

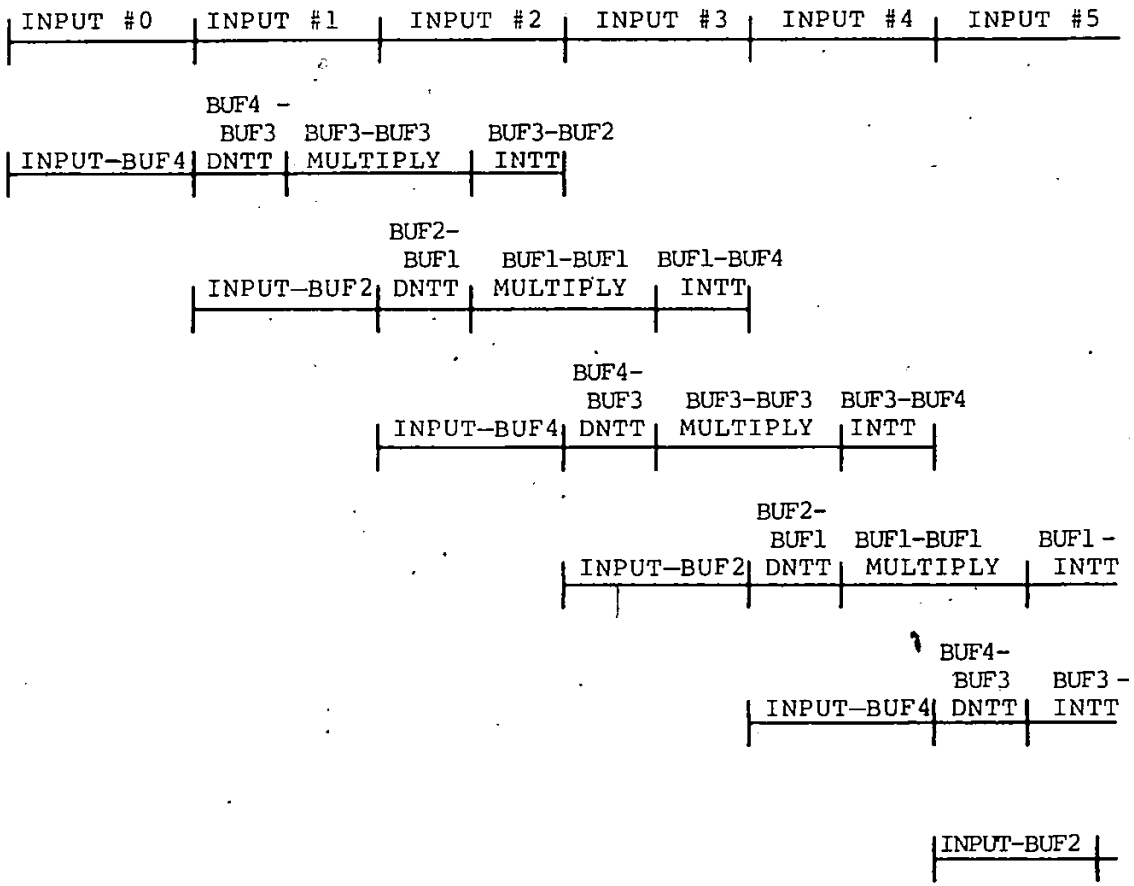


Fig. (3.10) Timing Diagram of an NTT Processor Organization with 4 Buffer Memories

in transform domain multiplication and BUF4 will be shifting out a filtered output and at the same time it will be receiving the new data for the next stage.

Compared to three-buffer memory organization, this scheme calls for an additional buffer and some extra control circuitry, but it is cost-efficient as it does not require as many fast memories in BCU. If we use the PROM with an access time of 40 nsec in BCU, the processor can have the throughput rate of 3.5 MHz.

As all the arithmetic operations in the processor are done in RNS, we need a Binary to Residue Converter (BRC), at the input side of processor and a Residue to Binary Converter (RBC), at the output side of processor as shown in Fig.(3.9).

3.4.1 Distributor unit (DU)

The DU receives the input from a binary to residue converter, BCU and complex multiplier. Its function is to route in-coming sequences to the appropriate buffer.

If BUF2 is shifting the input to BCU, the DU has to route output coming from BCU to BUF1, output from multiplier to BUF3 and input in residue form to BUF4.

The distributor unit can be realized as the bank of multiplexing circuitry associated with each buffer. Any of the four buffers can receive the input from BCU and either from BRC or from multiplier. The multiplier output will always be going to either BUF1 or BUF3 and BRC output will be stored in BUF2 and BUF4. Hence, multiplexing circuitry associated with each buffer has to multiplex two of the in-

coming inputs at a time. Fig.(3.11) shows the block diagram of DU consisting of the four-multiplexing unit associated with each buffer. Each unit receives the select signals from a control unit. These multiplexing units are made up of standard two-line to one-line multiplexers.

3.4.2 BCU Input Multiplexer Unit

Any of the four-buffers can give an input to BCU. Each buffer is divided into four shift registers. Two out of four shift registers of a selected buffer can give input to BCU at a time. Fig.(2.10) in section 2.10.2 shows the buffer connections in the first stage of NTT and Fig.(2.11) shows connection in subsequent stages. Hence this BCU input multiplexing unit has to select two out of sixteen shift registers at a time, which are containing an input to BCU. This multiplexing unit can be implemented using standard multiplexer or octal buffers.

3.4.3 Output Multiplexer Unit

The buffer BUF2 or BUF4 will be always storing the convolved output of the processor. The shift registers of the buffer containing output are connected serially and the output is taken from the top most shift register. The output multiplexer has to select the output either from BUF2 or BUF4 depending upon the cycle. It can be implemented using two-line to one-line multiplexers. Fig.(3.12) shows the block diagram of output multiplexing unit.

3.4.4 Multiplier Input Multiplexer Unit

Either BUF1 or BUF3 will be giving input to the multi-

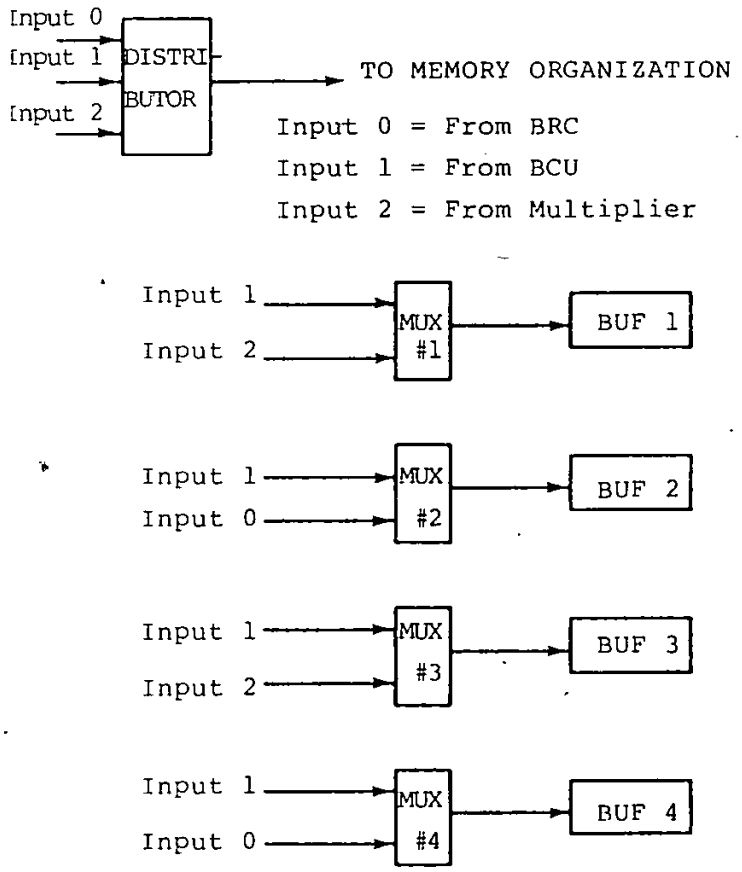


Fig. (3.11) Block Diagram of the Distributor Unit

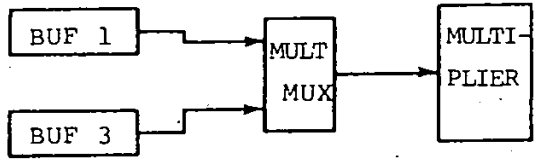


Fig. (3.12) Block Diagram of an Input Multiplexer unit for the Serial Complex Multiplier

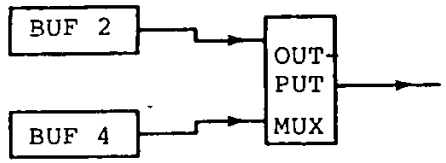


Fig. (3.13) Block Diagram of Output Multiplexer Unit

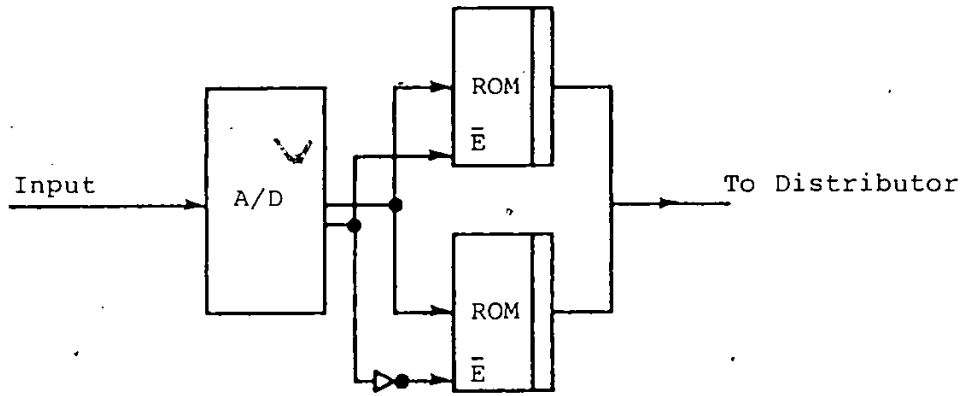


Fig. (3.14) Block Diagram of a Binary to Residue Converter for $m_i < 256$

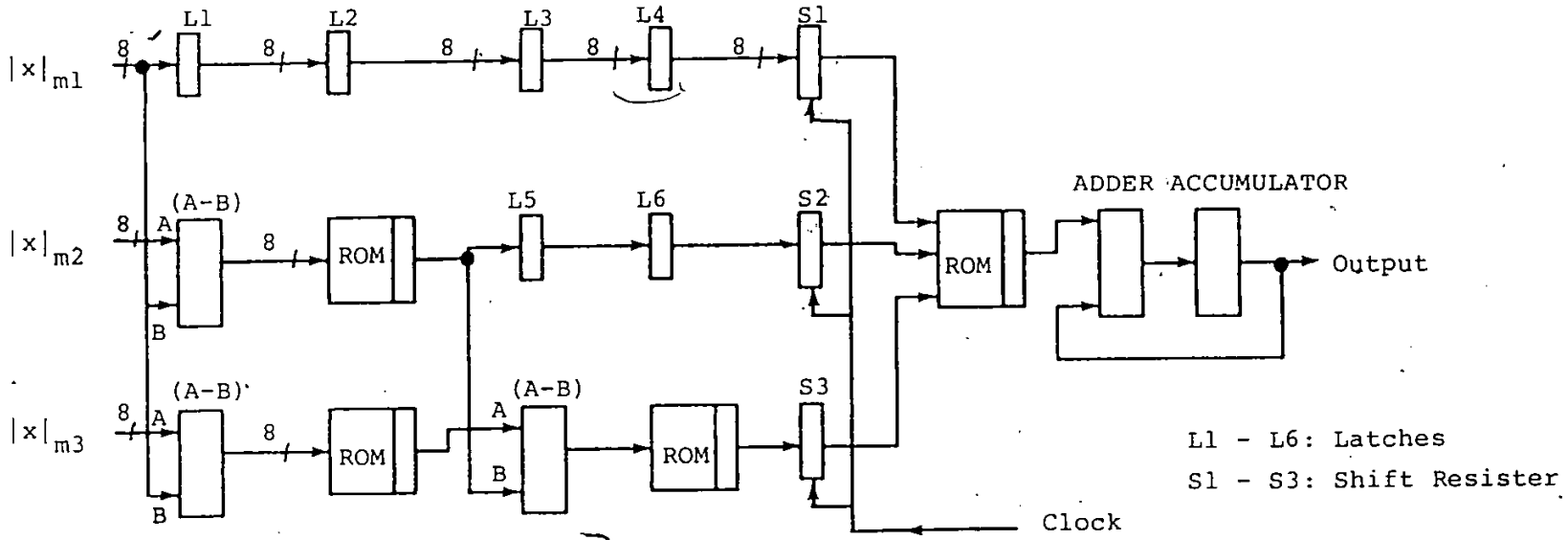


Fig. (3.15) Block Diagram of a Residue to Binary Converter for 3 Moduli

plier, where it is multiplied by pre-stored filter response coefficients so this multiplexing unit has to select input either from BUF1 or BUF3. Fig.(3.13) shows the block diagram of the multiplier input multiplexer unit.

3.4.5 The Memory Buffers

The memory organization consists of four buffers, each of the size 128 x 16. Each buffer is divided into two sub-buffers SM1 and SM2, which are further divided into two shift registers SR1 and SR2. Hence any buffer can be implemented using four shift registers, each of size 32 x 16.

The bipolar random access memories can be used to implement these buffers. Fairchild's 3348 or 3349 Hex 32 bit static shift registers or AMD's AM2812A, 32 x 8, First In, First Out (FIFO) memories can be used. These shift registers can operate up to 1 MHz. For higher throughput rate, TI's 74LS222/74LS227, 16 x 4 FIFO can be used [16] - [18].

Additional switching circuitry is required to interconnect the shift registers of each buffer. It is explained in detail in the next chapter.

3.4.6 Binary to Residue Converter (BRC)

The binary to residue converter is necessary to have interface between the conventional digital system and the digital system using the residue number theory.

As we have decided to use PROMs of the size 1k x 8 to store the filter response coefficients, and our dynamic range 23 bits, hence we can use a 12 bit A/D convertor at the input side. The BRC for $m_1 = 191$ and $m_2 = 193$ can be implemented

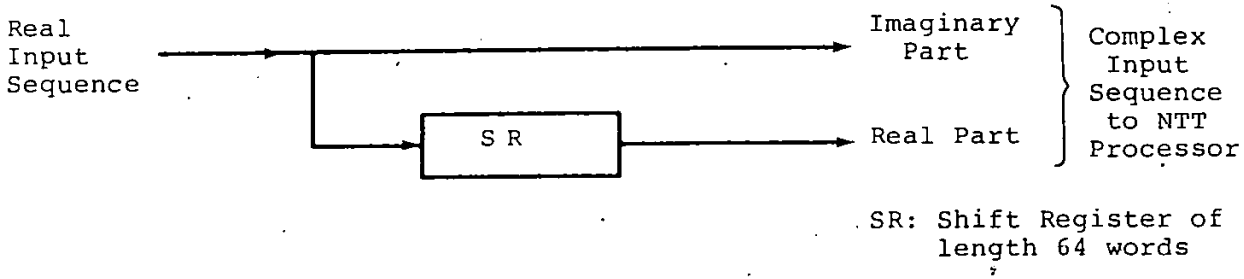


Fig. (3.16a) Generation of Complex Input Sequence to NTT Processor

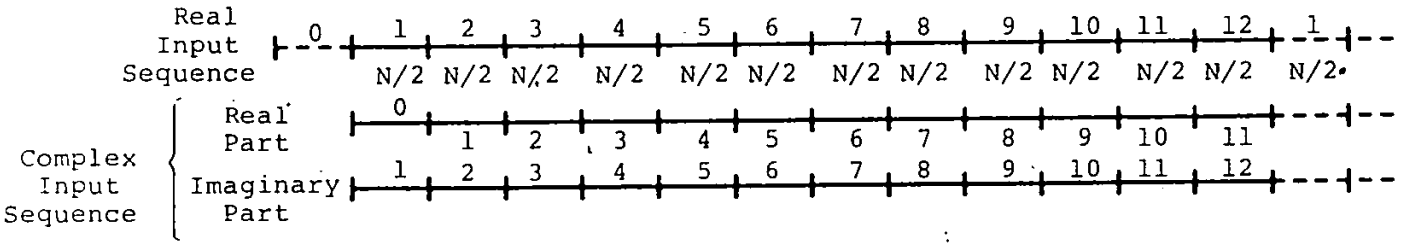


Fig. (3.16b) Timing Diagram

by using 4k x 8 PROMs each. For $m_3 = 449$ we require a storage capacity of 4k x 9 for BRC. Fig.(3.14) shows the block diagram of BRC for $m_i < 256$.

3.4.7 Residue to Binary Converter (RBC)

The filtered output from the processor is in a residue form. The conversion of residue digits into a weighted binary form calls for a residue to binary converter unit.

The equation (2.11) in section (2.3) can be implemented using look-up tables. But this scheme requires large amounts of memory. These equations can be efficiently implemented with subtractors which are followed by the look-up tables as shown in Fig.(3.15). The RBC for moduli $m_1 = 191$, $m_2 = 193$ and $m_3 = 449$ shown in Fig.(3.15), requires four PROMs of size 1k x 8, six latches, four 8 bit subtractors, three shift registers and an accumulator register.

3.5 PROCESSORS ORGANIZATION FOR REAL VALUED INPUT SEQUENCE

In many practical situations, we are interested in computing the convolution of two finite duration sequences, where one sequence is much longer than other sequence, ie, $L >> M$, where L and M are the lengths of f and h respectively. The larger sequence f is sectioned and partial results are computed which can be combined to form desired output sequence.

An overlap add and an overlap save are the two techniques used to perform convolution by the method of sectioned convolution. In overlap add method, results of two consecutive sections are added to generate the desired results. In Overlap save method an input sections are overlapped and some

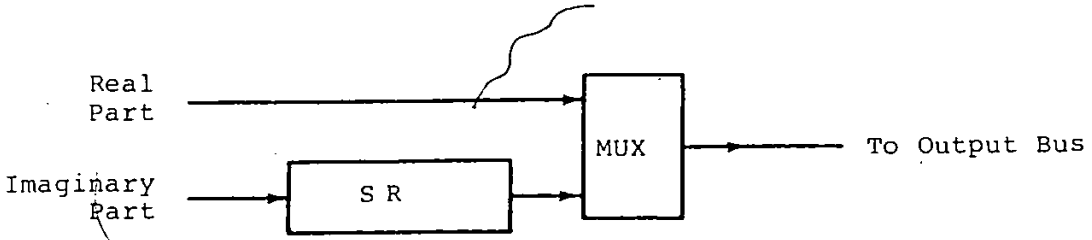


Fig. (3.17a) Generation of Real Output from the Complex NTT Processor Output

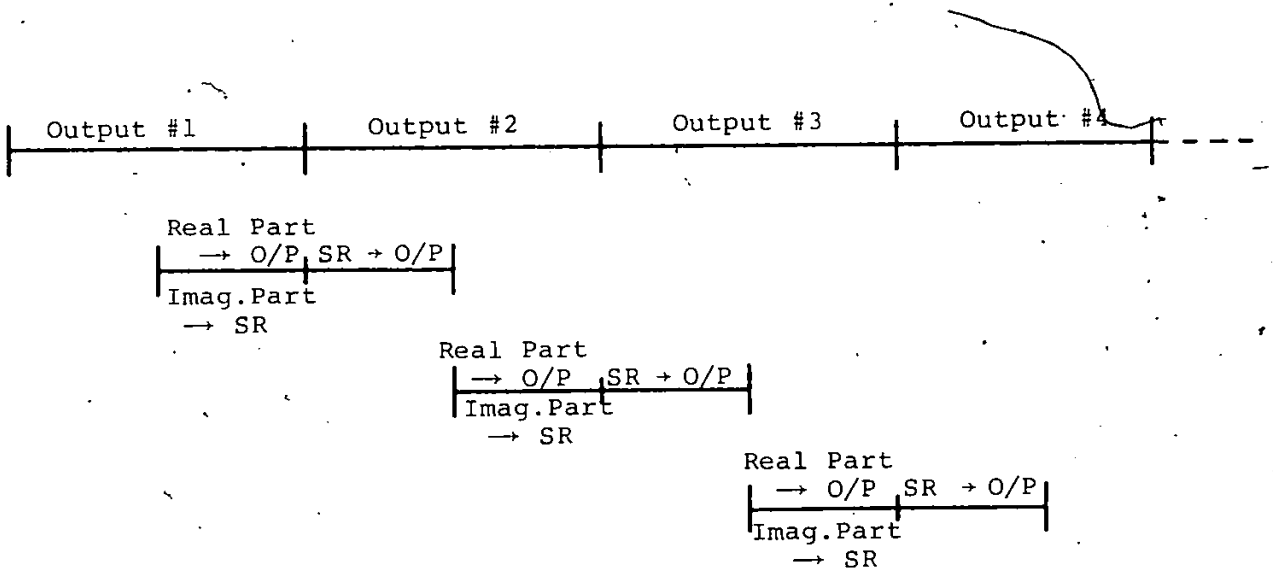


Fig. (3.17b) Timing Diagram

of the samples of the results are discarded. The overlap add technique requires more hardware than the overlap save technique, since it involves the addition of two sequences. Thus an overlap save technique was chosen.

As discussed in [14], to implement this technique we require an additional shift register the size 64 x 16 at the input and the output of the processor. Let the length of impulse response M, be (N/2 + 1). Fig.(3.16a) shows a scheme to generate the complex input to an NTT processor for processing a real-valued input sequence. The shift register SR is used to generate a delay of N/2 samples between the real and imaginary parts. At the start SR is cleared so as to fill it with 0's. The real part is taken from the output of SR whereas an imaginary part is taken directly; at the same time the input sequence is also shifted into SR. Thus the real input sequence is divided into sections of length N/2 and a combination of different sections form an input to the processor. For example, a combination of sections 0-1, and 1-2 forms one input, 2-3 and 3-4 forms another input as shown in Fig.(3.16b).

To generate the filtered output, a delay of N/2 samples between the real and the imaginary parts of the complex output is required. The first half of the output sequence is always discarded. During the second half of the complex output sequence, the real part is routed to the output bus and at the same time the imaginary part is moved into SR as shown in Fig.(3.17b). The imaginary part in SR is gated to

output bus during the first half of the next filtering operation. This process is shown in Fig.(3.17a) and Fig.(3.17b).

If the access time of the PROMs in BCU is 40 nsec, and the filter impulse response is (N/2 + 1) points, the real valued input sequence of 128 points will take

$$4 \times \tau_{BACC} \times 64 \times 7 \text{ nsec}$$

i.e. 71.78 μsec. The throughput rate of 3.5 MHz can be obtained.

3.6 SIMULATION OF AN NTT PROCESSOR

The proposed processor structure, consisting of three moduli, was simulated on IBM 370-3031. The subroutines were written to simulate different units involved in the processor organization. A simple program was written to simulate BCU. The pipeline delay of BCU and the multiplier was taken into consideration. The main program and the subroutines were written in WATFIV.

1) Main Program:-

The main program assigns values to various parameters used, and initializes the latches and memory buffers. By calling different subroutines, it generates necessary look-up tables. Mainly it simulates the overall modular structure by calling different sub-routines. It governs the various controls required in the distributor unit and the multiplexing units. The individual blocks are simulated within the sub-routines.

2) Subroutine TWLF:-

The TWLF generates the powers of α and α^{-1}. It stores

these powers in the form required by OIOO algorithm. First the powers of α are calculated and then the powers of α^{-1} are computed using the following equation:

$$\alpha^{N} = 1 \quad \text{and} \quad \alpha^{-I} = \alpha^{N-I}$$

The real and imaginary parts are stored separately.

3) Subroutine LTABLE:-

The subroutine LTABLE computes all the necessary look-up tables required for multiplication and NTT calculations. It calculates the index table - IND(I), the inverse index table - INV(I,J), the addition look-up tables - ADD(I,J) and a reconstruction table -RECT(I,J). It stores these tables at moduli 30 and 31. INV1(I,J) and INV2(I,J) represents the inverse look-up table at moduli 30 and 31 respectively. The modular reduction was done using the instruction

IX = MOD (IR, MMOD)

where MMOD is modulus and IR is the number to be reduced.

4) Subroutine STAG1:-

The first NTT stage calls for data separation of N/2 words. The buffer interconnection during this stage is different than that during subsequent NTT stages. The STAG1 simulates the entire operation during this stage. The subroutine call is,

CALL STAG1 (BUF11, BUF12, BUF13, BUF14, BUF21, BUF22, BUF23, BUF24, NTT, STAG, INV) where BUF1's and BUF2's represents the shift registers of corresponding memory buffers BUF1 and BUF2 respectively. STAG indicates the stage at NTT and

INV tells whether DNTT or INTT. This subroutine calls for the subroutine NTT, which performs the BCU calculations.

4) Subroutine ASTAG:-

All the stages other than the first one require a data separation of N/4 words. The shift register selection for accessing the data depends upon the NTT stage as described in the previous chapter. The subroutine ASTAG simulates the memory organization during these NTT stages. The subroutine call is,

CALL ASTAG (BUF21, BUF22, BUF23, BUF24, BUF11, BUF12, BUF13, BUF14, NTT, STAG, INV). The BUF2 contains the input to the NTT stage and BUF1 stores the output of the BCU.

5) Subroutine NTT:-

This subroutine implements butterfly calculation. It does not simulate the BCU structure. The subroutine call is, CALL NTT(STAG, POS, A11, A12, B11, B12, B01, B02, B03, B04, INV), where STAG, POS and INV provides the necessary information regarding the stage of NTT, whether DNTT or INTT and the position of a butterfly to be implemented. The A1's and B1's are complex inputs and B0's are complex outputs.

6) Subroutine MULTLY:-

The subroutine MULTLY simulates the structure of an external serial complex multiplier. The computer program was written so as to preserve the pipeline structure. The pipeline delay was also taken into consideration. This subroutine also simulates the memory buffer interconnections during the multiplication of two sequences. The subroutine

call is, CALL MULTLY (CUF11, BUF12, BUF13, BUF14, MMI, MMF, BUF31, BUF32, BUF33, BUF34).

Where the BUF1 contains the NTT coefficients to be multiplied with the filter response coefficients and it also stores the product of two. The MMI and MMF provides the information about the number of points to be multiplied as from 1 to 64 or 65 to 128. The multiplier has the delay of three samples. The first three samples coming out of the multiplier belong to the multiplication of the previous cycle. As BUF3 stores the multiplier output of the previous cycle, these three samples should be stored in BUF3. This subroutine is self-explanatory and the details can be found in Appendix A.

7) Subroutine OVLAP:-

The subroutine OVLAP converts the real valued input sequence to the complex sequence and arranges the input in the form required by the overlap save method. The subroutine call is

CALL OVLAP (IKT, IIN, ITS)

where IKT contains the real valued input sequence and IIN stores the complex input sequence in the processor.

8) Subroutine INOUT:-

During the input and output operations the shift registers of the selected buffer are serially connected. The output is shifted out and at the same time new input is shifted in. This operation is simulated by the subroutine INOUT.

9) Input Files:-

The subroutine INFILE and FILTER provide the input data to be filtered and the filter response coefficients respectively. It is assumed that input files are premultiplied by the appropriate scale factors.

Appendix A contains the listing of all the subroutines.

3.6.1 Simulation Results

The testing of the simulation of an NTT processor was done with the different sets of INPUT and FILTER files. In the first stage of testing the ramp was convolved with the impulse and in second stage the processor was tested for the lowpass filtering. The details are as follows:

1) The real valued input sequence was made up of points from 0 to 127. It was convolved with the impulse and the output was observed. The Overlap save technique was employed. Fig.(3.18) shows the input and output waveforms.

2) The frequency response of lowpass filter with cutoff frequency of 2.6k Hz was taken. The impulse of LPF was premultiplied by a scale factor of 10. The upperbound on the input of each moduli was calculated. The scale factor used for moduli 191 and 193, was 8.5 and for 449 was 22. In first case input was the sum of the sinewaves with frequencies 1 kHz and 10.0 kHz. In the second case the input was sinewave with frequency 10.0 kHz. Fig.(3.19) and (3.20) shows the input and output of simulation.

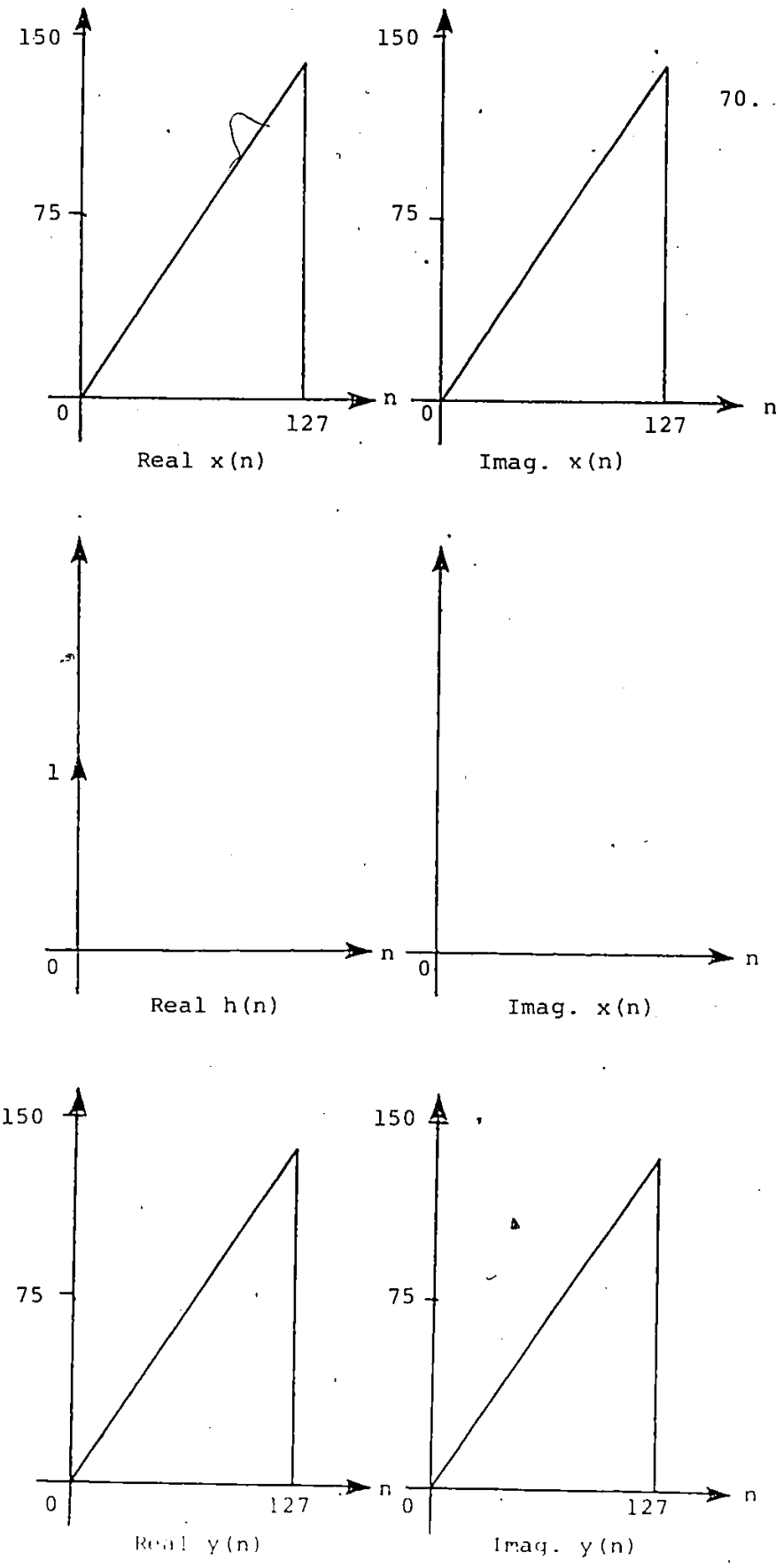


Fig.(3.18) Convolution of Ramp With Impulse Using NTT

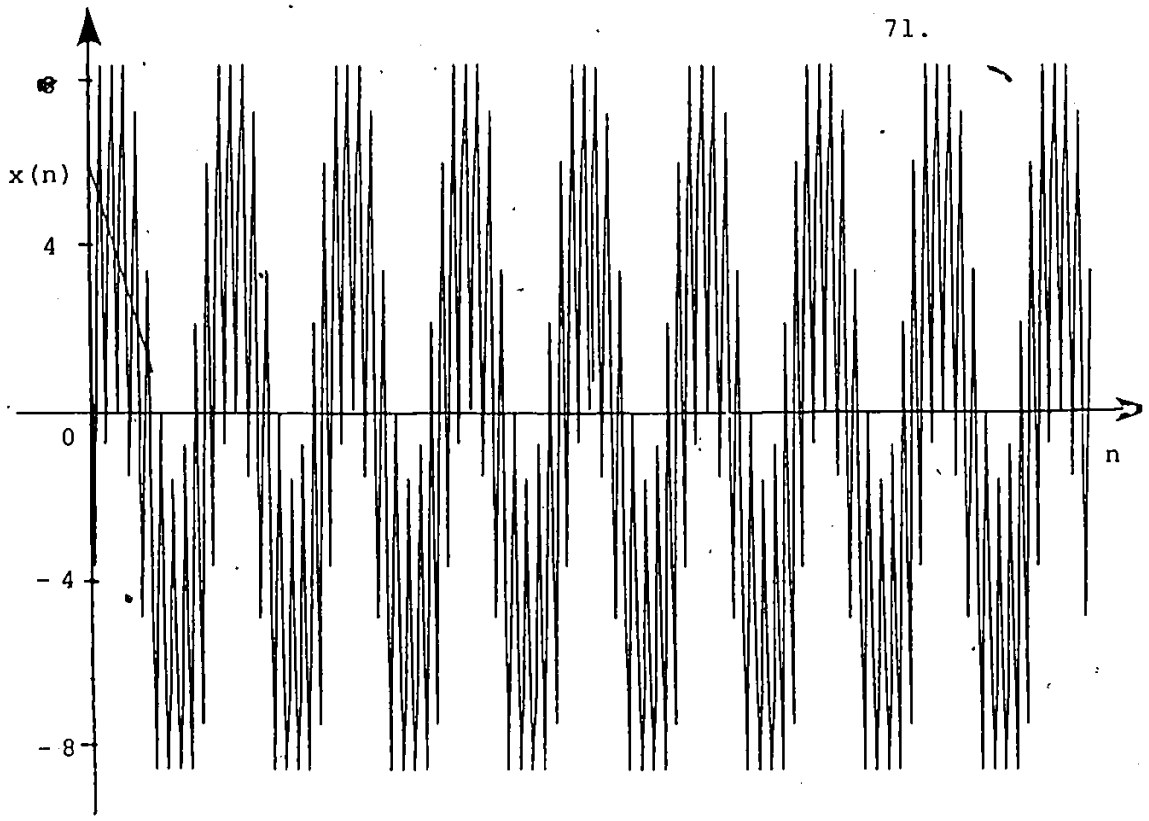


Fig. (3.19 a) Input to the Processor.
$x(n) = \sin W_1 t + \sin W_2 t$
f_1 = 1 kHz and f_2 = 10 kHz

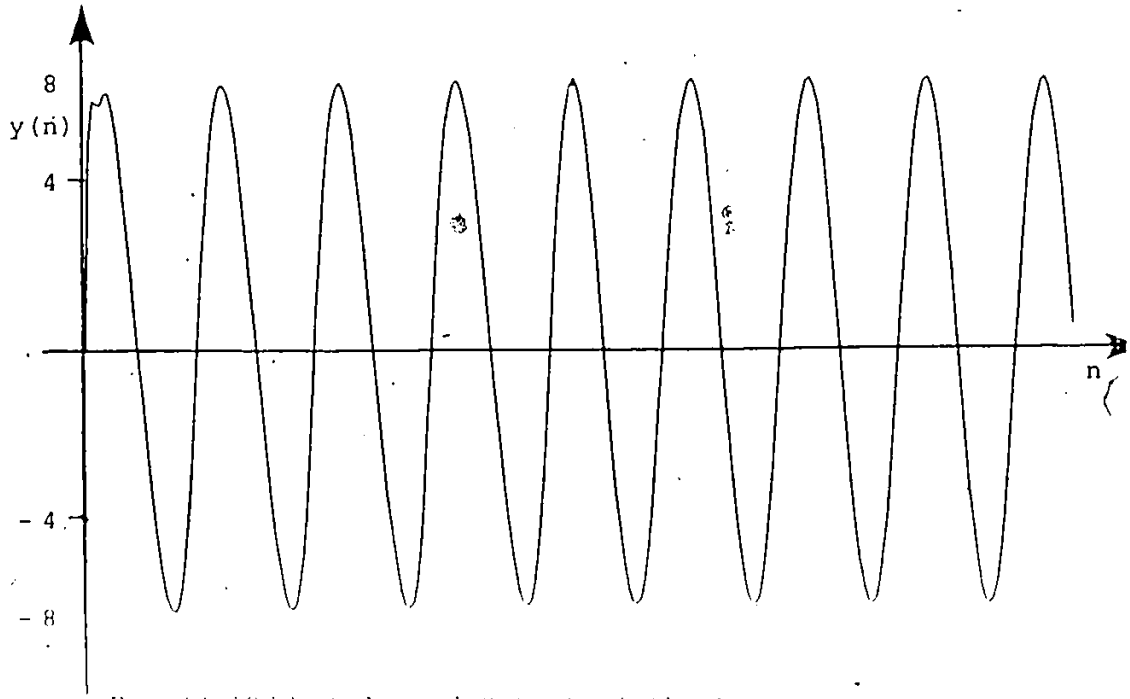


Fig. (3.19 b) Filtered Output of the Processor

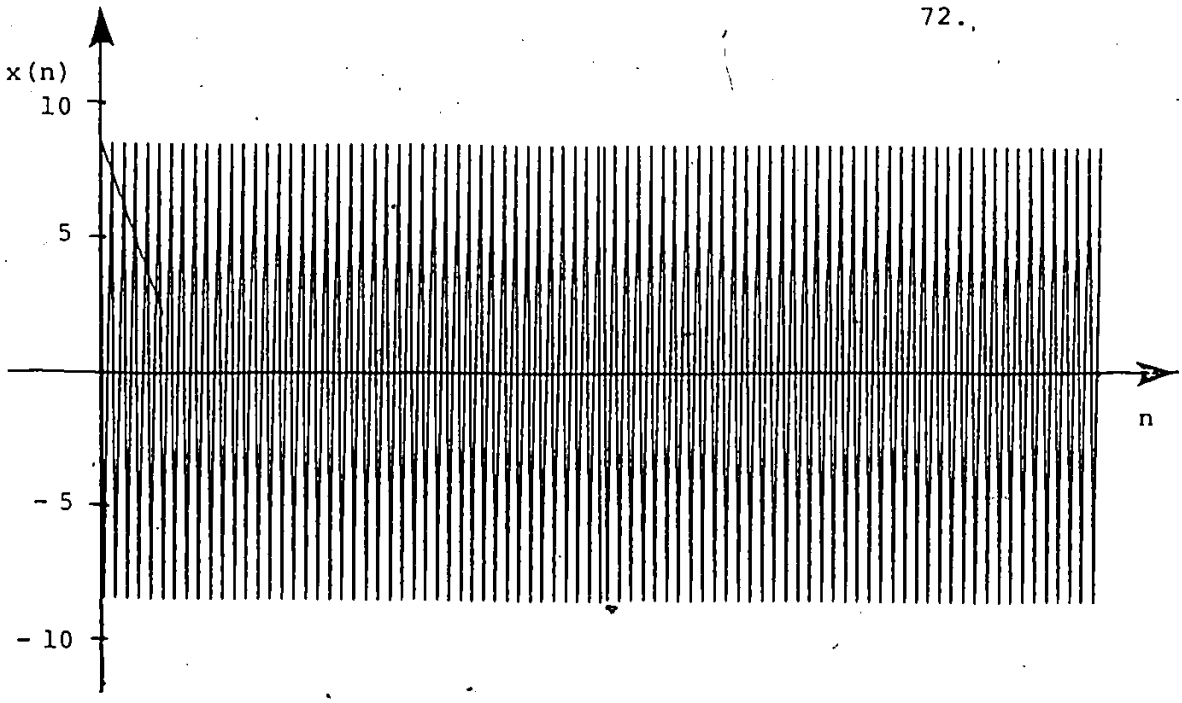


Fig. (3.20 a) Input to the Processor

$x(n) = \sin \omega_1 t$; $f_1 = 10$ kHz

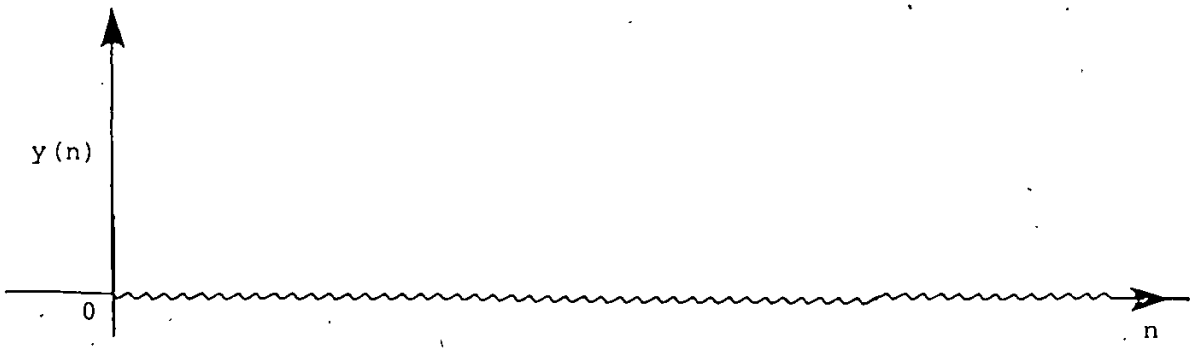


Fig. (3.20 b) Filtered Output of the Processor

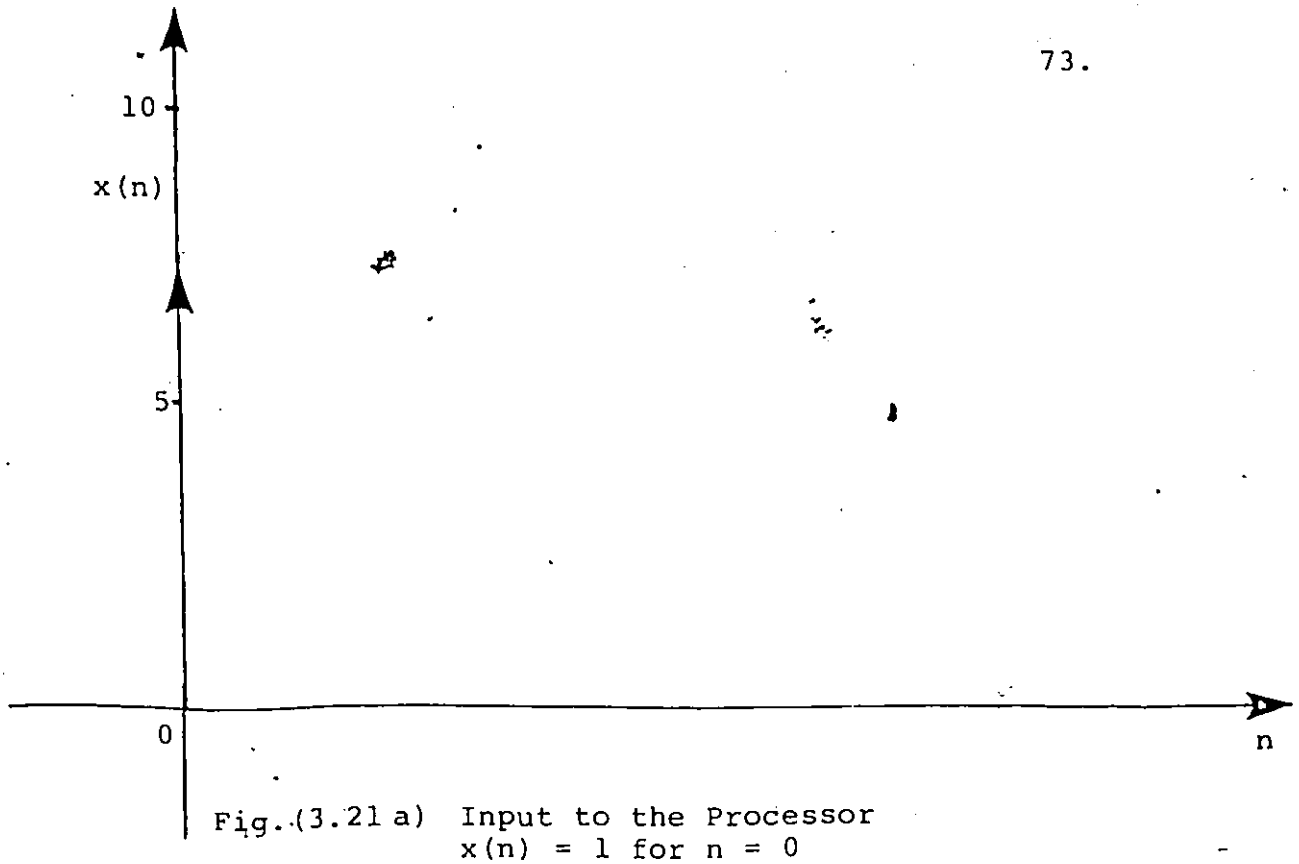


Fig.(3.21 a) Input to the Processor
x(n) = 1 for n = 0
= 0 otherwise

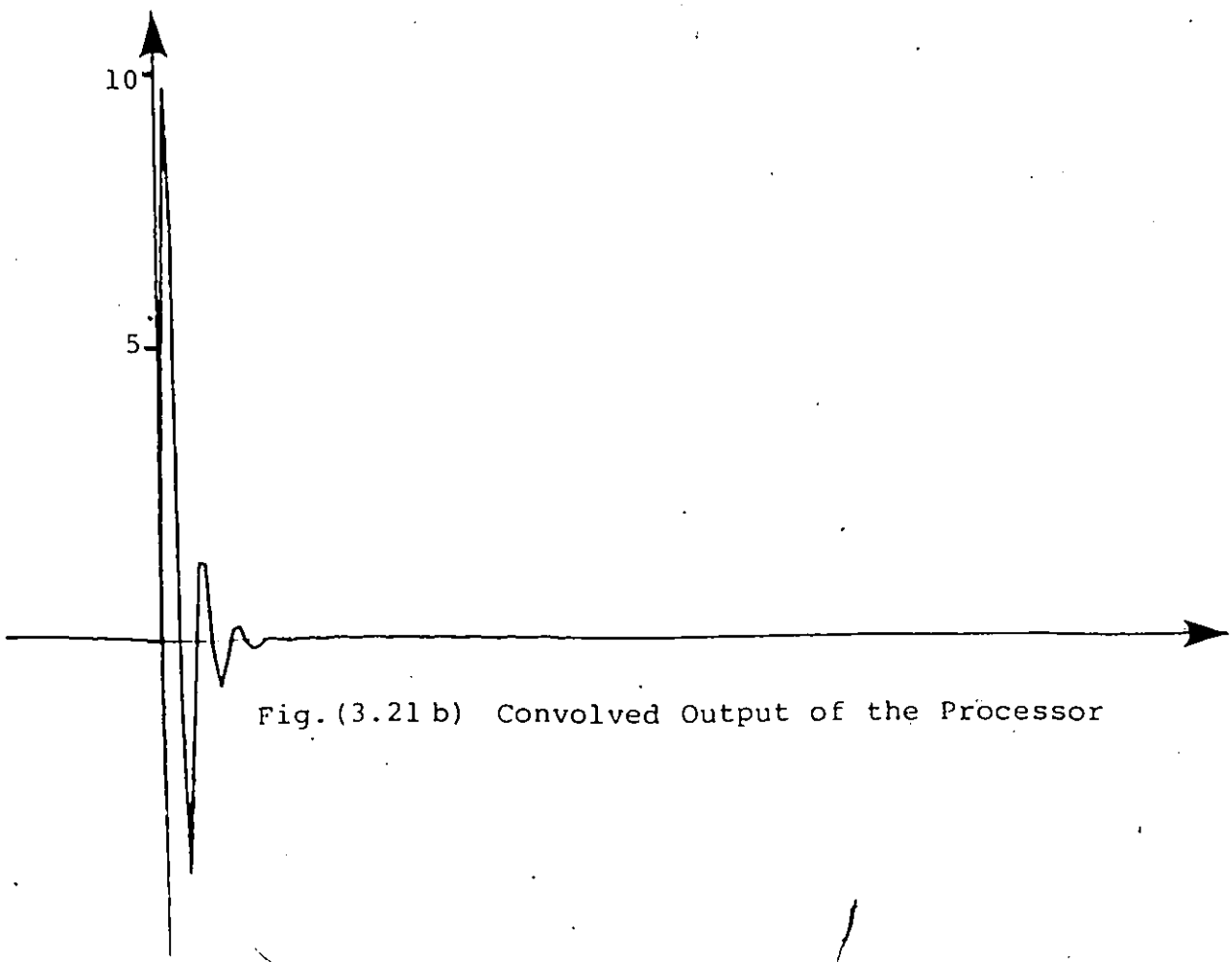


Fig.(3.21 b) Convolved Output of the Processor

3.7 SUMMARY

The modular design of an NTT processor was discussed. The method of selecting primes for efficient implementation was described. The processor with slower external serial complex multiplier and four buffer memory organization was proposed. The implementation of the processor's supporting structure (which includes R/B, B/R converters, Distributor and various multiplexing unit) was done. Finally the simulation of the proposed NTT processor for three moduli scheme was done.

CHAPTER 4

THE HARDWARE IMPLEMENTATION OF AN NTT PROCESSOR MODULE 193

4.1. INTRODUCTION

The sequential NTT processor essentially consists of three processor modules namely module 193, module 191 and module 449. The decision was made to build the module 193 because of the availability of a butterfly unit for this module. This module consists of a complex multiplier for modulus 193, a memory organization, a distributor unit, an input/output multiplexing unit and a control unit. The prototype was constructed using standard IC's and proto-boards. The proto-boards selected were Experimenter 300, 600 and quad bus strip 4 B. The UV erasable, programmable, read-only memory of the size 1k x 8, ie, EPROM 2708 with access time of 450 nsec was chosen. The EPROM requires three power supplies in read mode, V_{CC}, V_{BB} and V_{DD} which are +5, -5 and +12 volts respectively. It has ten address lines and eight data lines. The higher address lines are grounded if they are not in use.

The Octal D flip-flop with 3-state outputs, 74S374 was used as an edge triggered latch.

The 32 x 8 First-in - First-out (ie.FIFO) memory with maximum parallel load frequency of f MHz, Am2812A was selected as a building block for memory organization. Using these components, we arrive at the throughput rate of an NTT

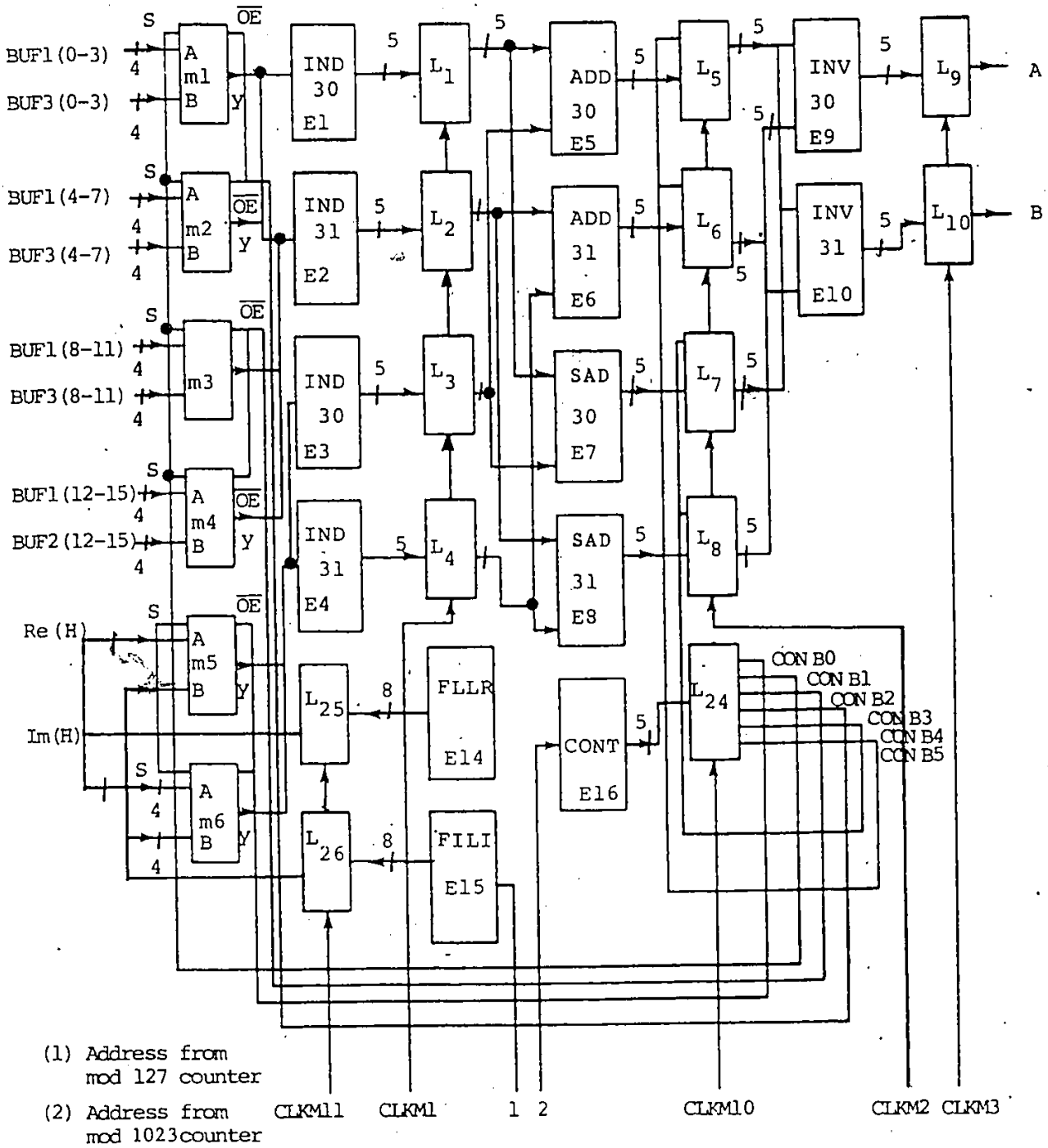
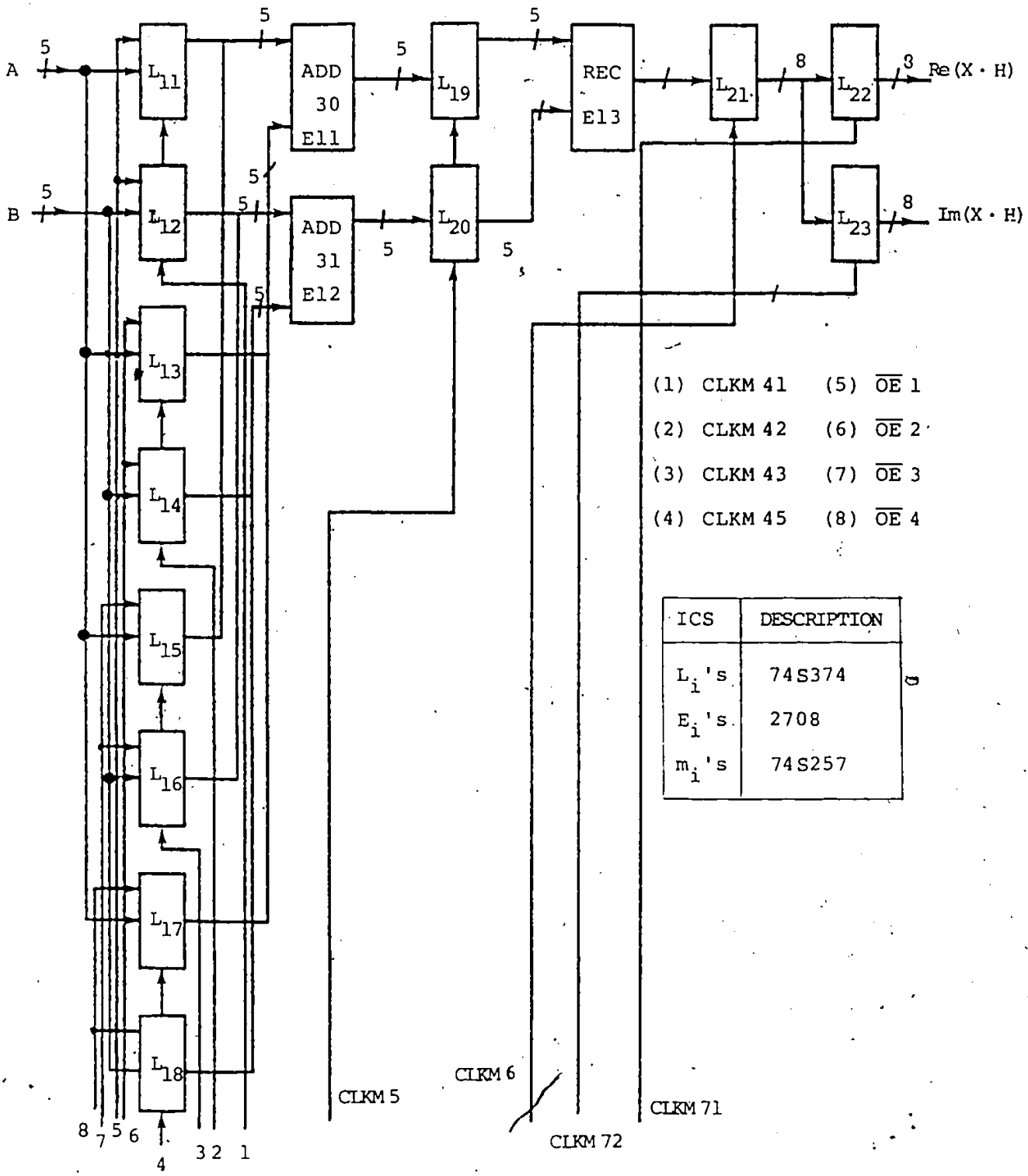


Fig.(4.1) Block Diagram of Complex Multiplier for m_i = 193

(continued on Fig.(4.2)).



ICS	DESCRIPTION
L_i's	74S374
E_i's	2708
m_i's	74S257

Fig. (4.2) Block Diagram of Complex Multiplier for $m_i = 193$

processor as 0.15 MHz.

4.2. HARDWARE IMPLEMENTATION OF A SERIAL COMPLEX MULTIPLIER

The multiplier was built using mainly 2708 EPROM's and 74S374 latches. The access time of the EPROM used is 450 nsec and the settling time of latch is 20 nsec. Fig.(4.1) and Fig.(4.2) show the block diagram of the multiplier. E denotes the 2708 EPROM and L represents the 74S374 latches. The look-up tables are stored in the EPROM's, eg, ADD30 and ADD31 denote the addition look-up tables at sub-moduli 30 and 31 respectively. IND, INV, RCT represent the index, inverse index and reconstruction look-up tables respectively. The SAD30 and SAD31 contain the look-up table for operation $r \cdot A \cdot B$ at submoduli 30 and 31 respectively. The CLKM is the main clock of the multiplier and CLKM1, CLKM2 and CLKM3 are the clocks of the first, second and third stages respectively. FILR and FILI contain the filter response coefficients, which are premultiplied by N^{-1}. The multiplier does one real multiplication at every clock pulse. If RX, RH, IX and IH denote the real and imaginary parts of input and filter response coefficients respectively, then the typical sequence of operation is as shown in fig.(4.3), where $Re(X \cdot H)$ and $Im(X \cdot H)$ are the real and the imaginary parts of the product $X \cdot Y$.

If at the first positive going edge of CLKM3 we obtain the product $RX \cdot RH$, to be latched at L11 and L12 by CLKM41 and which will form the four least significant address bits for addition tables ADD30 and ADD31, then the second positive going edge of CLKM3 we will obtain the product $r \cdot IX \cdot IH$. This will be latched at L13 and L14 by CLKM42 which then

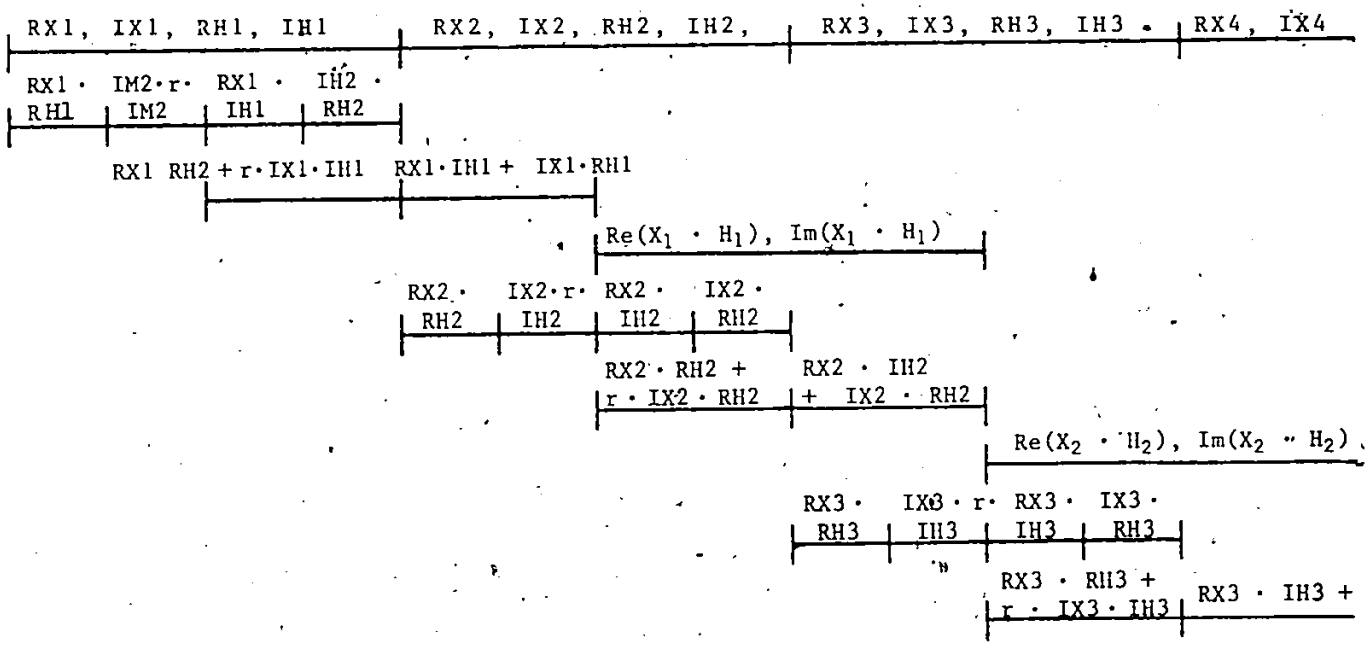


Fig. (4.3) Timing Diagram of a Serial Complex Multiplier

forms the four most significant address bits to the addition tables. At the same time, the output enable input, $\overline{OE}$s of the latches L15 to L19 is held high thus forcing them to the high impedance 'Z' state. The above sequence repeats while computing the imaginary part of product but this time the latches L15 to L18 will latch data with the clocks CLKM43 and CLKM44 and latches L11 to L14 will be in the 'off' state. The complete structure operates in pipeline configuration.

The time available for one complex multiplication is 7 μsec; hence each real multiplication must be performed within 7/4 μsec.

We need multiplexing circuitry at the input of the multiplier to select the inputs either from BUF3 or BUF1 and furthermore to multiplex between the real and the imaginary parts of the input data and filter response coefficient. This is accomplished by using six, quad two-line to one-line tristate multiplexers (74S257's), MM1 to MM6, as shown in Fig.(4.1).

The multiplexing circuitry requires some control signals. EPROM 2708 was used to store the control bits rather than generating control bits using standard sequential/combinational logic. The required pre-calculated control bit pattern can be stored in one EPROM. The address to this EPROM was taken from the modulo 1023 counter as shown in Fig.(4.5). At each count we read the required 8-bit pattern which provides the necessary control inputs to

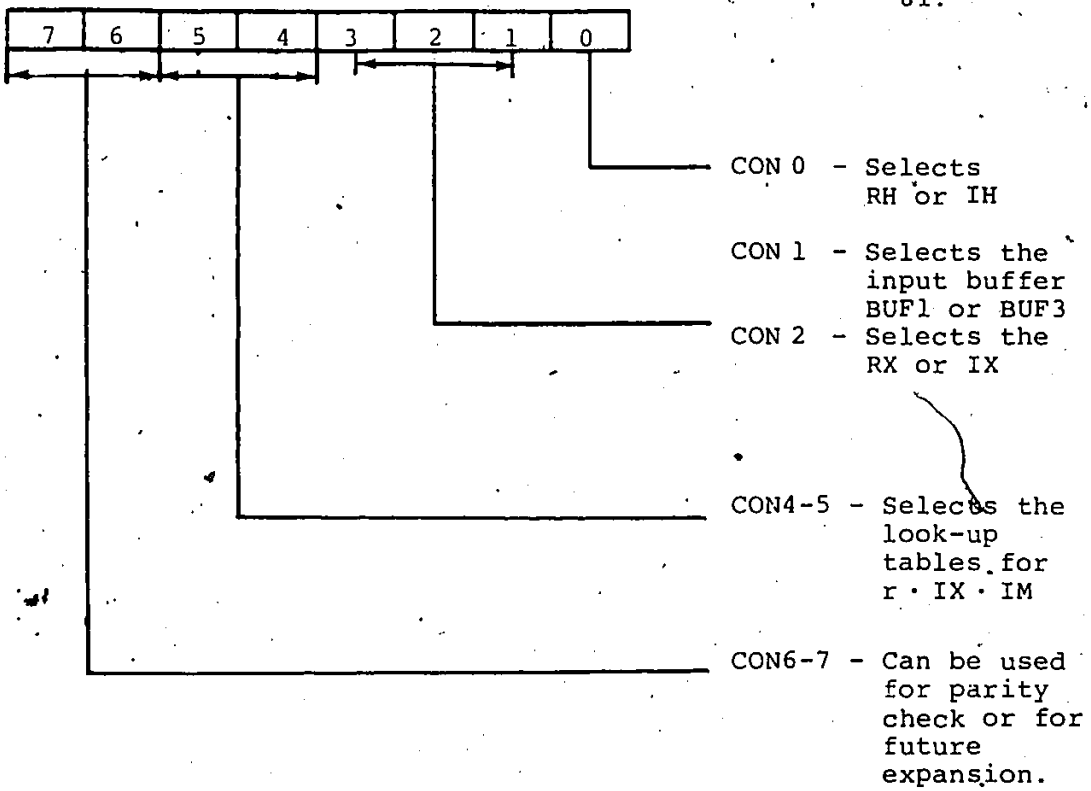


Fig.(4.4) 8-bit Control Word for the Multiplier

Bit Pattern						Functional Details
5	4	3	2	1	0	
0	1	0	1	0	0	Selects, RX B1, RH
0	1	1	0	0	1	Selects, IX B1, IH
0	1	0	1	0	1	Selects, RX B1, IH
1	0	1	0	0	0	Selects, IX B1, RH
0	1	0	1	1	0	Selects, RX B2, RH
0	1	0	1	1	1	Selects, IX B2, IH
0	1	0	1	1	1	Selects, RX B2, IH
1	0	1	0	1	0	Selects, IX B2, IH

Table 4.1 Functional Details of a Multiplier Control Word

the multiplexing circuits and for selection of the SAD's look-up tables during operation of the r · IX · IH.

The control word consists of 8 bits, CON0 - CON7 as shown in Fig.(4.4). CON0 is connected to the select 'S56' of the multipliers MM5 and MM6; this then selects either RH or IH. CON1 is connected to the select input 'S14' of the multiplexers MM1 to MM4, which multiplexes the inputs coming from BUF2 and BUF3 . If CON1 is high BUF3 is selected; if it is low, BUF1 is selected. CON2 and CON3 are used to differentiate between the real and the imaginary parts of the input coming from the selected buffer. These control bits are connected to the output enable inputs, $\overline{OE}$ of MM1-MM2, and MM3-MM4 respectively. When CON2 and CON3 are '01', the real part of an input is selected and when they are '10' the imaginary part is selected. CON4 and CON5 are connected to $\overline{OE}$ inputs of latches L7, L8, L9 and L10 respectively, which then select the look-up table outputs for the operation of r · IX · IH. When CON4 and CON5 are '01', normal addition tables are selected and when they are '10' modified look-up tables, ie, SAD's, are selected. The typical sequence of the bit patterns of control word is as shown in Table (4.1). For example: if BUF1 is selected, the sequence will be 20 , 25 , 21 and 40 and for BUF3 it is 22 , 27 , 23 and 42 .

The filter coefficients are stored in EPROMs FILR and FILI. The address to these EPROMs is taken from the MOD 127 counter as shown in Fig.(4.5). The multiplier has the

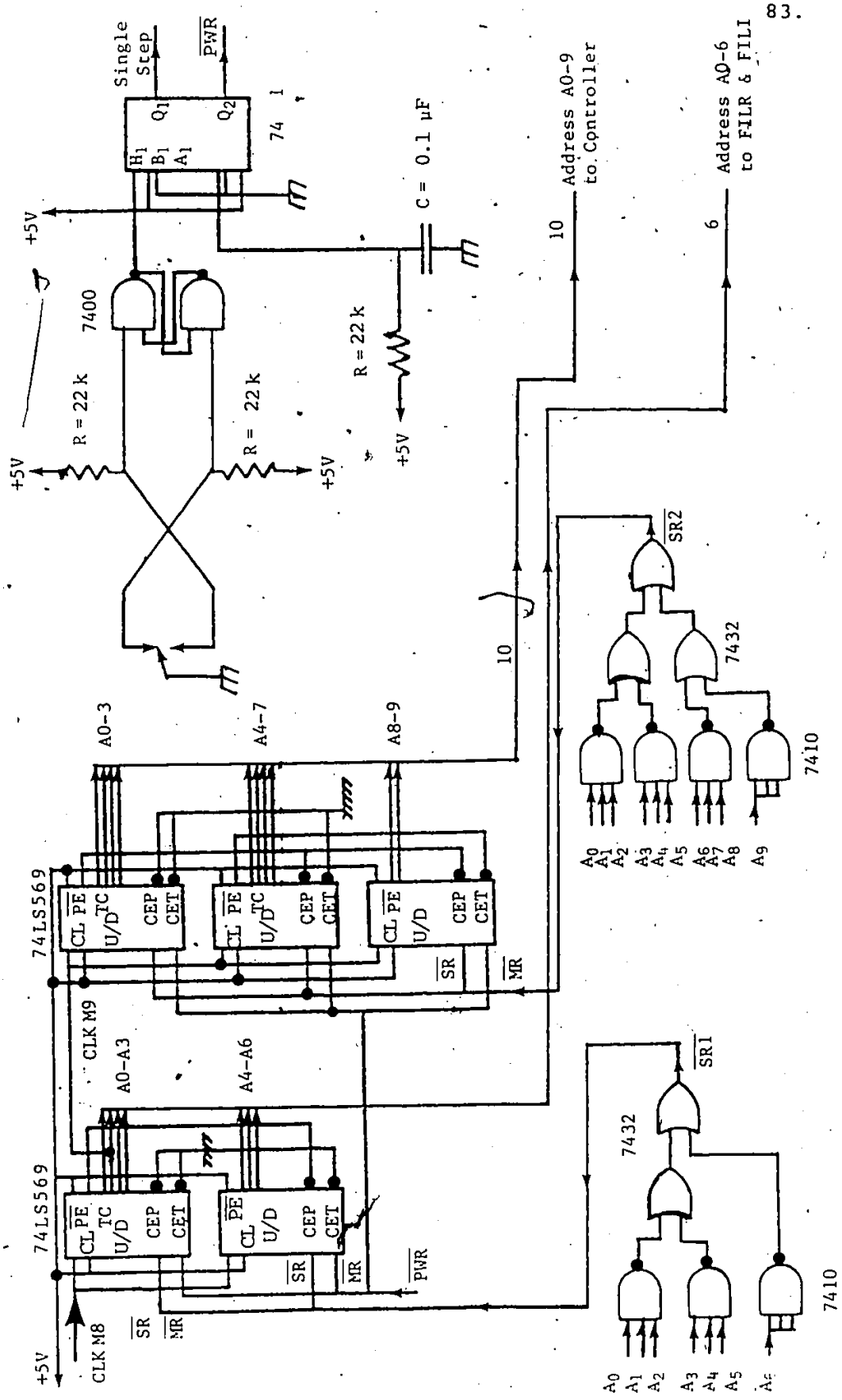


Fig.(4.5) Modulo 127 and 1023 Counters

pipeline delay of three samples.

4.2.1 The Clock Circuitry of the Multiplier

The main clock of the multiplier is CLKM, which is 4/7 MHz. The settling time of the 74S374 is 20 nsec. The clock of each stage has a relative delay of 25 nsec. This delay was obtained by using Hex buffer 7407. The frequency of the clocks of stages 5 and 6, CLKM5 and CLKM6, is half that of CLKM. Fig.(4.6) shows the block diagrams of the clock circuitry. The 4-bit binary counter SN74193 was used. Its outputs A0 and A1, were connected to the A and B inputs of the BCD to decimal (1-of-10) decoder, 7442, respectively. The C and D inputs of 7442 were grounded.

Outputs $\bar{1}$ and $\bar{3}$ from the decoder were inverted using Hex Inverter Buffer/Driver (open collector) 7406 and were used as the clocks CLKM71 and CLKM72 of the seventh stage respectively. The A0 output of 74193, after introducing a relative delay of 25 nsec and inverting using 7406 was then used as a clock in fifth and sixth stages, viz, CLKM5 and CLKM6.

The decoder's outputs $\bar{0}$, $\bar{1}$, $\bar{2}$ and $\bar{3}$ were inverted using 7406's and a relative delay of 25 nsec with respect to CLKM5 was introduced. These outputs were used as the clocks to fourth stage, namely CLKM41, CLKM42, CLKM43 and CLKM44. The output enable signals $\overline{OE}1$, $\overline{OE}2$, $\overline{OE}3$ and $\overline{OE}4$ were generated by using quad two-input AND gates as shown in Fig.(4.6). Fig.(4.7) shows the various clocks used for the multiplier.

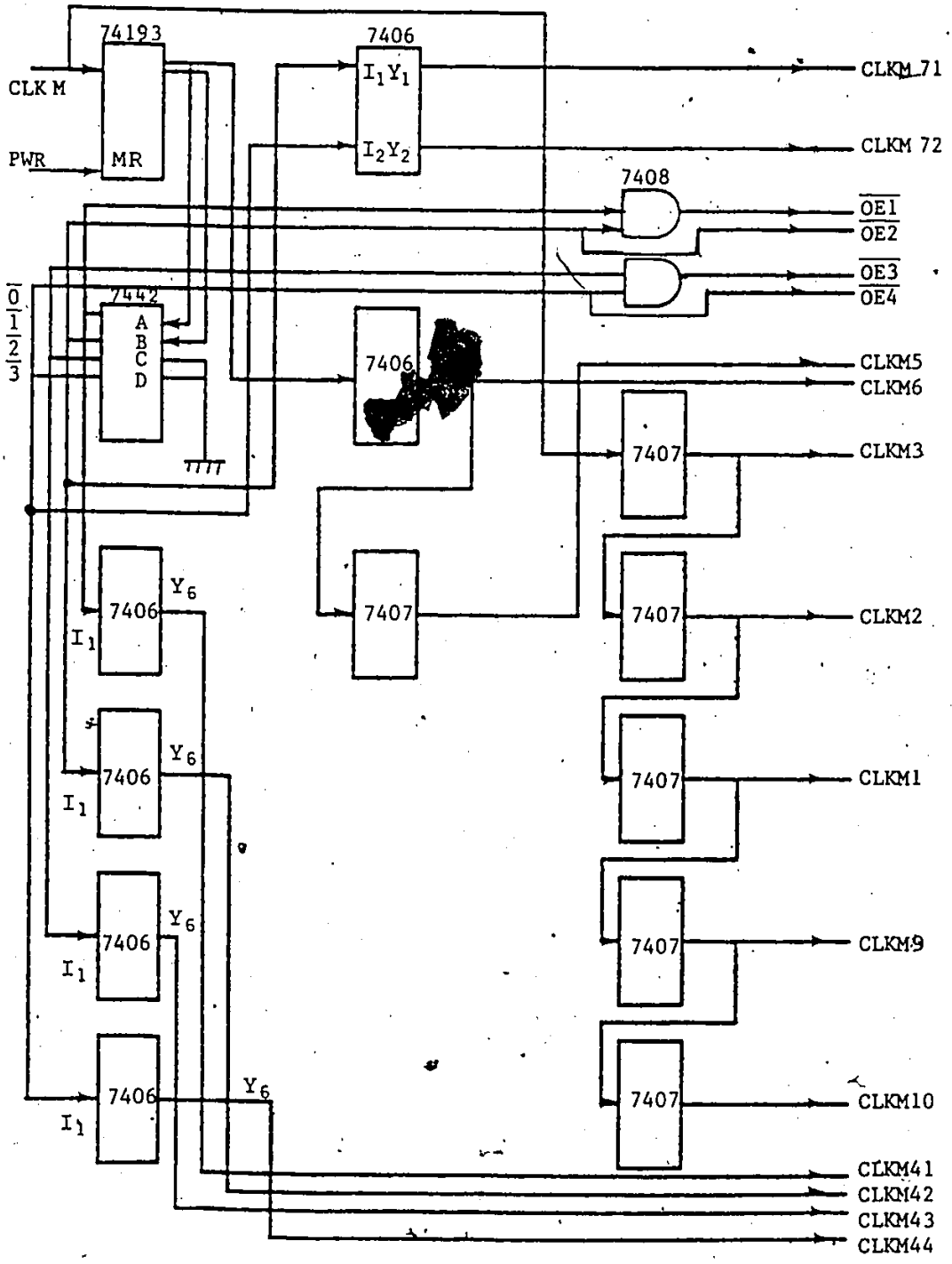


Fig.(4.6) Clock Circuitry of the Multiplier

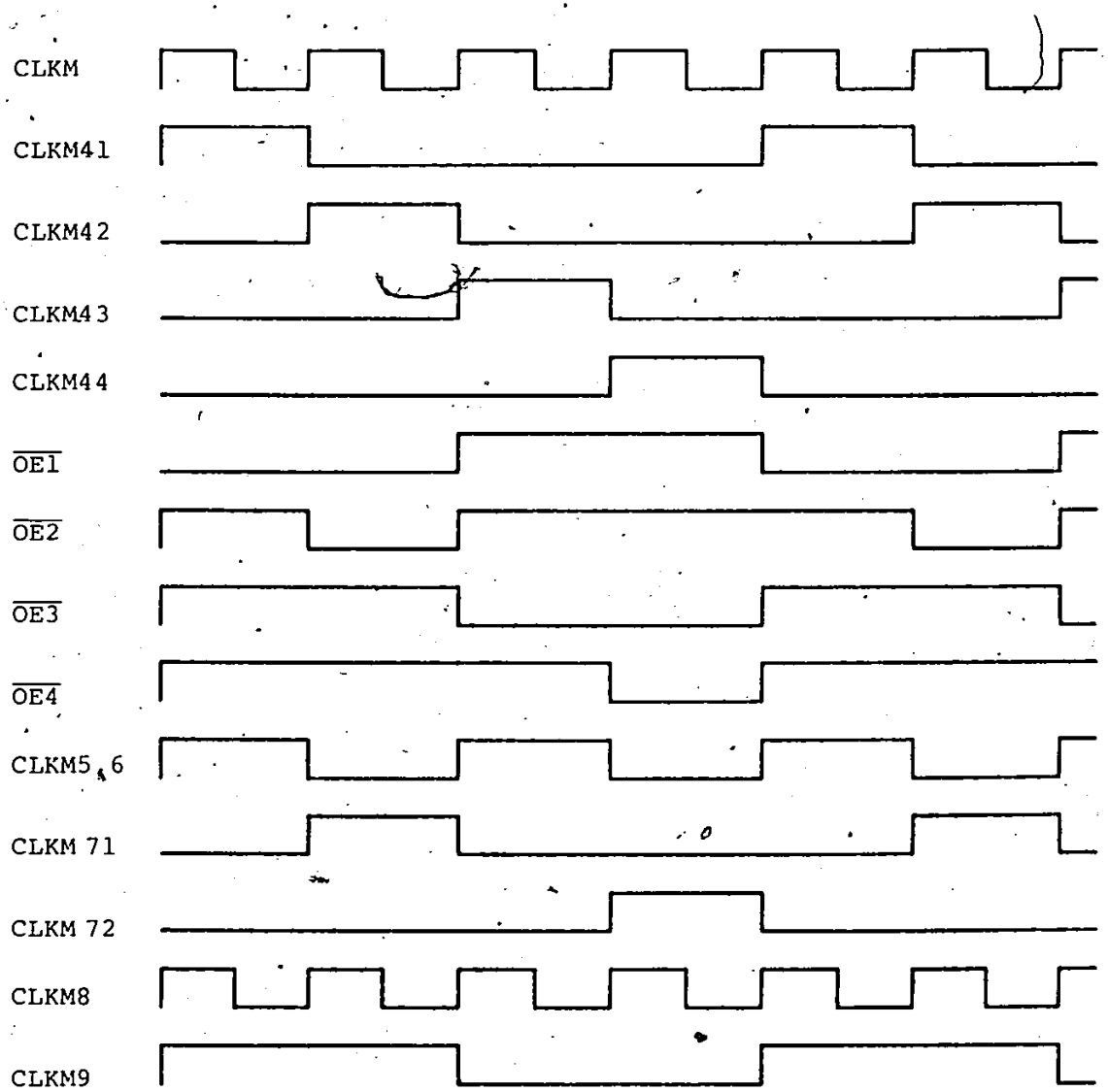


Fig. (4.7) Various Clock Pulses for Multiplier

Fig.(4.6) shows the block diagram of the modulo 127 and modulo 1023 counters. The 74LS569 was used as a building block to construct these counters. The '569' is a synchronous 4-stage modulo 16 binary UP/DOWN counter featuring preset capability for programmable operation, carry look ahead for easy cascading and U/$\bar{D}$ input to control the direction of counting. It has 3-state drivers at outputs, hence direct interfacing to bus organized system is possible. It also has synchronous Reset and asynchronous Master Reset capabilities. The clocks CLKM8 and CLKM9 shown in Fig.(4.7) were used to drive the modulo 127 and modulo 1023 counters respectively. These modulo counters were developed using 74LS569's, Triple three input NAND gates 74S10's and quad two input OR gates 7432's. The low going signals were generated on detecting counts 127 and 1023 and were used as the synchronous reset inputs to the modulo 127 and 1023 counters. The dual retriggerable monostable multivibrator 74123 was used to generate the power on reset signal $\overline{PWR}$, and the single stepping pulse as shown in Fig.(4.5). The $\overline{PWR}$ was connected to Master Reset, $\overline{MR}$, inputs of the 74LS569's.

4.2.2 The Hardware Requirement of Multiplier

The typical IC count of the multiplier is given in Table 4.2.

Description of IC	No. of IC's
2708	16
74S374	26
74S257	6
74S569	5
7407	5
7406	8
74193	1
74123	1
7432	2
74S10	4

Table 4.2 The Hardware Requirement of Multiplier

4.3 HARDWARE IMPLEMENTATION OF MEMORY ORGANIZATION

The processor's memory organization is made up of a four-buffer structure as discussed in previous chapter. Each buffer is of length 128 points. The wordlength is 16 bits. The eight least significant bits store the real parts and the eight most significant bits store the imaginary parts. Each buffer is further divided into a four shift registers of length 32 points each, which can be connected in series so as to form a long shift register of 64 points or even 128 points.

The 32 x 8 First-In-First-Out (FIFO) memories namely Am2812A were used as the shift registers. The FIFO can have independent read and write operations and can be easily cascaded to form a shift register of any size. It can operate up to the data rate of 1 MHz. It requires three power

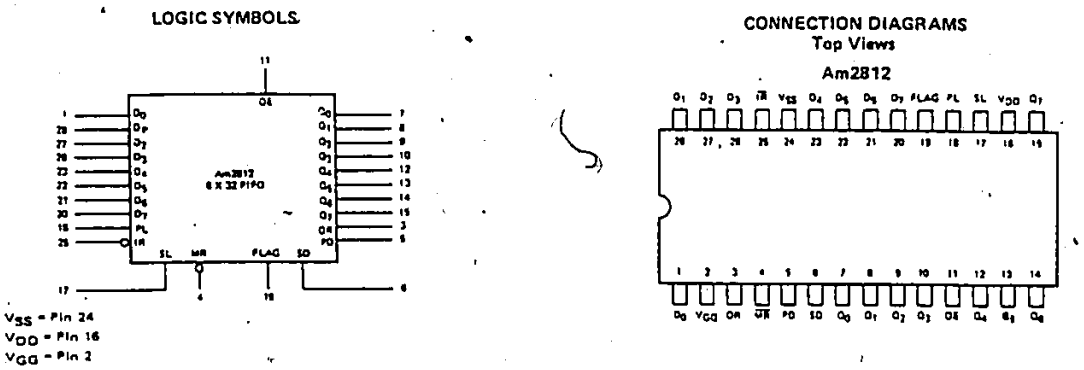


Fig. (4.8) Logic Diagram and Pin Configuration of AM2812A

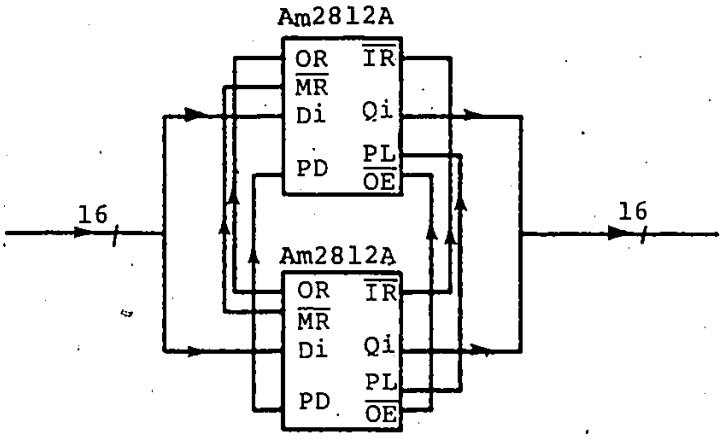


Fig. (4.9) Interconnection of FIFO's for Size 32 x 16

supplies V_{SS}, V_{DD} and V_{CC} which are 5 Volts, GND and -12 Volts respectively. The signals, input ready, $\overline{IR}$ and output ready, OR of FIFO can be used for cascading the FIFOs. In write mode, data on the data inputs (Di) is written into the memory by a pulse on load (PL). The data ripples through the memory until it reached the output or another data word. The data is read from the memory by applying shift-out pulse on PD. This dumps the word on the output (Qi) and the next word in the buffer moves to the output Fig. (4.8) shows the logic symbol and pin connection diagram of FIFO Am2812A. Fig.(4.9) shows the FIFO connections to form the shift register of size 32 x 16.

4.3.1 Memory Buffer Interconnections

In chapter 2, section (2.6.2) we discussed the buffer interconnections of an NTT processor during various stages of computation, Fig.(4.10) shows the typical interconnections of FIFO's in the first stage of NTT. Any control signal of the shift register is denoted by X_{ij}, where X represents any of the control signals, PD or PL or OR or $\overline{IR}$; i denotes the buffer number to which this shift register belongs and j represents the position of the shift register; e.g., OR12 will represent the output ready signal of the second shift register of BUF1.

In the first stage the PD22 and $\overline{IR}$21, OR22 and PL21 are interconnected, PL22 is grounded. This connects FIFO #1 and FIFO #2 serially as shown in Fig.(4.9). Similar interconnections are done with FIFO #3 and FIFO #4. The shiftout pulse

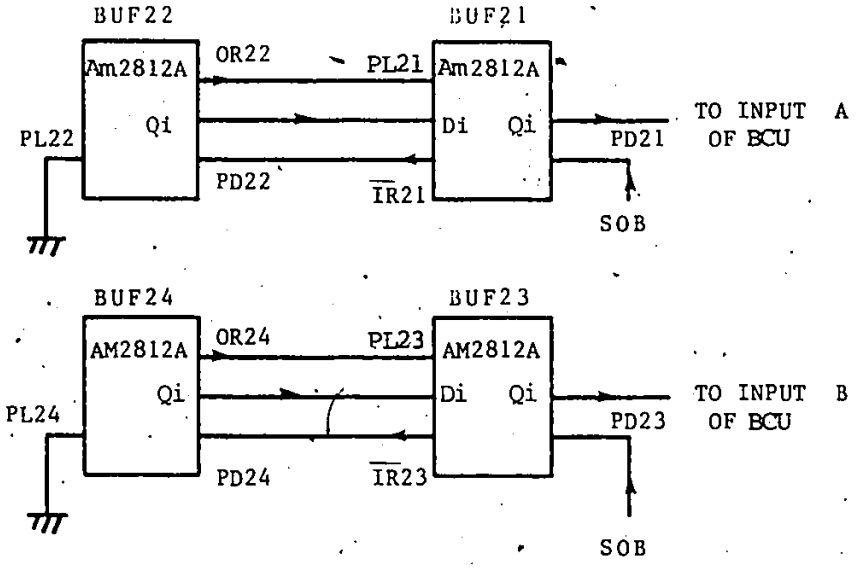


Fig. (4.10) FIFO Interconnections for 1st Stage of an NTT with SOB is Shift Out Pulse

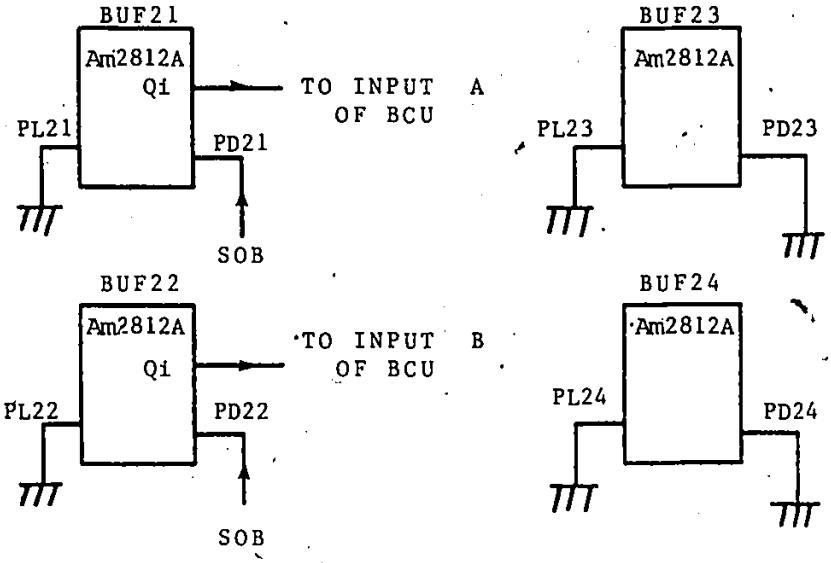


Fig. (4.11) FIFO Interconnections for 2nd to (N-1)th Stage

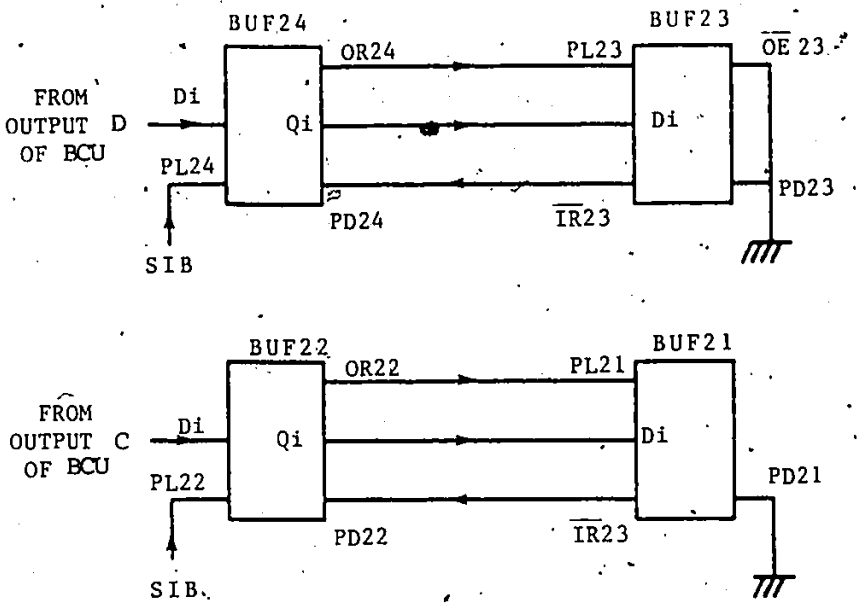


Fig. (4.12) FIFO Interconnections for Storing the BCU Output with SIB is shift Out Pulse

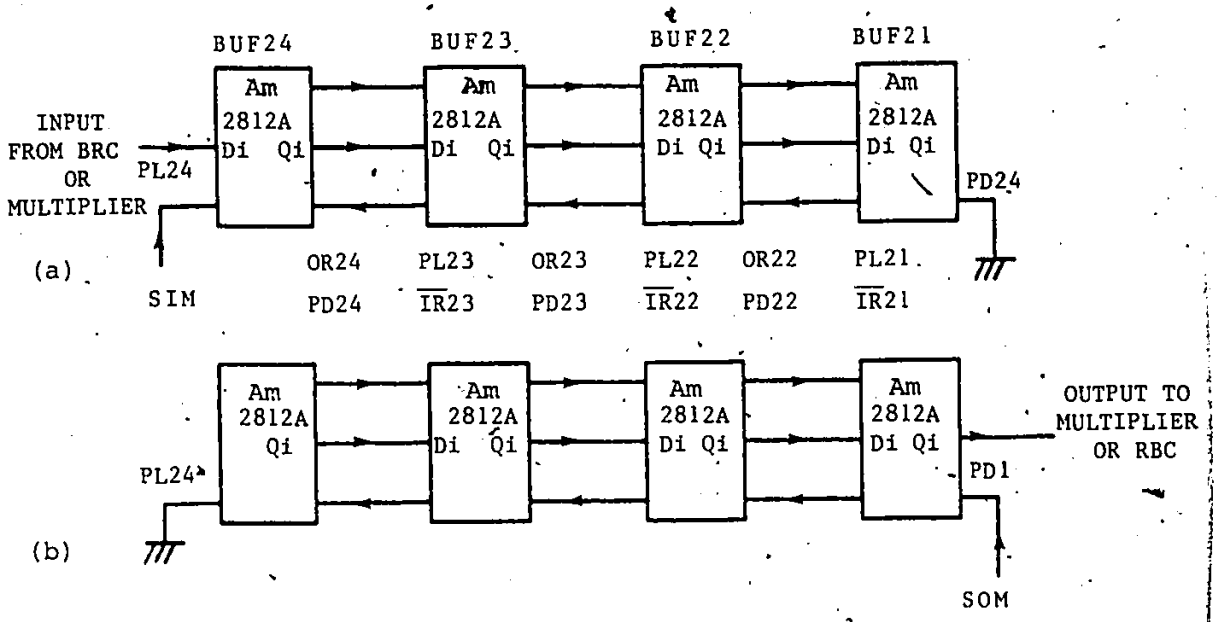


Fig. (4.13) FIFO Interconnections for Input/Output with SIM and SOM are Shift In and Shift Out Pulse

is applied to PD21 and PD23.

In any subsequent stage, the SR1 and SR2 of submemory SM1 or SM2 of the selected buffer give input to the BCU. Fig.(4.11) shows the typical condition when FIFO #1 and FIFO #2, i.e., SR1 and SR2 of submemory SM1 are giving input to the BCU. PL21 and PL22 are grounded and shift out pulse is applied to PD21 and PD22. The PL23, PD23, PL24 and PD24 of submemory SM2 are grounded as shown in Fig.(4.11).

While storing the output from the BCU, that at OUTPUT C is shifted in SR2 of submemory SM1 and simultaneously that at OUTPUT D is shifted to SR2 of SM2 of the selected buffer. SR1 and SR2 of a submemory are serially connected so as to form the shift register of the length 64 points. Fig.(4.12) shows the FIFO's interconnections while storing the output from BCU. The PD22 and $\overline{IR}$21, PD24 and $\overline{IR}$23, OR22 and PL21 and OR24 and PL24 are interconnected. The PD21 and PD23 are grounded. The shift in pulse is applied to PL22 and PL24 as shown in Fig.(4.12).

The shift registers, SR's of the selected buffer are serially interconnected so as to form a shift register of length 128 points during an INPUT/OUTPUT or multiplication operation. Fig.(4.13) shows the FIFO interconnections. While giving input to the multiplier or to the RBC the shift out pulse is applied to PD21 with PL24 grounded. The shift in pulse applied to PL24 and PD21 is grounded when BUF1 is storing the output of BRC or multiplier.

A set of multiplexers, SWMs assigned to each buffer, performs the necessary FIFO interconnections. Fig.(4.14)

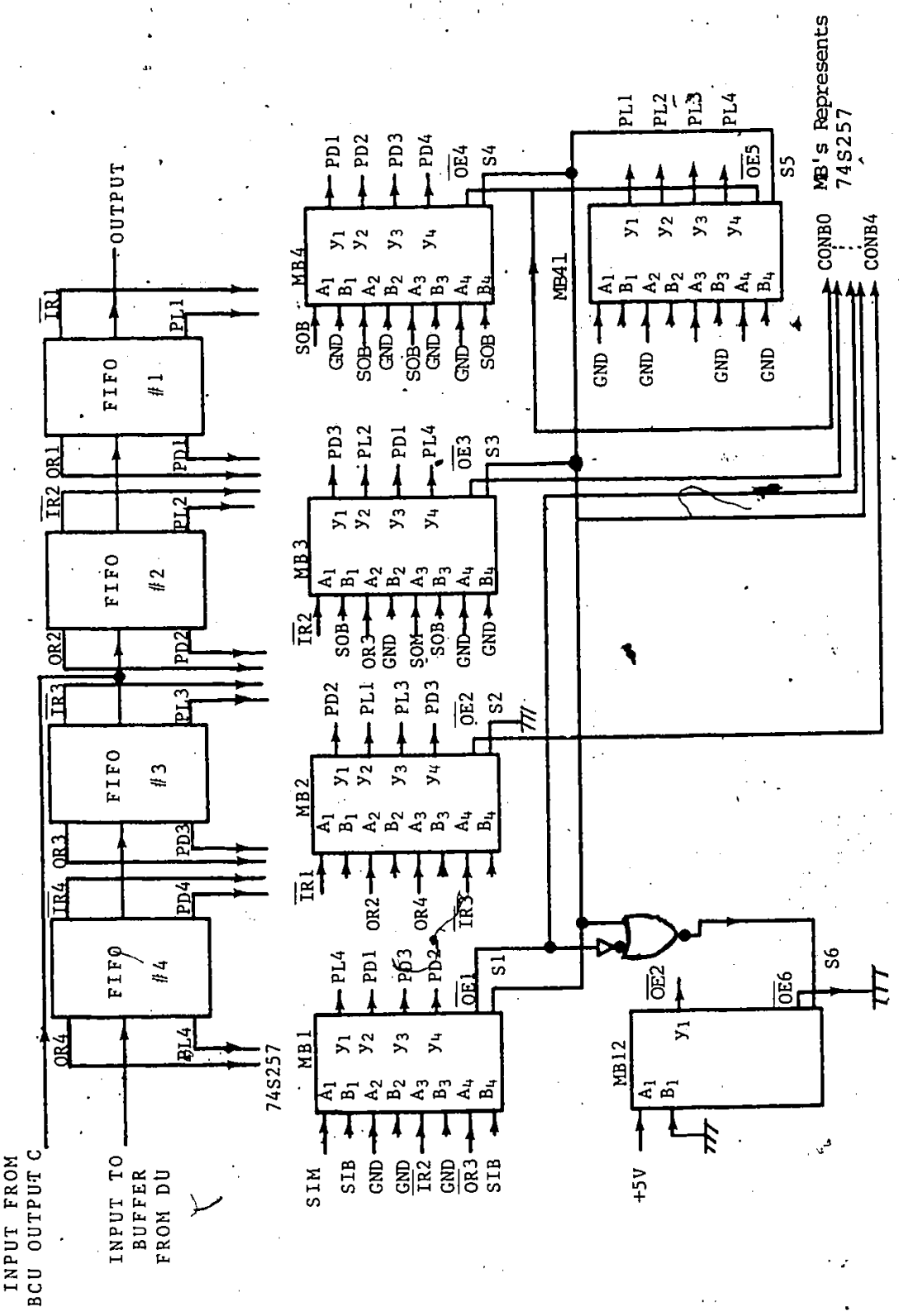


Fig. (4.14) Buffer Interconnections Using Multiplexers SWM1

shows the IC block diagram of the buffer interconnections using multiplexers for BUF2. Each set is made up of six 74S257's. The control signals are provided to select input 'S' and to the output enable '$\overline{OE}$' of the multiplexers. The control bit pattern for each buffer was stored in four 2708's with a total storage capacity of 4k x 8 bits.

The control word, CONB0 - CONB7, is made up of 8-bits and controls the FIFO interconnections, the operation of the distributor unit and the operation of the output multiplexing unit. Fig.(4.15) shows the details of the control word. The least significant digit CONB0 of control word is connected to output enable inputs $\overline{OE}4$ and $\overline{OE}5$ of MB4 and MB42. CONB1, CONB2 and CONB4 are connected to $\overline{OE}3$, $\overline{OE}1$ and $\overline{OE}2$ of MB3, MB1 and MB2 respectively. The select input, S, of the multiplexers MB1, MB2, MB4 and MB42 are tied together and CONB3 connected to them. The $\overline{OE}6$ of MB12 is grounded and the select input signal S6 of MB12 is derived from CONB3 and CONB2 as shown in the Fig.(4.14).

Table (4.3) shows the functional details of the control word. When CONB2 to CONB4 are high and CONB0 and CONB1 are low, the multiplexers interconnect the FIFO's for storing the output data from the BRC or the multiplier. These interconnections are achieved by forcing MB3, MB4 and MB42 to high impedance state and selecting A inputs of the MB1 and MB2.

While storing input from the BCU the CONB0, CONB1 and CONB3 are high and CONB2 and CONB4 are low. This selects the B input of MB1 and forces MB3, MB4 and MB42 to 'Z' state.

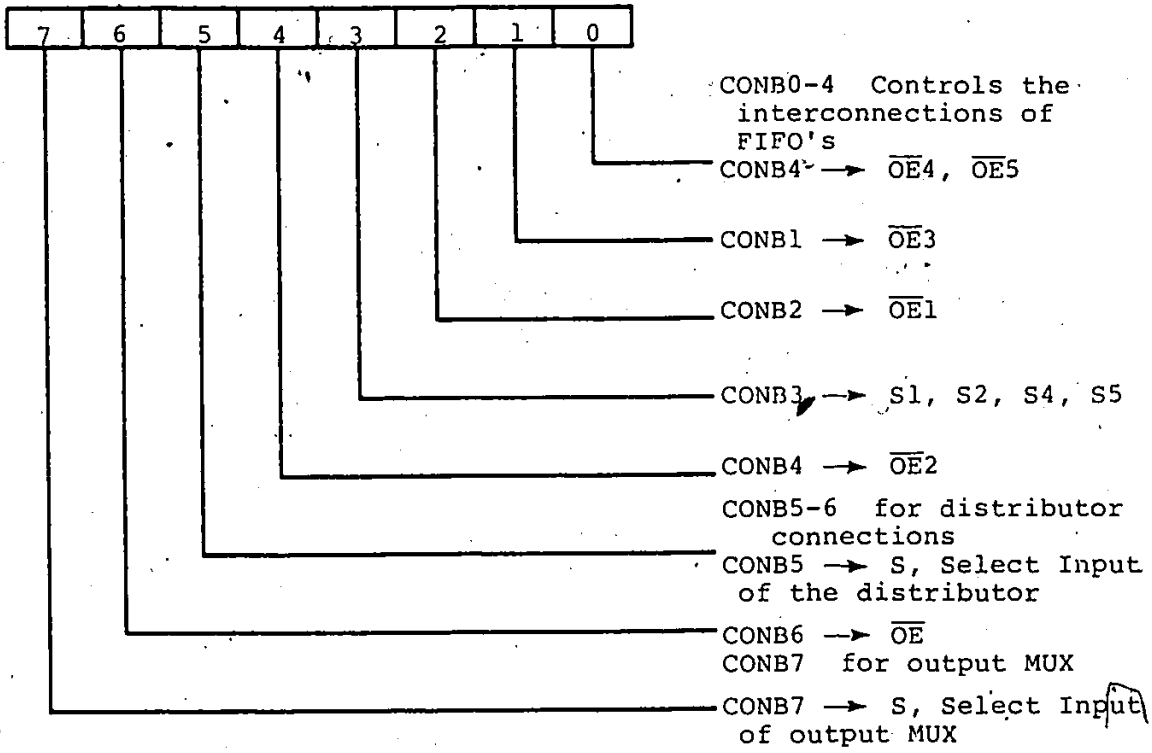


Fig. (4.15) Control Word for Memory Organization

Bit Pattern					Functional Details
4	3	2	1	0	
0	0	0	1	1	Stores the Output of BRC or Multiplier
0	1	0	1	1	Stores the Output of BCU
0	0	1	0	1	Stores Input to Multiplier/Output MUX
0	1	1	0	1	Stage one of NTT
1	0	1	1	0	Gives Input to BCU from SM1
1	1	1	1	0	Gives Input to BCU from SM2

Table 4.3 Bit Patterns of Memory Control Word

The OE23 is held low by selecting B input of MB12. This makes necessary FIFO connections for storing the BCU output as shown in Fig.(4.13 a) .

The MB4, MB42 and MB1 are forced to high impedance state and A input of MB2 and MB3 are selected by bit pattern of 00101 as shown in Table (4.3), which applies the shift out pulse SOM to PD21 and grounds the PL24. This makes the necessary connections for giving input to the multiplier or RBC as shown in Fig.(4.13 b).

The bit pattern is 01101 while giving input to the BCU at the first stage of NTT which selects B inputs of MB3 and forces MB1, MB3, MB4 and MB5 in 'Z' state. The shift out pulse, SOB, is applied to PD21 and PD23 while PL22 and PL24 are grounded as shown in Fig.(4.10).

During the second to the seventh stage of an NTT calculation, the input to BCU is taken either from FIFO #1 and FIFO #2 or from FIFO #3 and FIFO #4 depending on the stage. The bit pattern 10110 selects the input from FIFO #1 and FIFO #2. The bit pattern 11110 selects output from FIFO #3 and FIFO #4. During this process, MB4 and MB42 are functioning and the other multiplexers are in the high impedance state. Fig.(4.11) shows the necessary connections.

The control bit pattern for each buffer is stored in four 2708's. The addresses of these EPROMs were taken from the modulo 4095 counter. Fig.(4.16) shows the circuit diagram for the control unit with modulo 4095 counter. This counter was constructed by using three 74LS569's. The three

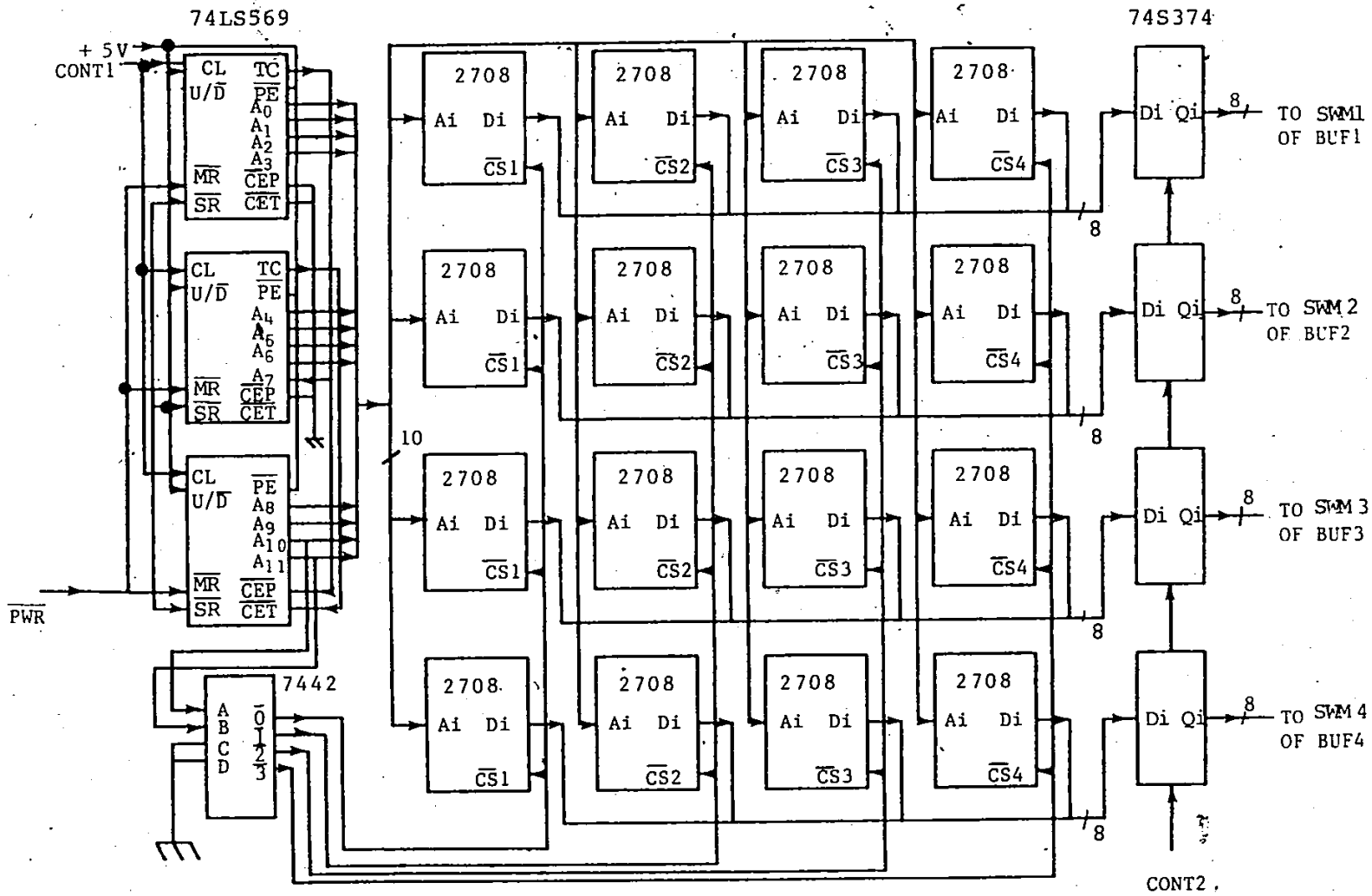


Fig. (4.16) Buffer Interconnection Controller Using Look-up Tables

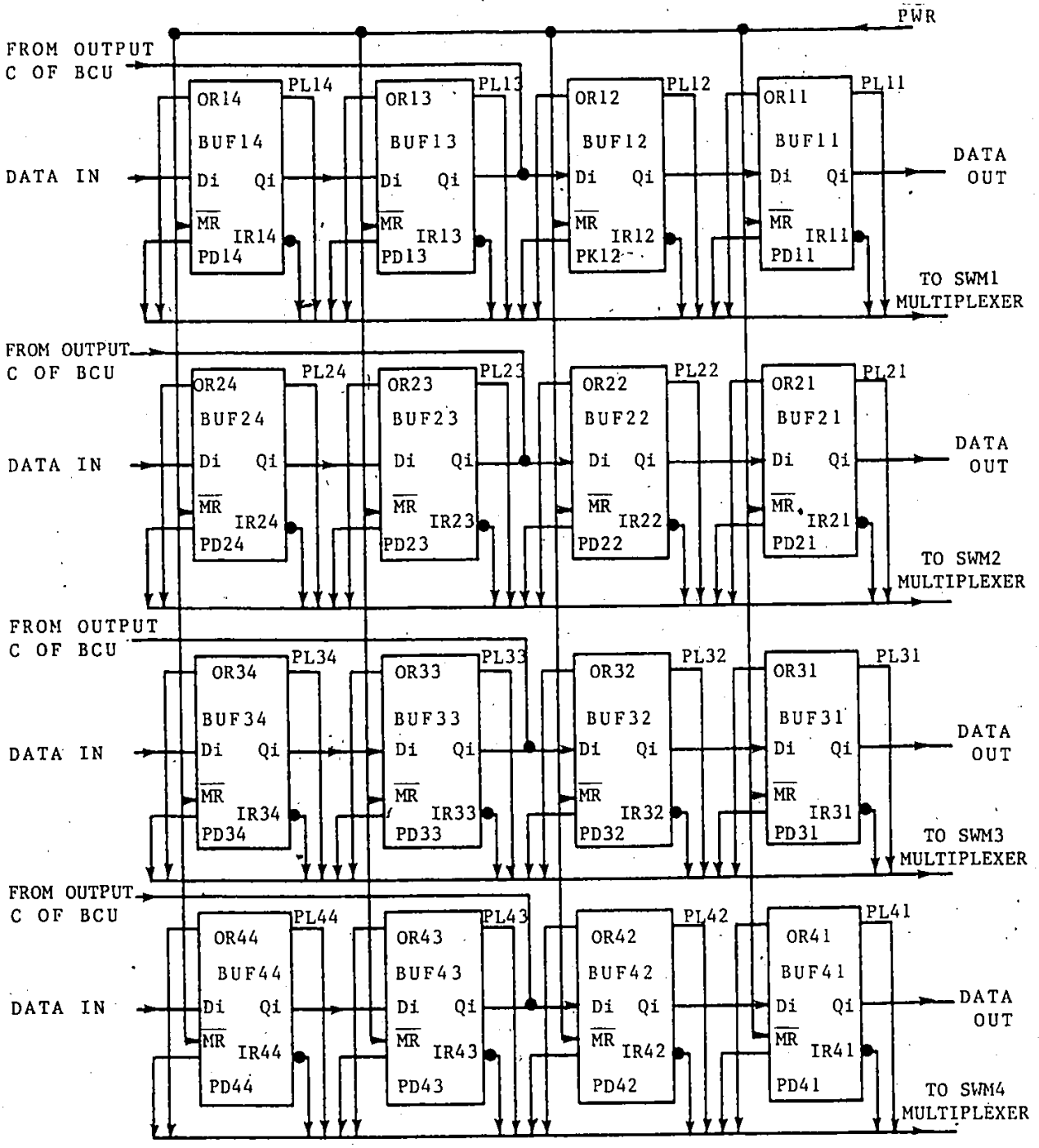


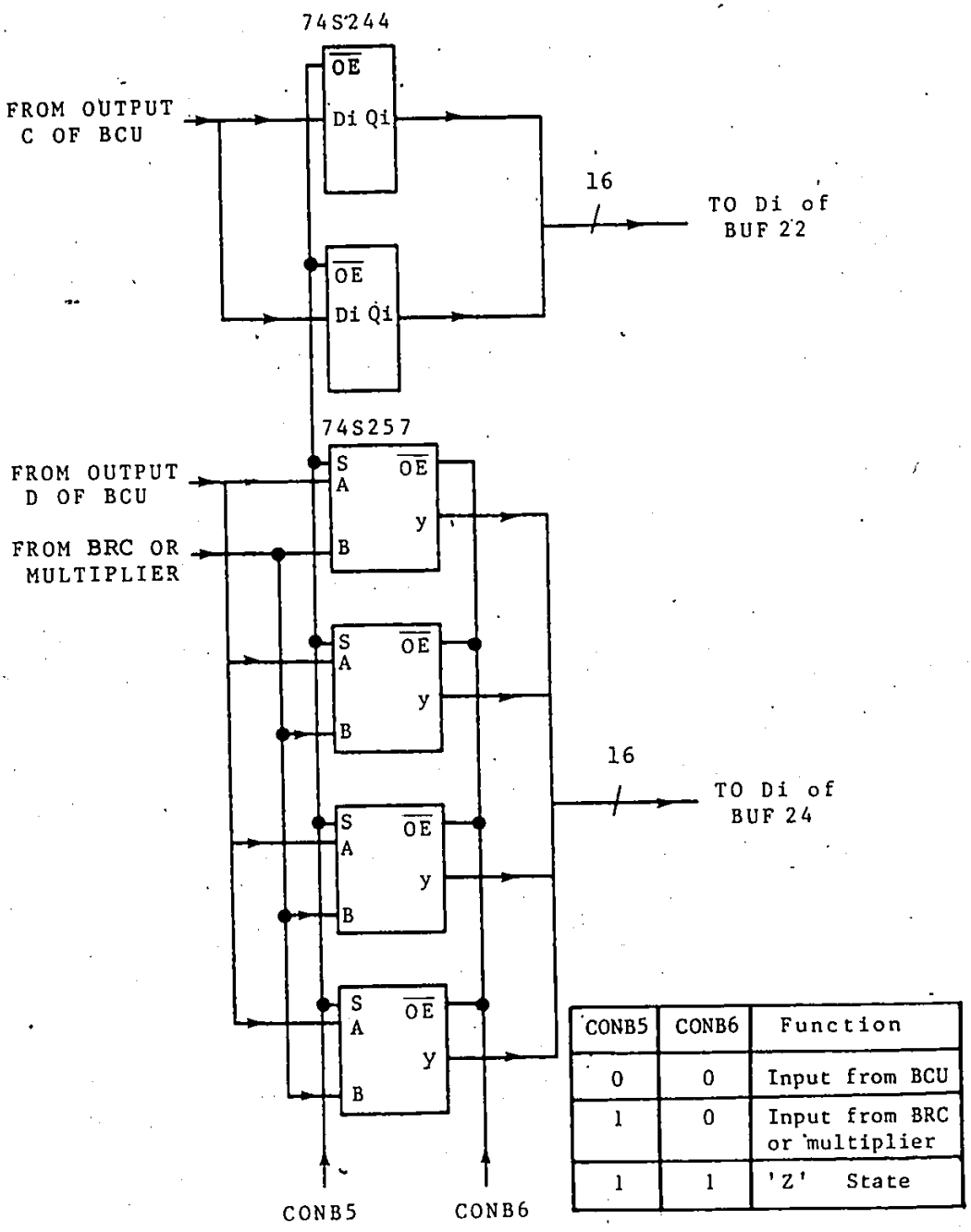
Fig. (4.17) Block Diagram of a Memory Organization

sample pipeline delay of the multiplier and the five sample pipeline delay of the BCU was taken into consideration. These controls were stored for the two cycles of an NTT processor.

The serial input, SI, and serial output, SO, were grounded. The power on reset signal, $\overline{PWR}$ is applied to $\overline{MR}$. The FLAG output was not used. The input to memory buffers coming from the BCU, BRC or the multiplier is TTL compatible. The typical output high voltage, V_{OH}, of tri-state TTL, IC 2.4 volts and input high level, V_{IL}, of Am2812A is 4 volts which creates a problem. As both these devices are operating from a 5 Volt supply, the pull-up resistors of 10kΩ were connected to the TTL output and $+V_{CC}$. This makes certain that the logic-1 level of the TTL gate exceeds the 4 volts. The output of Am2812A are fully compatible with the input characteristics of TTL gates. Fig.(4.17) shows the block diagram of a memory organization consisting of 32 Am2812A's.

4.4 HARDWARE IMPLEMENTATION OF THE DISTRIBUTOR UNIT

The distributor unit essentially consists of a set of multiplexers assigned to each memory buffer, namely DM1, DM2, DM3 and DM4. Each unit must select between the input from the BCU, from the multiplier or from the BRC. Fig. (4.18) shows the block diagram of the DM2, i.e., a set of multiplexers assigned to BUF2. The DM1, DM3 and DM4 essentially have a structure similar to that of DM2. It is made up of four 74S257's and two tri-state octal buffers, 74S244. OUTPUT C of the BCU is connected to the octal



CONB5	CONB6	Function
0	0	Input from BCU
1	0	Input from BRC or multiplier
1	1	'Z' State

Fig. (4.18) Block Diagram of Distributor Unit for BUF2

buffer inputs and octal buffer outputs are connected to Di inputs of FIFO #2. OUTPUT D of the BCU is connected to the input A's of the multiplexers, DM21 to DM24. The multiplier or BRC output is connected to the B inputs of the multiplexers. The outputs of the multiplexers are connected to the Di inputs of FIFO #4 as shown in Fig.(4.18). The $\overline{OE}$s of the 74S244's and tied to the S inputs of the multiplexers. The functioning of the distributer unit is controlled by the sixth and seventh bits, i.e., by CONB5 and CONB6, of the control word. The CONB5 and CONB6 are connected to the S and $\overline{OE}$ inputs of the multiplexers as shown in Fig.(4.18). When CONB6 and CONB7 are 00, the input is selected from the BCU. When they are 01, the input is selected from the BRC or from the multiplier. These individual distributor units can be forced to high impedance state when CONB6 and CONB7 are 11.

4.5 HARDWARE IMPLEMENTATION OF THE OUTPUT MULTIPLEXING UNIT

The memory buffer, BUF2 or BUF4, stores the filtered output of an NTT processor. The output multiplexing unit therefore must select either BUF2 or BUF4, depending upon the cycle of operation, then connect it to the output bus. It is made up of four 74S257 IC's. The output coming from BUF2 is connected to the A inputs of the multiplexers. That coming from BUF4 is connected to the B inputs of the multiplexers. The most significant bit, CONB7, of the control word is connected to the select input, 'S', at the multiplexer. When it is low, BUF2 is routed to an output bus.

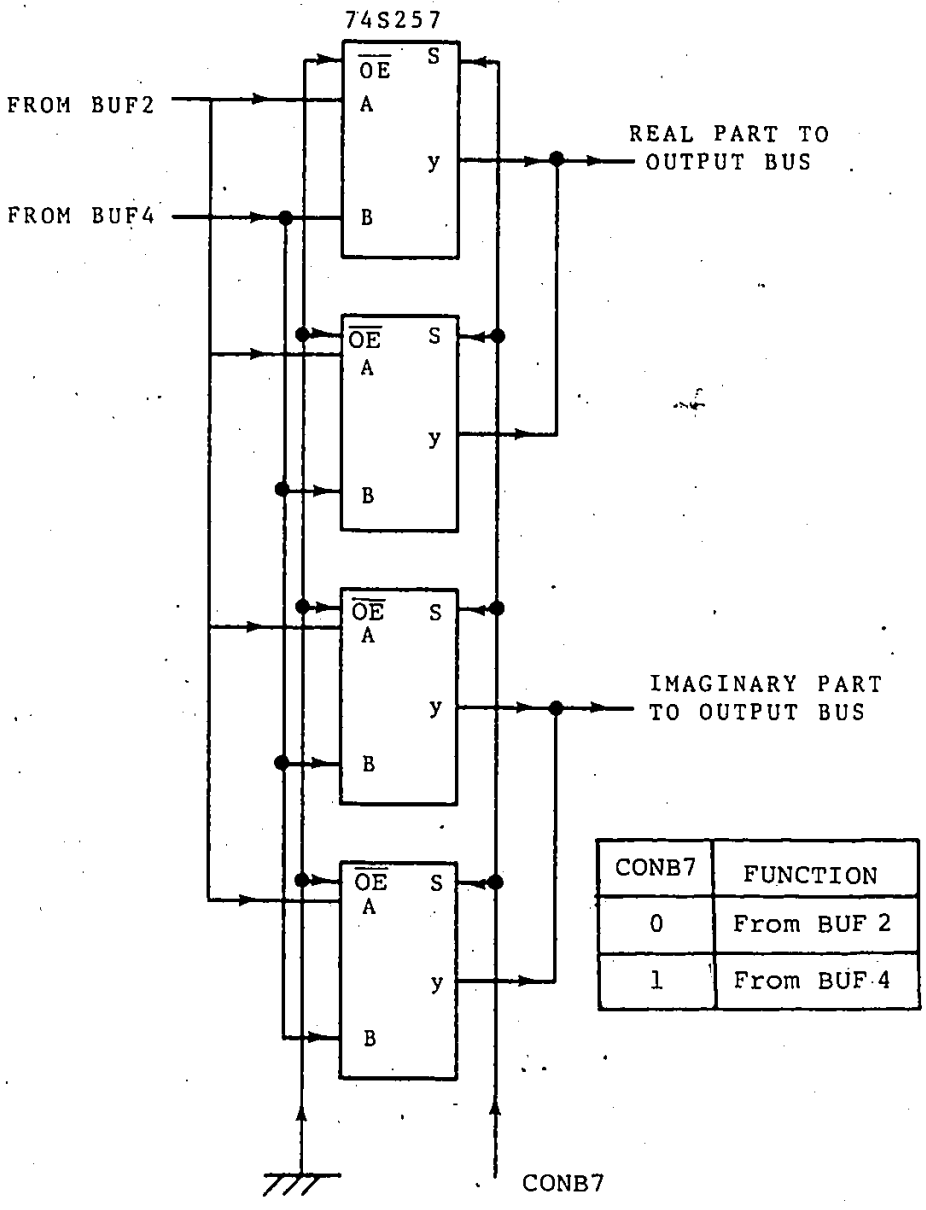


Fig. (4.19) Block Diagram of Output Multiplexer

When it is high, BUF4 is connected to the output bus. Fig. (4.19) shows the block diagram of the output multiplexing circuitry.

4.6 THE BCU INPUT MULTIPLEXING UNIT

In the previous chapter we have seen that the BCU requires two inputs, namely INPUT A and INPUT B. We have also seen that any of the four buffers can deliver input to the BCU. For example, if BUF2 contains the input to the BCU then in the first stage of NTT, INPUT A will receive input from BUF21, ie, FIFO #1. INPUT B will receive input from BUF23, ie, FIFO #3. In subsequent stages INPUT A can receive input from BUF21 or BUF31 and INPUT B can receive input from BUF22 or BUF24 as shown in Fig.(4.20). Hence we need some multiplexing circuitry at the input of the BCU which must first select the buffer containing the input to BCU and then select the appropriate FIFO's.

This multiplexing unit can be implemented using a number of octal buffers each assigned to a buffer and providing the control signal to its output enable input. Therefore each sub-multiplexing unit can be implemented using ten 74S244 IC's. Fig.(4.21) shows the multiplexing circuitry of BUF2. Instead of using conventional sequential/ combinational logic for generating the control signals, the look-up table technique was used. The pre-calculated control bit pattern was stored in EPROMs. Fig.(4.22) shows the block diagram of the BCU multiplexing unit consisting of forty 74S244 IC's.

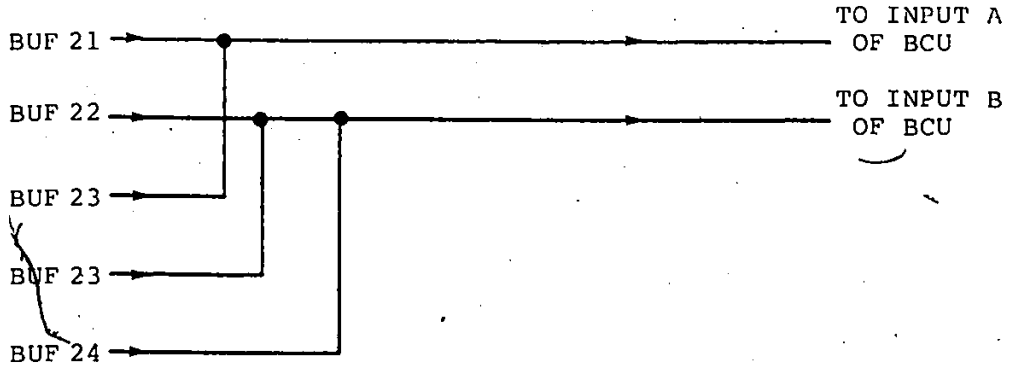


Fig. (4.20) Input Connections to BCU from BUF2

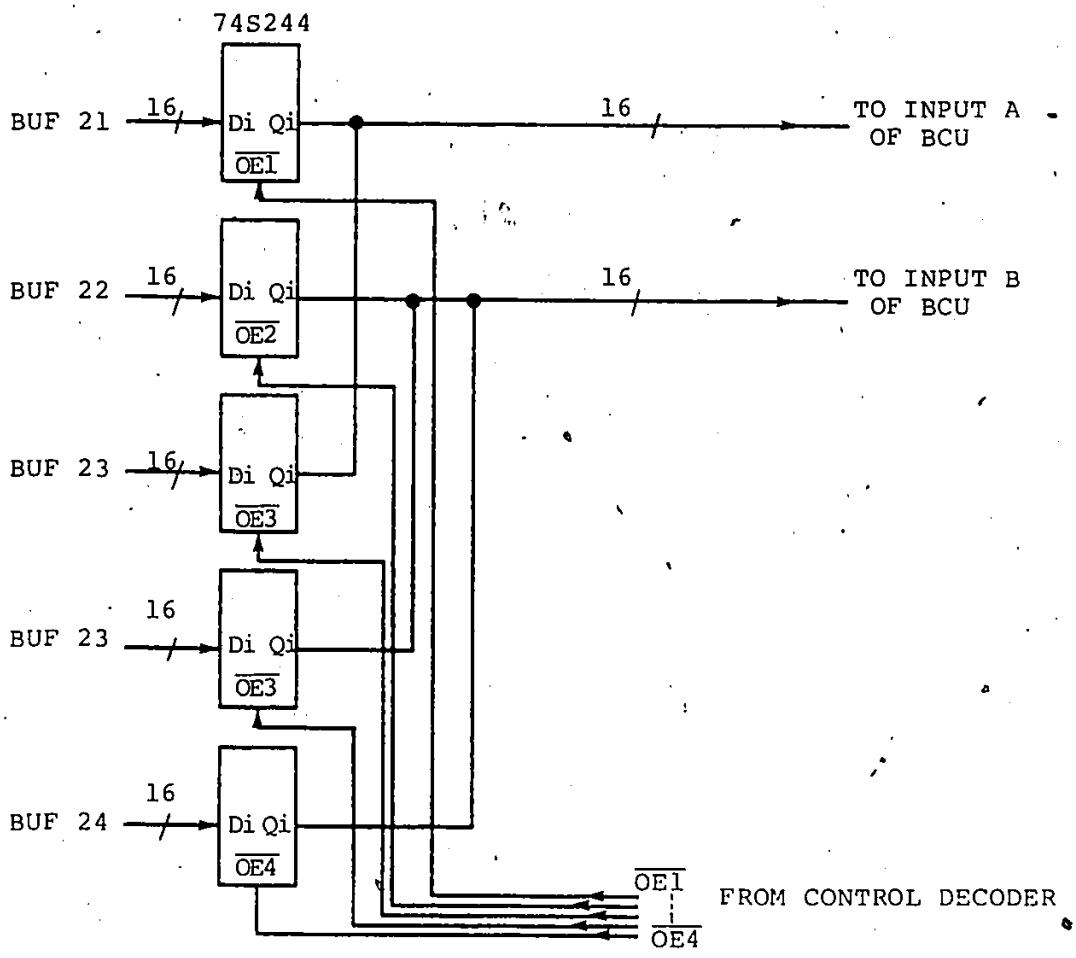


Fig. (4.21) Multiplexing Unit for BUF2

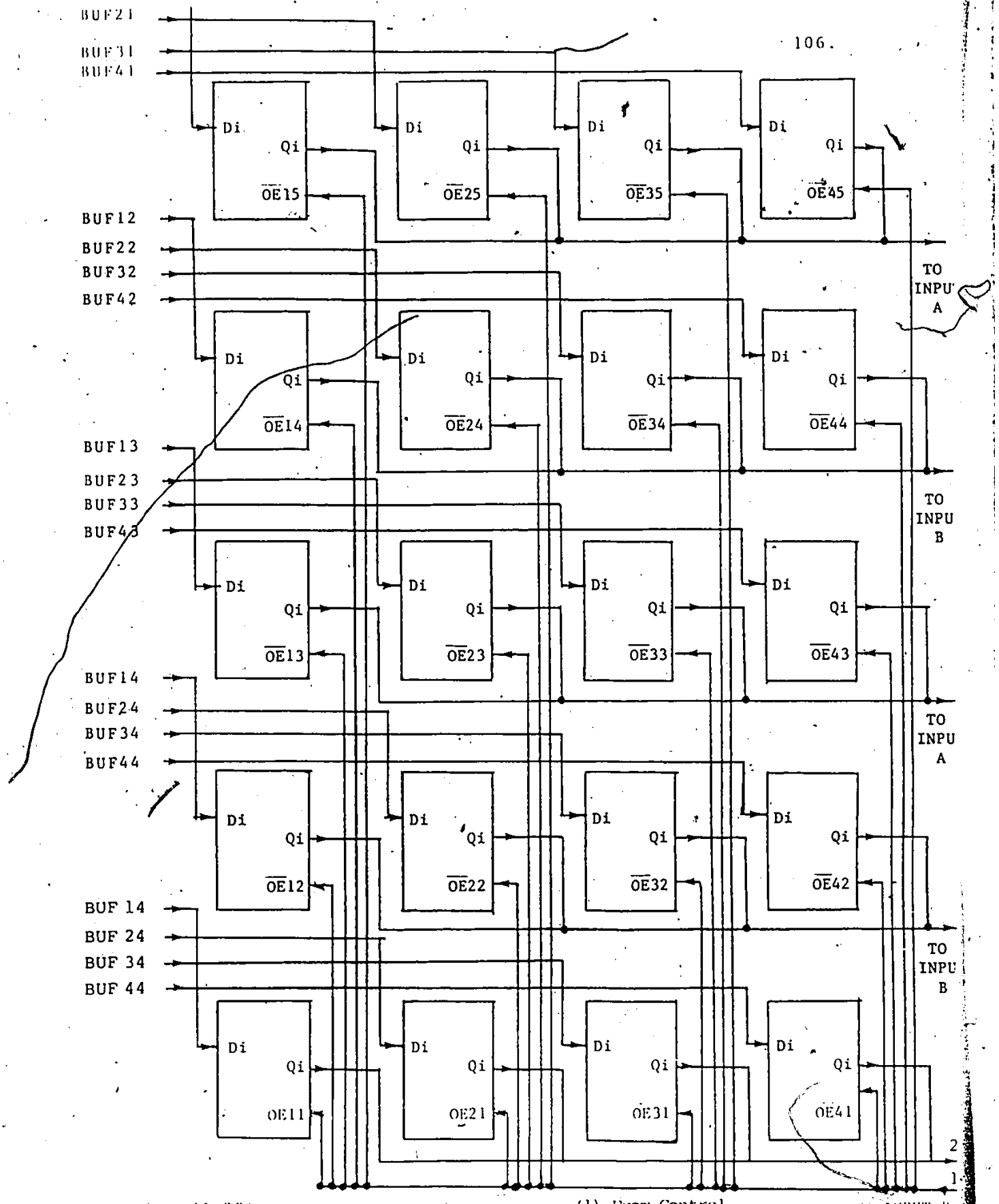
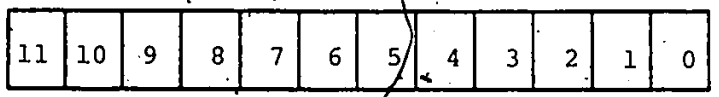


Fig. (4.22) BCU Multiplexing Unit (1) From Control Decoder (2) TO INPUT B



CBM0-3

Provides the bit pattern to select SR's for the different NTT stages

CBM4-7

Provides information for buffer selection

CBM8-10

Indicates the NTT stage

CBM 11

For DNT/INTT

Fig. (4.23) Control word for BCU

4.6.1 Control Unit for the BCU

The twelve-bit control word represents the required bit pattern which can be used to generate the necessary control signals for the BCU as well as for the BCU input multiplexing unit. Fig.(4.23) shows the details of the control word. Fig.(4.24) shows the block diagram of the control decoder. It consists of a group of NAND gates. The 7400's two input NAND gates were used to implement the control decoder.

The control bits, CBM0 to CBM3 provide the information for selection of the shift registers. The control bits, CBM4 to CBM7 provide the information about the memory buffer which contains the input data to the BCU. These eight bits are fed to the control decoder and twenty control signals are obtained from them. These signals are connected to the output enable inputs, '$\overline{OE}$'s, of the 74S244. If control bits, CBM4 to CBM7 are 0010, it would indicate that the BUF2 contains the input to the BCU. Each of the four bits corresponds to each memory buffer. When CBM0 to CBM3 are 0101, the BUF21 and BUF23 are selected, (i.e., the input in the first stage of NTT) when they are 0011 or 1000, the BUF21 and BUF22 or BUF23 and BUF24 are selected, forming the input during subsequent NTT stages. Fig.(4.24) shows the block diagram of the decoder. Table (4.4) contains the functional details of the control word.

The control bits CBM8 to CBM10 give information regarding the stage of the NTT, such as: 000 and 011. This will

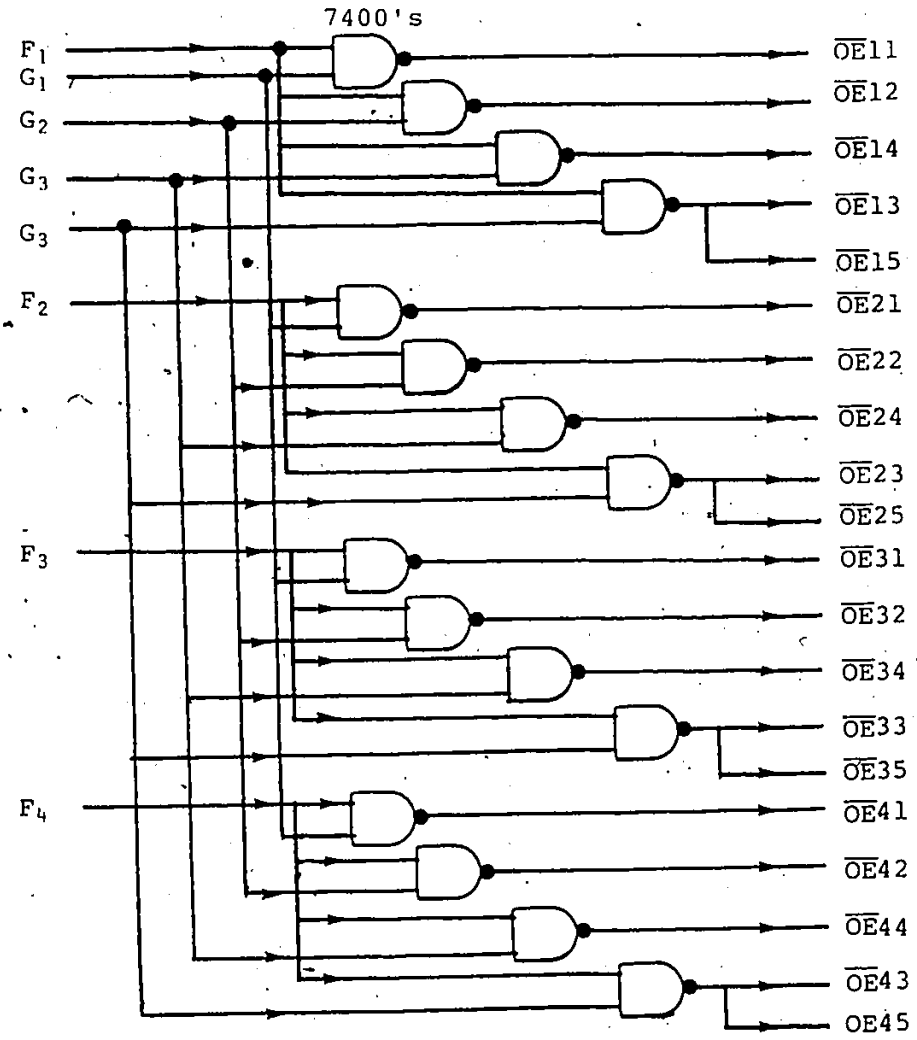


Fig. (4.24) The Control Decoder
F1-F4 = CBM4-CBM7,
G1-G4 = CBMØ-CBM3

F_4	F_3	F_2	F_1	Functional Details
0	0	0	1	Selects the input from BUF1
0	0	1	0	Selects the input from BUF2
0	1	0	0	Selects the input from BUF3
1	0	0	0	Selects the input from BUF4
G_4	G_3	G_2	G_1	
0	1	0	1	For 1st NTT stage
0	0	1	1	For 2nd N^{th} NTT stage. Input from SM1
1	0	0	0	For 2nd N^{th} NTT stage. Input from SM2

Table 4.4 Functional Details of BCU Control Word

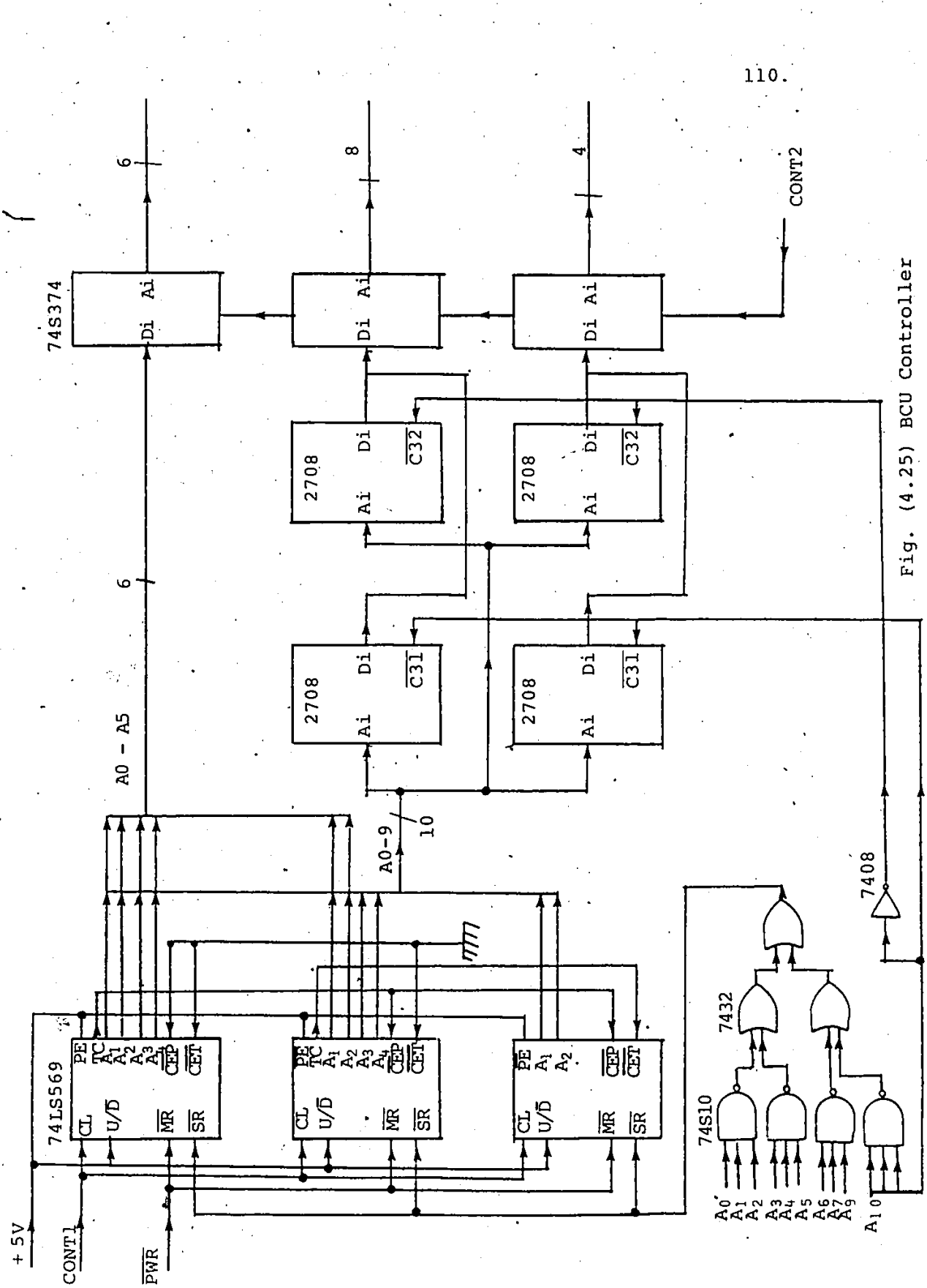


Fig. (4.25) BCU Controller

be decoded as the first and the fourth NTT stage. The most significant bit of control word, CBM11, when high, indicates INTT and when low, indicated DNTT.

These controls are stored in the two 2708 IC's. The address to these EPROM's is taken from the modulo 1791 counter. This modulo counter is constructed using three 74LS569's, two 74S10's and a 7432 as shown in Fig.(2.25). The controls are stored for two cycles of the processor. The six least significant bits of the counter are used to give the position of the butterfly as shown in Fig.(4.25).

4.7 THE MASTER CLOCK CIRCUITRY

Fig.(4.26) shows the block diagram of the master clock circuitry of an NTT processor. The main clock of 4 MHz was used. With this clock we find the processor's throughput rate to be 0.15 MHz.

The main clock was divided by two by using a 4-bit counter, 74LS569 and the clock for the memory buffer interconnection controller was generated. It was also used as the master clock CLKBM for the butterfly clock circuitry. The CLKBM was fed to 74LS569 and A1 to A4 outputs of the counter were connected to four inputs of the 7442 decoder. The alternate outputs of the decoder were inverted using Hex open collector inverters, 7406, and were used as the clocks to the different BCU stages as shown in Fig.(4.27). For example; the $\bar{0}$ is used as a clock to the fifth stage, $\bar{2}$ for 4th stage and so on.

The main clock was divided by four to generate shift-in,

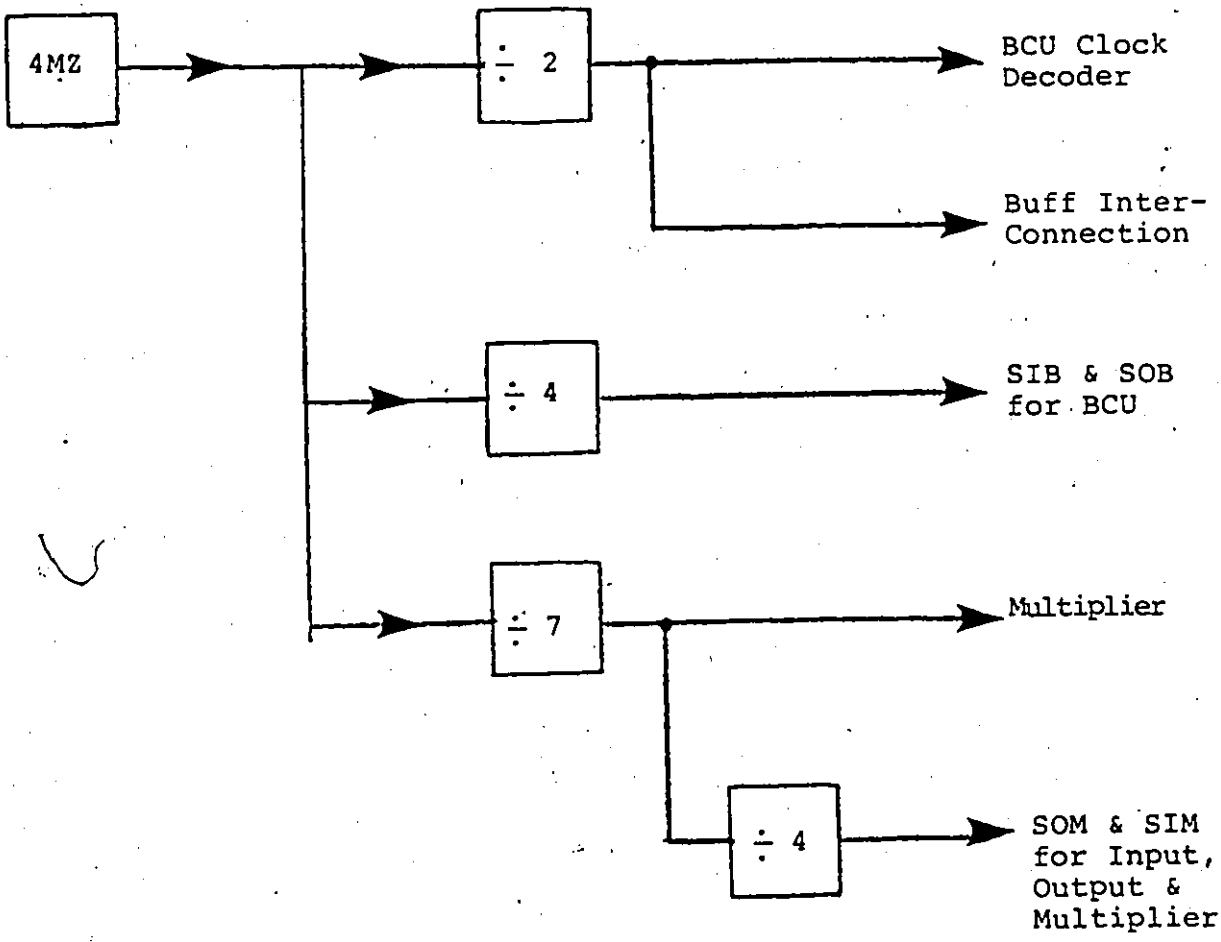


Fig. (4.26) Block Diagram of the Processor Clocks

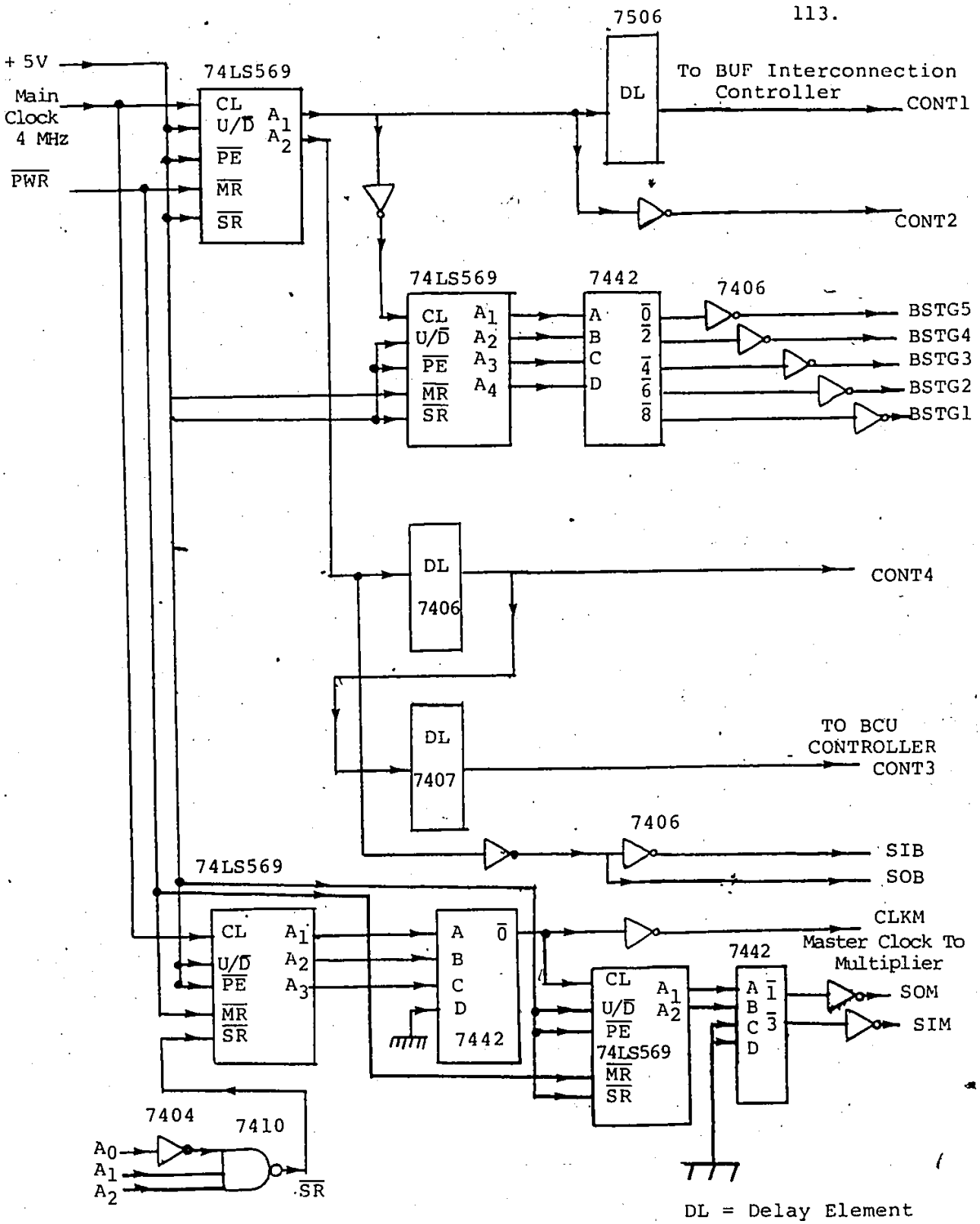


Fig. (4.27) Master Clock Circuitry

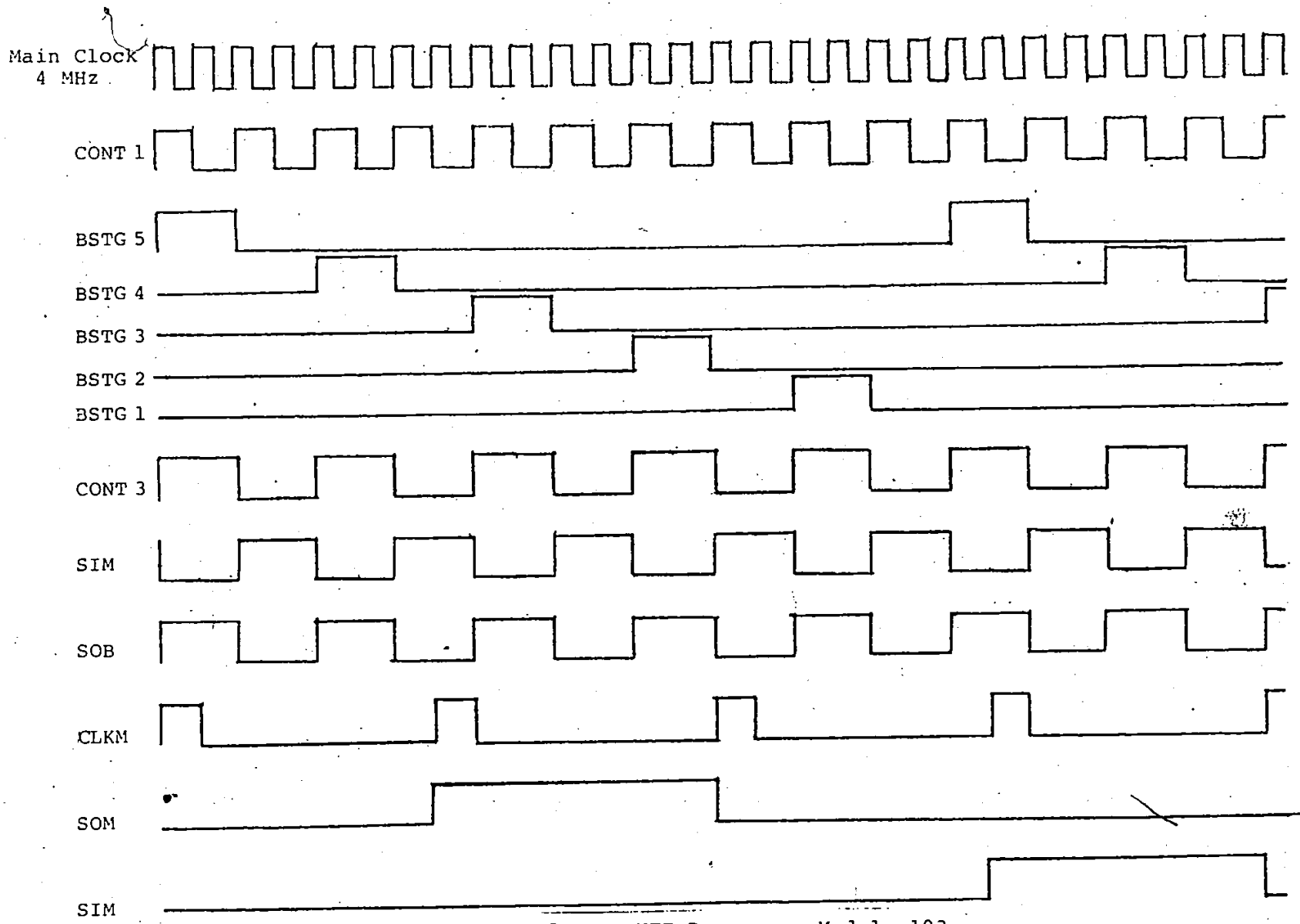


Fig. (4.28) Clock Pulses for an NTT Processor Module 193

SIB, and shift-out, SOB, clocks for a data transfer between the memory and the BCU. It was also fed to the modulo 1791 counter of BCU controller. Considering the worst case propogation delays of the various multiplexing circuitry involved the appropriate delays were introduced in SIB and SOB.

Furthermore, the master clock was divided by seven using modulo 7 counter and was used as a multiplier master clock, CLKM. This CLKM was fed to 74LS569, the A1 and A2 outputs of the counter were connected to the A and B input of the decoder 7442. The C and D inputs of the decoder were grounded. The outputs $\bar{1}$ and $\bar{3}$ were inverted and used as shift-out and shift-in pulses, SOM and SIM respectively, for input/output operations and also for a data transfer between the multiplier and the memory. Fig.(4.28) shows various clocks used for an NTT processor module 193.

4.8 GENERATION OF THE LOOK-UP TABLES

For storing the look-up tables in 2708, a universal PROM programmer by Intel was used and the tables were generated using assembly language for 8085 on the Intel 220 development system. The Intel 220 system is a microprocessor-based system which uses an 8085, 8-bit, micro-processor chip as the central processing unit.

The programs written in assembly language for generation of look-up tables for the multiplier and for controllers can be found in Appendix B. The subroutines were written for multiplication, multi-byte addition and subtraction. The separate programs were written to generate index, inverse

index, addition, modified addition and reconstruction tables. The results were generated at modulo 30 and 31. For example, ADD1 and ADD2 represent the addition look-up table at modulus 30 and 31 respectively. The five least significant address lines of the addition table are for the addend. The remaining address lines are for the adder. The inverse index and reconstruction tables are generated in such a way that the data at modulo 30 will be applied to the five least significant address lines. The data at modulo 31 will be applied to the five most significant address lines.

To detect the multiplication by zero in the index, look-up table FFH is stored as an index of zero. In the addition table, FFH is stored at location FFH. The inverse table stores, 00H at the location corresponding to FFH.

The programs which generate the control look-up tables do not require many calculations. The specific bit pattern is stored at the pre-determined address. The bit patterns for different controllers were pre-determined and loaded into the memory. The programs which generate the control look-up tables consist mainly of replacing one set of parameters for a bit pattern with another in repeated loops. These programs were developed using MACROS. The MACROS were used to repeat numerous instruction sequences with only certain parameters changed. The detail listing of these programs can be found in Appendix B.

4.9. SUMMARY

In the previous chapter we discussed the modular design of an NTT processor. In this chapter the hardware implementation of the external serial complex multiplier, memory buffers and various multiplexing units were described in detail. The generation of control signals using look-up tables as a method to replace sequential/combinational logic is discussed. In addition, the processor module 193 was constructed and a complete design approach was presented.

CHAPTER 5

TESTING AND HARDWARE REQUIREMENT OF AN NTT PROCESSOR

5.1. INTRODUCTION

An NTT processor module 193 described in the previous chapters has been constructed using inexpensive off the shelf components. The main elements of the processor module 193 are

(i) four memory buffers,

(ii) a complex multiplier,

(iii) a BCU.

The complex multiplier and the BCU perform the arithmetic operations required for the convolution and the input and output of the convolution are stored in memory buffers. The testing of these elements has been done in two phases. In phase one, the operation of the multiplier and the BCU was tested. In phase two the processor was tested by convolving a number of sequences with the impulse response of a lowpass filter. In this chapter, the test procedure used for the verification of the NTT processor and the results obtained are presented. The total hardware requirement for the NTT processor using three moduli is also discussed.

5.2. TESTING OF THE NTT PROCESSOR MODULE 193

(a) Phase One

In this test procedure, the operation of a multiplier and the BCU was tested. Fig.(5.1) shows the block diagram

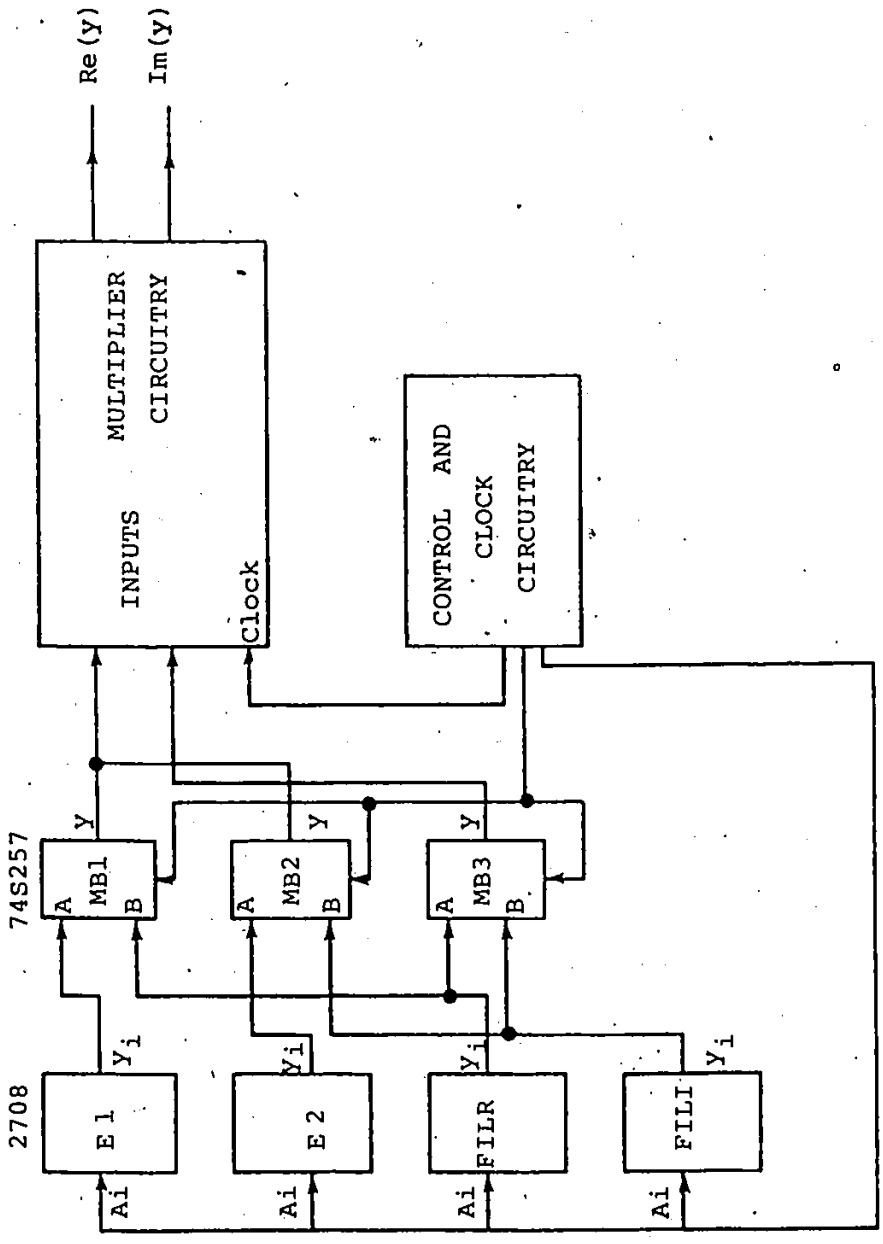


Fig.(5.1) Block Diagram of a Circuitry to Test a Multiplier

RX	IX	RH	IH	Re(X·H)	Im(X·H)
1	1	1	1	126	2
7	7	7	7	191	98
21	21	21	21	175	110
47	47	47	47	28	172
69	69	69	69	42	65
82	82	82	82	165	131
101	101	101	101	139	137
123	123	123	123	186	150

TABLE (5.1) Inputs and Outputs of the Multiplier

STAGE	POSITION	INV/$\overline{FWD}$	α	α^{-1}	OUTPUT C	OUTPUT D
0	46	0	20,0	--	125,110	95,30
1	6	0	158,0	--	"	74,44
2	11	0	64,0	--	"	111,96
3	16	0	43,0	--	"	156,161
4	30	1	--	9,0	"	91,110
5	4	1	--	1,0	"	53,98
6	55	1	--	1,0	"	53,98

TABLE (5.2) Some of the Details of the Testing of the BCU With INPUT A = 89,104 and INPUT B = 36,6

for testing the operation of the multiplier. The multiplier was tested by using a known input sequence and then checking the output of the multiplier. The input sequence of 128 points consisted of the numbers from 0 to 127. This sequence was stored in four EPROMs, E1, E2, FILR and FILI as shown in Fig.(5.1). The outputs of the EPROMs, FILR and FILI, were connected to the A and B inputs of the multiplexer MB3. The A input of the multiplexers MB1 and MB2 were taken from the outputs of EPROMs E1 and E2 and the B inputs of these multiplexers were taken from FILR and FILI as shown in Fig.(5.1). The main clock of the multiplier, CLKM, was single stepped and the outputs of the multiplier were checked. Some of the outputs obtained are tabulated in Table (5.1). The output of the multiplier observed during testing was the same as that obtained from the simulation of the multiplier.

For testing of the BCU, two fixed inputs (89, 104) and (36, 6) were connected to the INPUT A and B of the BCU. The main clock of the BCU was single stepped and the position of the butterfly within this stage was incremented and the outputs of the BCU at the point C and D were checked. Some of the outputs observed during testing are tabulated in Table (5.2).

(b) Phase Two

In phase two of the test procedure, the processor module 193 was tested by convolving a sine, a triangular and a ramp input with an impulse. It was also tested to perform a filtering operation. The main clock of the

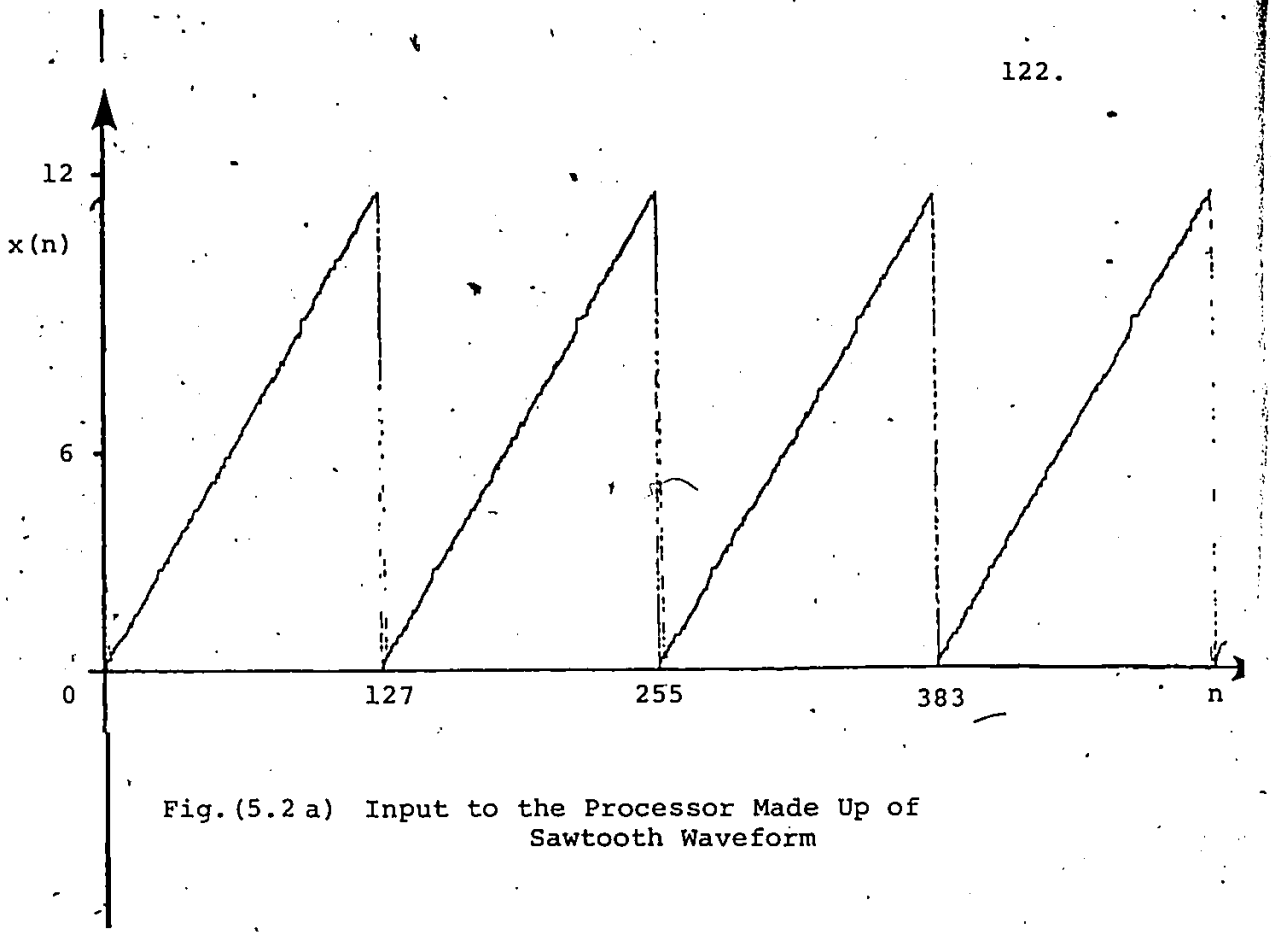


Fig. (5.2 a) Input to the Processor Made Up of Sawtooth Waveform

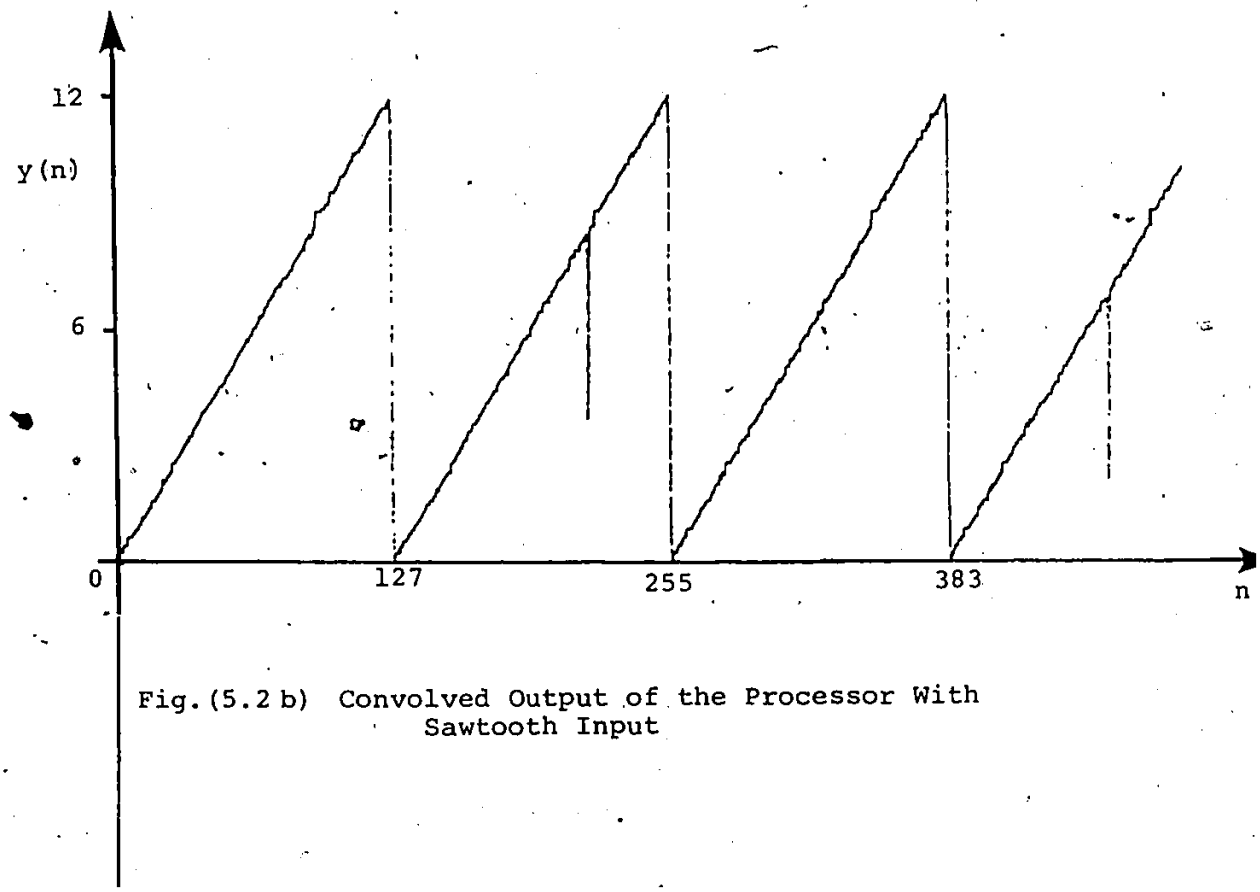


Fig. (5.2 b) Convolved Output of the Processor With Sawtooth Input

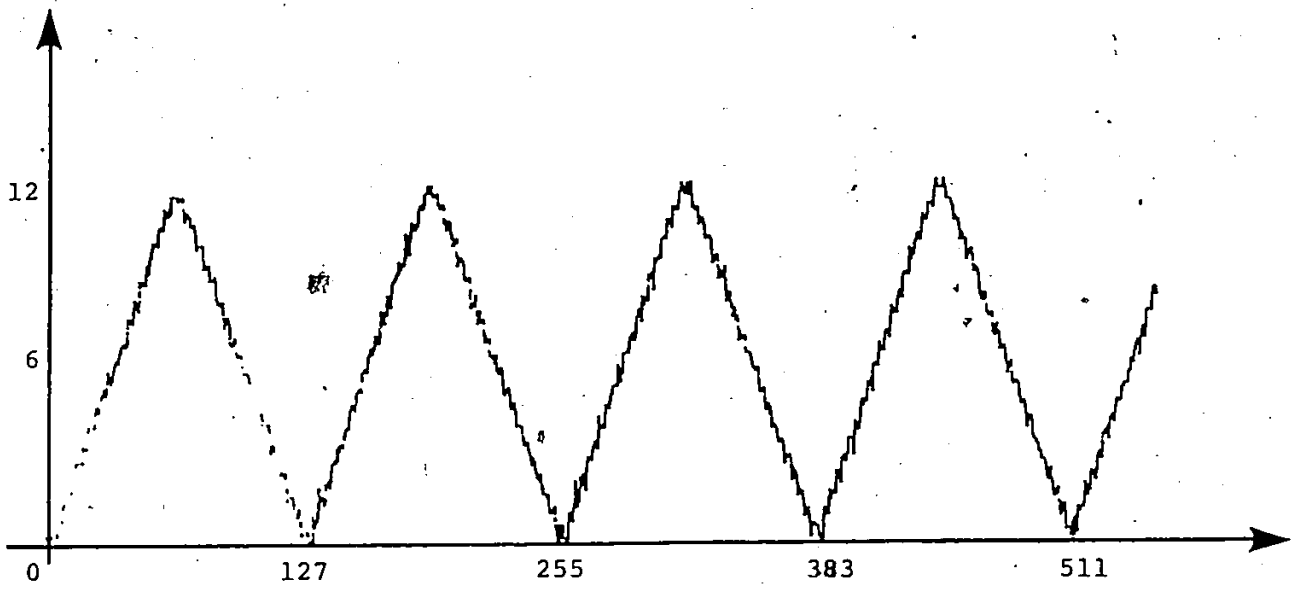


Fig. (5.3 a) Triangular Wave Input to the Processor

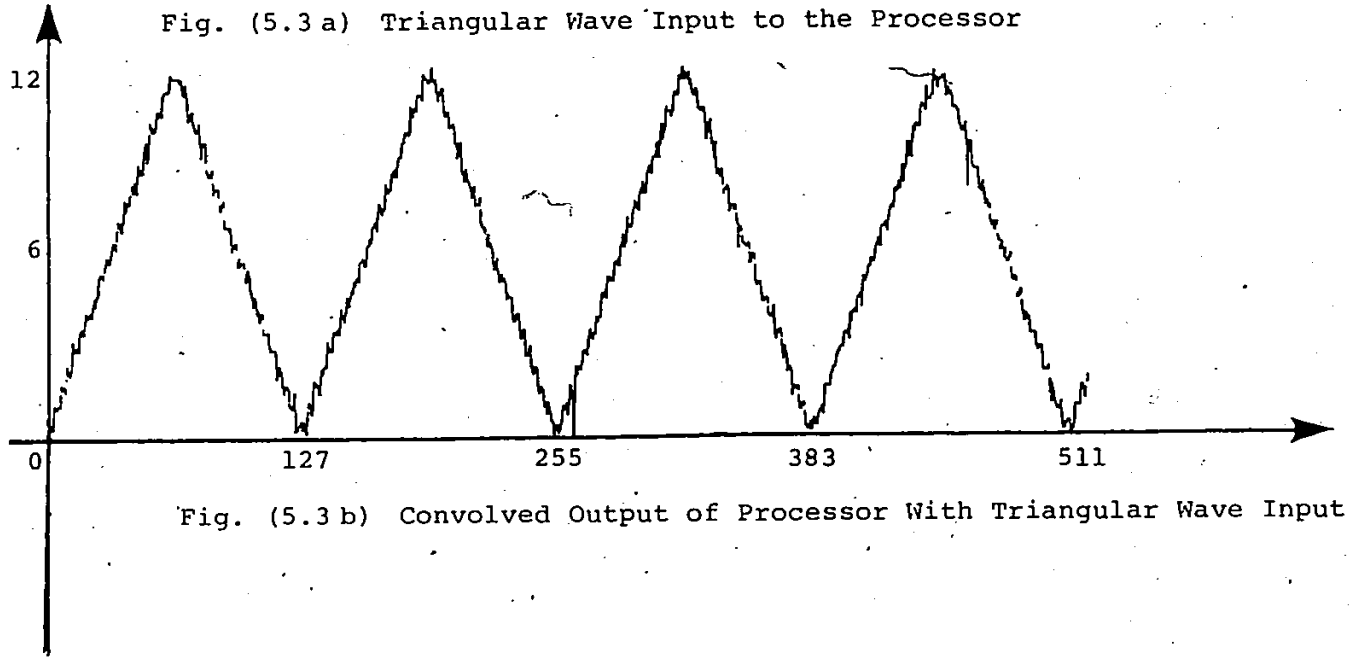


Fig. (5.3 b) Convolved Output of Processor With Triangular Wave Input

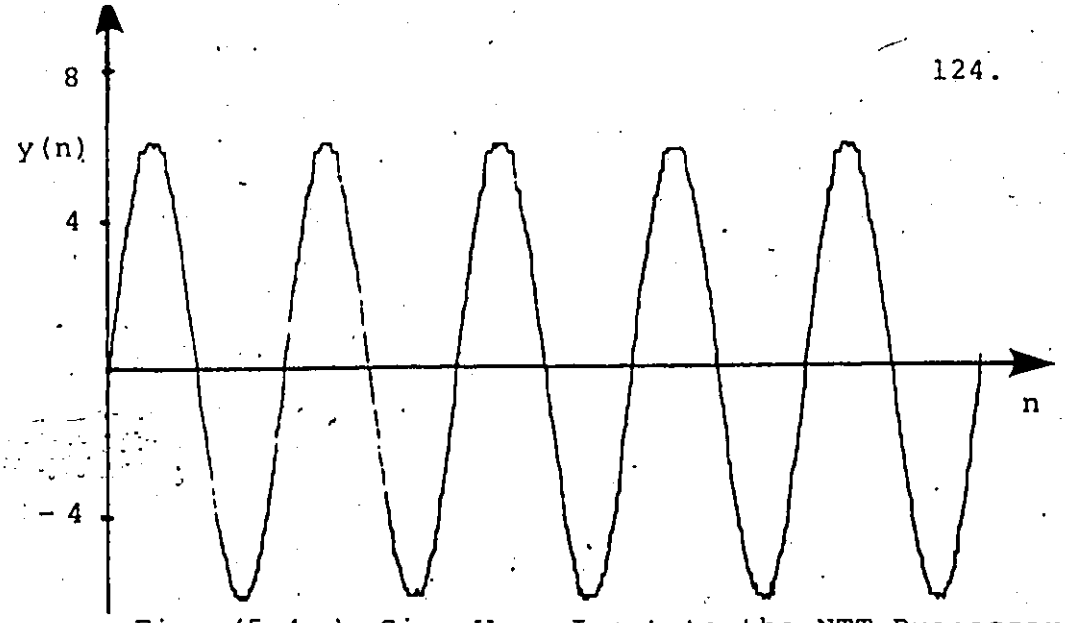


Fig. (5.4 a) Sine Wave Input to the NTT Processor

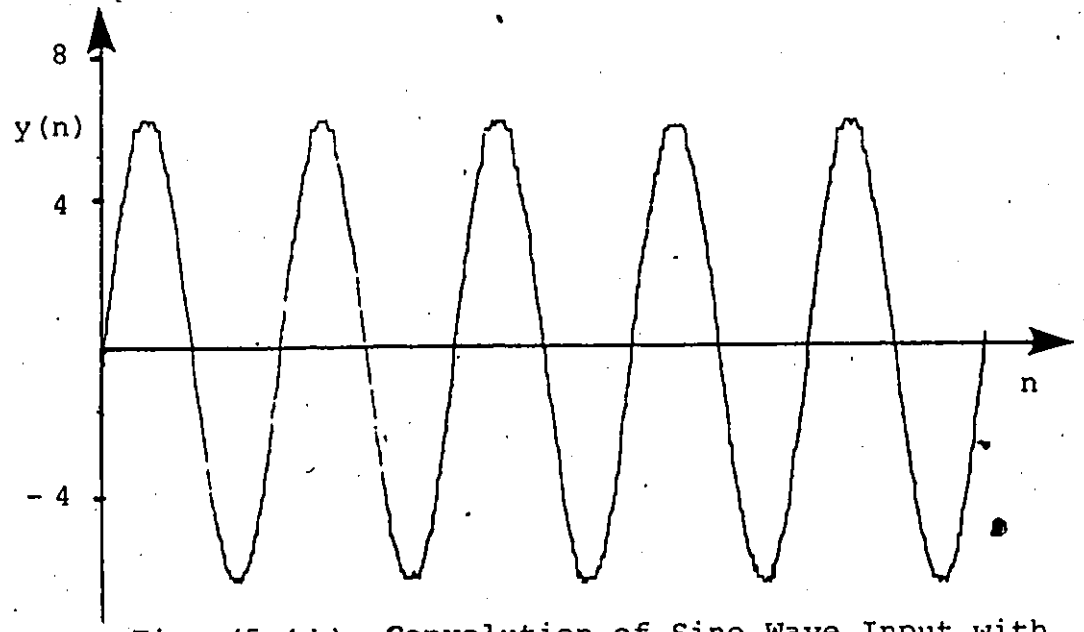


Fig. (5.4 b) Convolution of Sine Wave Input with An Impulse

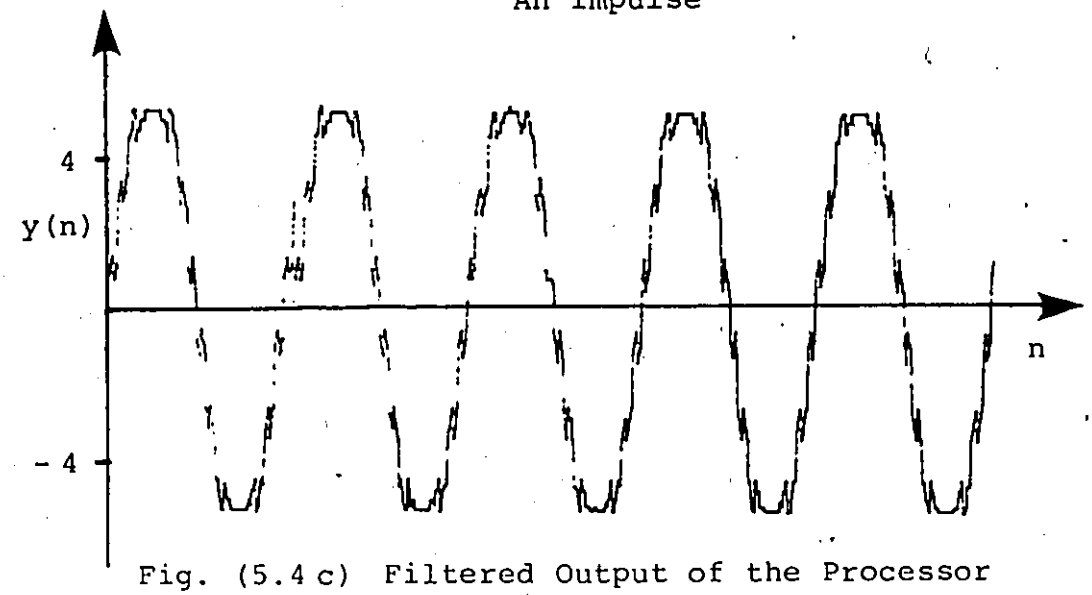


Fig. (5.4 c) Filtered Output of the Processor

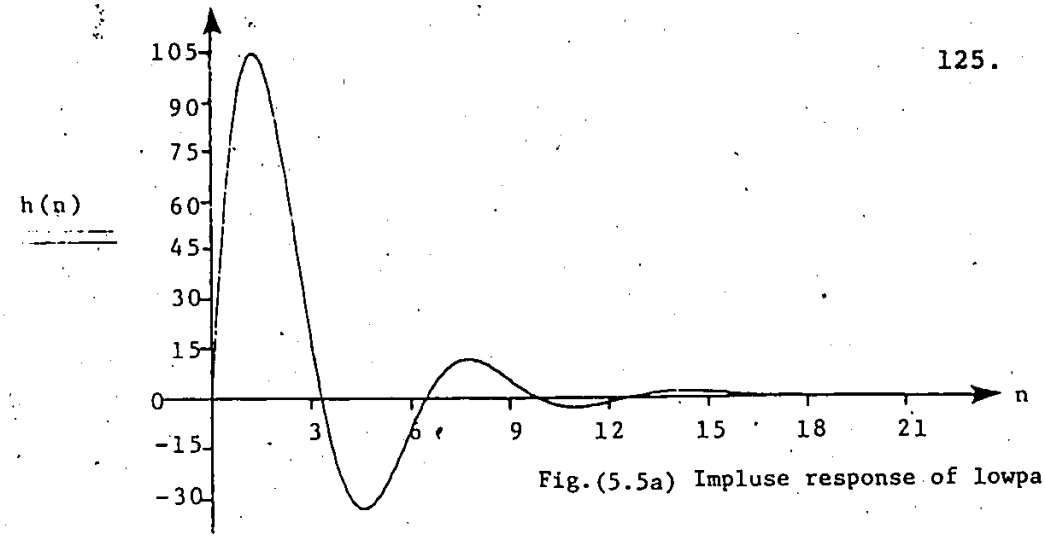


Fig.(5.5a) Impluse response of lowpass filter.

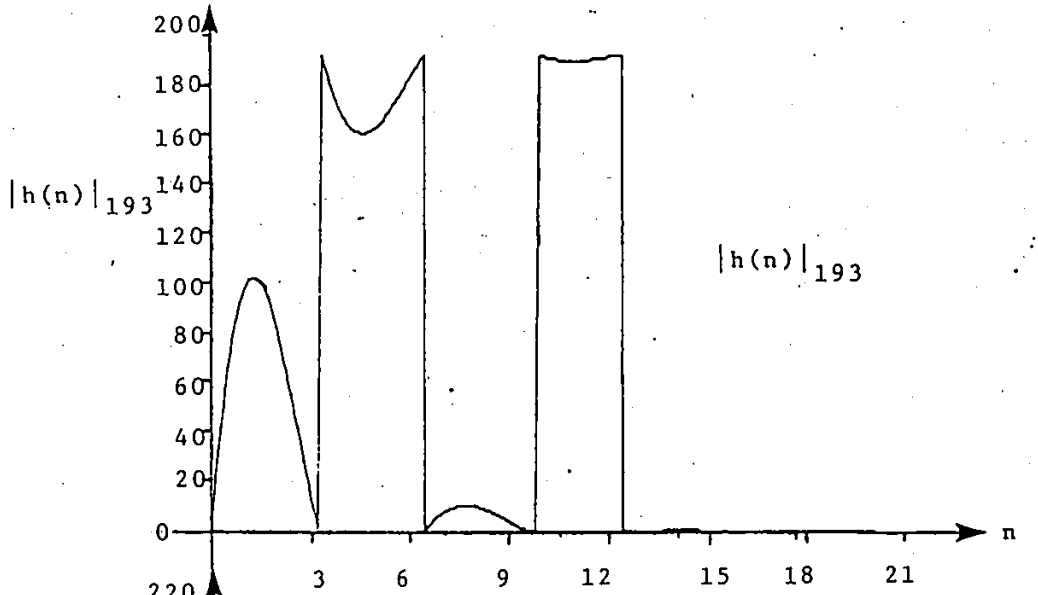


Fig.(5.5b) Impulse response of lowpass filter at modulo 193.

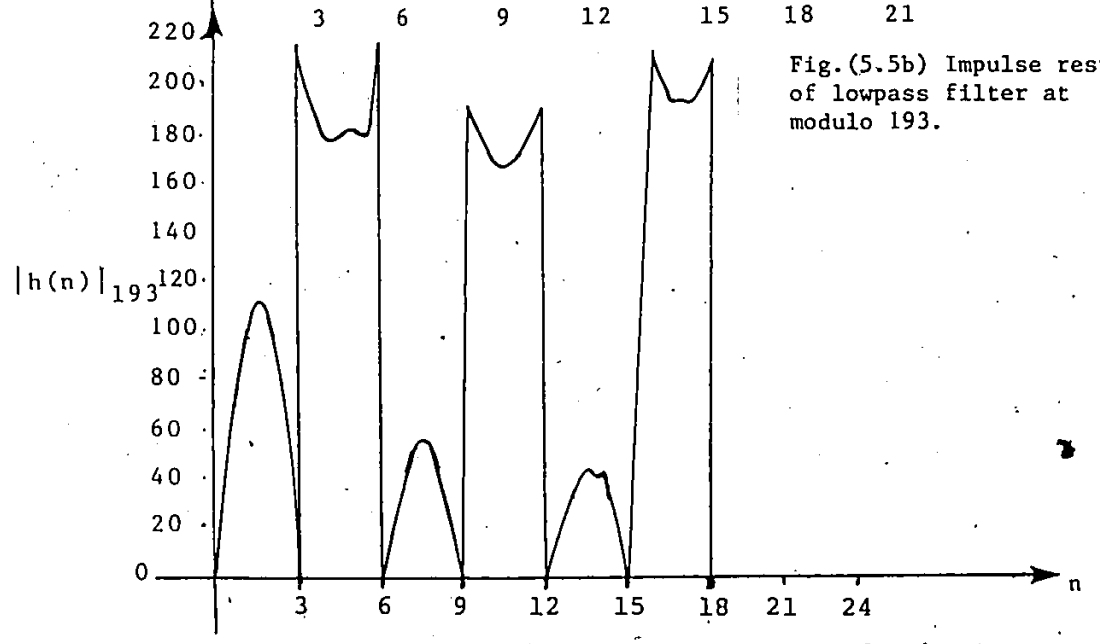


Fig. (5.5c) Processor output with impulse input.

processor was taken from the function generator and the clock frequency was varied between 1 and 2 kHz. The input waveforms were premultiplied by the scale factor, 12, and were stored in EPROMs. The filter coefficients were stored in EPROMs FILR and FILI.

The sawtooth, triangular and sine wave inputs shown in Fig.(5.2 a), Fig.(5.3 a) and Fig.(5.4 a), respectively, were stored in EPROMs. Fig.(5.2 b), Fig.(5.3 b) and Fig.(5.4 b) show the outputs of the convolver.

The frequency response of the lowpass filter with the cut-off frequency of 2.6 kHz was stored in the EPROMs FILR and FILI. Fig.(5.4 c) shows the output of the processor observed on the scope when a sine wave input was convolved with the impulse response of the lowpass filter. Fig.(5.5 b) shows the output of the processor when an impulse was convolved with impulse response of the lowpass filter. Fig. (5.5 a) shows the impulse response of the lowpass filter observed during simulation of the processor. The results obtained during testing of the processor match with that of simulation results.

5.3. GENERAL COMMENTS

The type of components (protoboards, interconnecting wires, IC's) used to construct the processor module 193, imposed some constraints on the operating speed of the processor.

The experimental set-up was built on the protoboards, which were mounted on a wooden board. Each protoboard can

hold either three 24 pin ICs or four 20 pin ICs. Hence the circuitry of the processor module was spread over the area of 2m x 1m. MW-C-22 solid hook-up wire was used to interconnect the ICs and the different units (BCU, multiplier, distributor unit, output multiplier units and control units). The capacitance and inductance of these wires introduced a lot of noise and impose severe limitation on the operating speed of the processor. In order to operate the processor at higher frequencies and to eliminate noise problems, the use of Schottky TTL wirewrap boards to build prototype of the processor is recommended.

As discussed in previous chapter, 74S374 edge trigger latches whose settling time is 20 nsec, were used to latch the output of EPROMs used to build the multiplier. As the multiplier operates in a pipeline, the relative delay of 25 nsec was introduced in the clock of each stage of the multiplier. As discussed in the previous chapter, the delays in the different clocks were obtained by passing the clocks through a number of buffers or inverters. The propagation delay of buffer or inverter for low to high transition is a maximum of 15 nsec. To generate the delay of 25 nsec, the clock has to pass through three buffers. The delays generated by this technique are not precise and also the number of ICs required are large. To obtain an accurate delay and to have a lower chip count, commercially available digital delay lines may be used.

In order to interface TTL ICs to CMOS FIFO's, the pull-

up resistors of 10k were connected between the outputs of TTL gate and $+V_{CC}$ as discussed in the previous chapter. TTL to CMOS level converter IC or open collector TTL buffer must be used. The output drive characteristics of the buffer IC are equal or better than that of a typical totem-pole TTL gate; therefore, there are no problems as far as current-drive capability is concerned.

2708, EPROMs were used to store various look-up tables for the BCU and the multiplier and Am2812A, FIFO's were used to build a memory buffer. With these ICs the throughput rate of the processor is 0.15 MHz. In order to achieve a higher throughput rate of 3.5 MHz, the high speed PROMs with register (like TBP28R165 by TI) and high speed FIFO's (like 75LS222/74LS227 by TI) must be used. The use of PROMs with register will result in smaller chip count for the processor.

Out of the three modules required for a complete NTT processor, only module 193 was built. The other two modules, module 191 and module 449, have similar structures and all the three modules can share the same control unit and clocking circuitry.

2708, EPROM can store 1024 words with the wordlength of 8 bits each. While storing some of the look-up tables, the complete wordlength was not utilized. We stored the look-up tables for arithmetic operation at sub-moduli 30 and 31 in separate EPROMs, which is discussed in the previous chapters. We have three unused bits in each of these EPROMs. These

T I T L E	EPROMs 2708	FIFO Am2812A/ Am2813A	LATCHES 74S374	LATCHES 74S174 and 74S175	OCTAL BUFFER 74S244	BUFFERS 74125 and 74365	MULTIPLEXERS 74S257	COUNTERS 74LS569	TTL GATES
Serial Complex Multiplier	16	--	26	--	--	--	6	5	23
Memory Organization	16	32	4	--	--	--	24	3	6
Distributor	--	--	--	--	8	--	16	--	--
Output Multiplexing Unit	--	--	--	--	--	--	4	--	--
BCU Input Multiplexing Unit	4	--	3	--	40	--	--	3	10
BRC, RBC and Clock Circuitry	12	--	4	--	--	--	--	4	25
⑤ Total Hardware Requirement for Module 193	80	32	69	--	48	--	50	15	64
⑥ Total Hardware Requirement for Module 191	80	32	69	--	48	--	50	--	--
⑥ Total Hardware Requirement for Module 449	102	32	63	6 + 6	--	48 + 48	55	--	--

⑤ Figures includes IC Count for BCU

TABLE (5.3) Hardware Requirements of the NTT Processor for Three Moduli Schemes

unused bits can be used for parity check.

5.4. The Hardware Requirements of an NTT Processor

Table (5.3) gives the hardware requirement of the processor module 193. The module 191 will require approximately the same number of ICs. The module 449 can be constructed using Am2813A FIFOs, 74125 and 74365 buffers and 74S174 and 74S175 latches. Table (5.3) also gives the hardware requirement for the module 191 and module 449.

The processor requires four power supplies (+ 12 volts, - 12 volts, + 5 volts and - 5 volts). The current ratings for each of these power supplies for the module 193 is as follows:

+ 5 volts, 19A

- 5 volts, 1A

+ 12 volts, 3.5A

- 12 volts, 0.7A.

5.4.1. Measuring Instrument Used

The various measuring instruments were used during the testing and debugging of the hardware units. The main clock was taken from the function generator, MODEL 743. DOLCH logic analyzer, LAM3250, was used to observe the outputs of the intermediate stages of the processor and the relative delay between the various clocks. 5015T logic trouble shooting kit by HP was used to observe the output during the testing of the multiplier and BCU. The input and output waveforms during the testing of the processor were observed on the digital storage oscilloscope, OS4000, and the output

waveforms were plotted using HP 7045A X-Y recorder.

5.5. SUMMARY

The hardware set-up built for the processor module 193 was tested in two phases. In the first phase of testing, known inputs were applied to the multiplier and the BCU and their outputs were verified by single stepping. In the second phase of testing, the processor was tested to perform a filtering operation. The results obtained in both the phases match with the simulation results. The hardware requirement of the processor for the moduli, 191, 193 and 449 has also been presented.

CHAPTER 6

CONCLUSION

The design of the memory buffers, an external serial complex multiplier and various multiplexing units for a real time sequential NTT processor has been proposed. The processor performs the convolution of an input sequence with the impulse response of a filter by computing the NTT of input sequence, multiplying it by the NTT of the filter response and then taking inverse NTT.

An external complex multiplier has been realized using EPROMs and Registers which results in an extremely simple pipeline structure. The multiplication of the two sequence can also be performed in the final stage of NTT by using an additional complex multiplier in the BCU. This technique of multiplying the two NTTs requires more hardware and the complexity of the hardware design is much greater than the design proposed in this thesis.

The use of FIFO's than the Random Access Memories (RAMs) or static shift registers in the design of the memory buffers results in an efficient simple memory architecture. The use of RAMs to implement memory buffers requires more hardware for the generation of the memory addresses. The use of currently available low density, slow speed, static registers increases the chip count as well as higher throughput rate cannot be achieved.

The distributor unit, the output multiplexing unit and the BCU input multiplexing unit have been implemented using multiplexers and octal buffers. Instead of using sequential/

combinational logic to generate the control signals for various multiplexing units the look-up table technique has been employed. The use of look-up tables to generate the controls results in simpler hardware structure for the implementation of the control unit.

The prototype of the processor for module 193 was built and tested by convolving a number of input sequences with the impluse response of a lowpass filter. The result obtained during this testing of the prototype match with the software simulation results.

With the recent development in VLSI technology and growing need of industries for special purpose processors for digital signal processing applications, future work can be concentrated in developing special purpose chips for the proposed NTT processor.

REFERENCES

[1] N.S. Szabo and R.I. Tanaka, "Residue Arithmetic and Its Applications to Computer Technology", McGraw-Hill Inc., New York, 1967.

[2] G.A. Jullien, "Implementation of Multiplication Modulo A Prime Number with Application to Number Theoretic Transform", IEEE, Trans. on Computers, Vol.C-29, No.10, October 1980, pp.899-905.

[3] A.Z. Baraniecka, "Digital Filtering Using Number Theoretic Transform to Implement Fast Digital Convolution", Ph.D. Dissertation, Electrical Engineering, University of Windsor, 1980.

[4] A. Peled and B. Liu, "A New Hardware Realization of Digital Filters", IEEE Trans. Acoust., Speech, Signal Processing, Vol.ASSP-22, December 1974, pp.456-462.

[5] L.R. Rabiner and B. Gold, "Theory and Application of Digital Signal Processing", Prentice-Hall, Inc., Englewood Cliffs, N.J., USA.

[6] P.J. Nicholson, "Algebraic Theory of Finite Fourier Transforms", J.Comput., Syst.Sci., Vol.5, 1971, pp.524-547.

[7] J.M. Pollard, "The Fast Fourier Transform in a Finite Field", Math.Comp., V.25, April 1971, pp.365-394.

[8] R.C. Agarwal and C.S. Buerus, "Number Theoretic Transform to Implement Fast Digital Convolution", Proc.IEEE, Vol.63, April 1975.

[9] M.C. Pease, "An Adaption of the Fast Fourier Transform for Parallel Processing", J.Assoc.Comput.Mach., Vol.15, April 1968.

[10] M.J. Corinthios, "A Time-series Analyzer", MRI Symposia Ser. New York: Polytechnique Press, Vol.19, 1969, pp.47-61.

[11] M.J. Corinthios, "A Fast Fourier Transform for High Speed Signal Processing", IEEE Trans. on Computers, Vol.C-20, No.8, August 1971, pp.843-846.

[12] M.J. Corinthios, "The Design of a Class of Fast Fourier Transform Computers". IEEE Trans. on Computers, Vol. C-20, No.6, June 1971, pp.617-623.

[13] M.J. Corinthios, et al, "A Parallel Radix-4 Fast Fourier Transform Computer",IEEE Trans. on Computers, C-24, Jan.1975, pp.80-92

[14] H.K. Nagpal, "Processor Architectures for Fast Computation of Multi-dimensional Unitary Transforms", Ph.D. Dissertation, Electrical Engineering, University of Windsor, 1981..

[15] M. Akhtar, "A Read-Only-Memory Oriented Implementation of the Number Theoretic Transform Butterfly Unit", MASc Thesis, Electrical Engineering, University of Windsor, 1981.

[16] 'The Bipolar Microcomputer Components Data Book for Design Engineers', Texas Instruments Inc., 1981.

[17] 'MOS/LSI Data Book', Advanced Micro Devices, Inc., 1980.

[18] 'MOS Memory Data Book', Fairchild, A Schumberger Company, 1981.

APPENDIX A

SIMULATION PROGRAMS

The simulation of the processor modules 191, 193 and 449 was done on an IBM 370/3031 Computer. Listings of the programs are given here.

```
C     ************************************************************
C     *                                                          *
C     *   MAIN LINE PROGRAM TO SIMULATE AN NTT PROCESSOR         *
C     *  MODULE 193.                                             *
C     *                                                          *
C     ************************************************************
      IMPLICIT INTEGER (A-H,O-Z)
      EXTERNAL NTT
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      COMMON/BLK2/ IND1(193),IND2(193),INV1(31,31),INV2(31,31)
      COMMON/BLK3/ ADD1(31,31),ADD2(31,31),SAD1(30,30),SAD2(31,31)
     1,RECT(30,31)
      COMMON/BLK4/ MMOD,SUBM1,SUBM2,NP,MDOT1,MDOT2,IMDOT1,
     1IMDOT2,SOD,ALPSO,N
      COMMON/BLK5/ OX(1,2),IN(1,2),H(128,2),X(1,2)
      COMMON/BLK6/ STAG2,STAG3,STAG4,STAG5,STAG6,STAG7,STAG8,STAG9
     1,STAG10,STAG11,STAG12,STAG13,STAG14,STAG15,STAG16,STAG17,
     2STAG18,STAG19,STAG20,X1,X2,X3,X4
      DIMENSION BUF11(32,2),BUF12(32,2),BUF13(32,2),BUF14(32,2),
     1BUF21(32,2),BUF22(32,2),BUF23(32,2),BUF24(32,2),AOUT(64),
     2BUF31(32,2),BUF32(32,2),BUF33(32,2),BUF34(32,2),
     3BUF41(32,2),BUF42(32,2),BUF43(32,2),BUF44(32,2)
     4,OUT(128,2),IIN(128,2),ITS(64),IKN(128)
C
C
C     POWER-ON RESET OF LATCHES AND BUFFERS.
C
C
      STAG2=STAG3=STAG4=STAG5=STAG6=STAG7=STAG8=STAG9=STAG10=
     1STAG11=STAG12=STAG13=STAG14=STAG15=STAG16=STAG17=STAG18=
     2STAG19=STAG20=X1=X2=X3=X4=0
      DO 2 I=1,32
      ITS(I)=ITS(I+32)=AOUT(I)=AOUT(I+32)=0
      DO 2 J=1,2
      BUF11(I,J)=BUF12(I,J)=BUF13(I,J)=BUF14(I,J)=0
      BUF21(I,J)=BUF22(I,J)=BUF23(I,J)=BUF24(I,J)=0
      BUF31(I,J)=BUF32(I,J)=BUF33(I,J)=BUF34(I,J)=0
      BUF41(I,J)=BUF42(I,J)=BUF43(I,J)=BUF44(I,J)=0
      IIN(I,J)=IIN(I+32,J+32)=IIN(I+64,J+64)=IIN(I+96,J+96)=0
    2 CONTINUE
C
C
C     DEFINING THE PARAMETERS FOR LOOK-UP TABLES CALCULATIONS.
C     N-TRNSFORM LENGTH.
C     MMOD=MAIN MODULUS .
C     ALPSO-SOURCE OF THE CYCLIC GROUP GENERATOR.
C     MIV-MULTIPLICATIVE INVERSE OF N.
C     NP=MULTIPLICATIVE INVERSE OF MMOD.
C     SUBM1 & SUBM2 = SUB MODULI.
C     IMDOT1 & IMDOT2 = MULTIPLICATIVE INVERSE OF SUBM1 & SUBM2.
C
C
      MMOD=193; SUBM1=30 ; SUBM2=31 ; SOD=SUBM1*SUBM2 ; NP=5
      IMDOT1=-1 ; IMDOT2=30 ;MIV=95; ALPSO=125; N=128
C
C
C     SUBROUTINE TWLF TO CALCULATE THE THE POWERS OF
C     CYCLIC GROUP GENERATOR ALPHA.
C
C
      CALL TWLF
C
C
C     SUBROUTINE TO CALCULATE THE VARIOUS LOOK-UP TABLES.
C
C
      CALL LTABLE
C
C
C     SUBROUTINE TO CALCULATE THE FILTER RESPOSE.
C
C
      CALL FILTER (H)
C
C
C     MULTILPY THE FILTER COEFFIENTS BY MULTIPLICATIVE INVERSE
C     OF TRANSFOM LENGTH N.
C
C
      DO 997 I=1,N
      DO 997 J=1,2
      H(I,J)=MINV*H(I,J)
  997 H(I,J)=H(I,J)-MMOD*(H(I,J)/MMOD)
C
C
C     SUBROUTINE TO GET INPUT SEQUENCE .
C
C
      CALL INFILE(IKN)
C
C
      DM4=1 ; DM2=0 ; IMUX1=2 ; DM3=1 ; IMUX22=1
      DO 100 K=1,8
C
C
C     SUBROUTINE TO ARRANGE THE INPUT SEQUENCE IN THE FORM
C     REQUIRED FOR OVER-LAP SAVE METHOD.
C
C
      CALL OVLAP(IIN,ITS,IKN)
      IF(DM4.NE.1.AND.DM2.EQ.1) GO TO 3
      CALL INOUT(BUF41,BUF42,BUF43,BUF44,IIN,OUT)
      OPMUX=2 ; IMUX22=1 ; IMUX24=0
      GO TO 4
    3 CALL INOUT(BUF21,BUF22,BUF23,BUF24,IIN,OUT)
      OPMUX=1 ; IMUX24=1 ; IMUX22=0
    4 IF(IMUX1.NE.2) GO TO 5
      MMI=65 ; MMF=128
      CALL MULTLY(BUF31,BUF32,BUF33,BUF34,MMI,MMF,BUF11,BUF12,
     1BUF13,BUF14)
```

```
      IMUX23=1 ; IMUX21=0
      GO TO 6
5     MM1=65 ; MMF=128
      CALL MULTY(BUF11,BUF12,BUF13,BUF14,MM1,MMF,BUF31,BUF32,
     1BUF33,BUF34)
      IMUX21=1 ; IMUX23=0
6     IF((IMUX22.NE.1.AND.IMUX24.EQ.1) GO TO 7
      STAG=1 ; INV=0 ; DM1=1
      CALL STAG1(BUF21,BUF22,BUF23,BUF24,BUF11,BUF12,BUF13,BUF14,
     1NTT,STAG,INV)
8     STAG=STAG+1 ; DM2=2
      CALL ASTAG(BUF11,BUF12,BUF13,BUF14,BUF21,BUF22,BUF23,BUF24,
     1NTT,STAG,INV)
      STAG=STAG+1 ; DM1=1
      CALL ASTAG(BUF21,BUF22,BUF23,BUF24,BUF11,BUF12,BUF13,BUF14,
     1NTT,STAG,INV)
      IF(STAG.NE.7) GO TO 8
      IMUX1=1 ; DM1=2
      GO TO 10
7     STAG=1 ; INV=0 ; DM3=2
      CALL STAG1(BUF41,BUF42,BUF43,BUF44,BUF31,BUF32,BUF33,BUF34,
     1NTT,STAG,INV)
9     STAG=STAG+1 ; DM4=2
      CALL ASTAG(BUF31,BUF32,BUF33,BUF34,BUF41,BUF42,BUF43,BUF44,
     1NTT,STAG,INV)
      STAG=STAG+1 ; DM3=2
      CALL ASTAG(BUF41,BUF42,BUF43,BUF44,BUF31,BUF32,BUF33,BUF34,
     1NTT,STAG,INV)
      IF(STAG.NE.7) GO TO 9
      IMUX1=2 ; DM3=1
10    IF(IMUX1.NE.1) GO TO 15
      MM1=1 ; MMF=64
      CALL MULTY(BUF11,BUF12,BUF13,BUF14,MM1,MMF,BUF31,BUF32,
     1BUF33,BUF34)
      IMUX1=1
      GO TO 13
15    MM1=1 ; MMF=64
      CALL MULTY(BUF31,BUF32,BUF33,BUF34,MM1,MMF,BUF11,BUF12,
     1BUF13,BUF14)
      IMUX1=2
13    IF((IMUX23.NE.1.AND.IMUX21.EQ.1) GO TO 11
      STAG=1 ; INV=1 ; DM2=2
      CALL STAG1(BUF31,BUF32,BUF33,BUF34,BUF21,BUF22,BUF23,BUF24,
     1NTT,STAG,INV)
12    STAG=STAG+1 ; DM3=2
      CALL ASTAG(BUF21,BUF22,BUF23,BUF24,BUF31,BUF32,BUF33,BUF34,
     1NTT,STAG,INV)
      STAG=STAG+1 ; DM2=2
      CALL ASTAG(BUF31,BUF32,BUF33,BUF34,BUF21,BUF22,BUF23,BUF24,
     1NTT,STAG,INV)
      IF(STAG.NE.7) GO TO 12
      DM2=1 ; DM4=0
      GO TO 16
11    STAG=1 ; INV=1 ; DM4=2
      CALL STAG1(BUF11,BUF12,BUF13,BUF14,BUF41,BUF42,BUF43,BUF44,
     1NTT,STAG,INV)
14    STAG=STAG+1 ; DM1=1
      CALL ASTAG(BUF41,BUF42,BUF43,BUF44,BUF11,BUF12,BUF13,BUF14,
     1NTT,STAG,INV)
      STAG=STAG+1 ; DM4=2
      CALL ASTAG(BUF11,BUF12,BUF13,BUF14,BUF41,BUF42,BUF43,BUF44,
     1NTT,STAG,INV)
      IF(STAG.NE.7) GO TO 14
      DM4=1 ; DM2=0
16    PRINT 777,(OUT(I,1),I=65,N)
777   FORMAT(30I4)
      PRINT 778,(AOUT(I),I=1,64)
778   FORMAT(30I4)
      DO 541 I4=65,N
541   AOUT(I)=OUT(I,2)
100   CONTINUE
      STOP
      END
C
C
C     SUBROUTINE TO SIMULATE NECESSARY BUFFER INTERCONNECTIONS
C     FOR THE FIRST STAGE.
C
C
      SUBROUTINE STAG1(X21,X22,X23,X24,X31,X32,X33,X34,NTT,
     1STAG,INV)
      IMPLICIT INTEGER (A-H,O-Z)
      EXTERNAL NTT
      DIMENSION X21(32,2),X22(32,2),X23(32,2),X24(32,2),
     2X31(32,2),X32(32,2),X33(32,2),X34(32,2)
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      DO 10 I=1,64
      A11=X21(1,1) ; A12=X21(1,2) ; B11=X23(1,1) ; B12=X23(1,2)
      DO 1 K1=1,31
      DO 1 J=1,2
1     X21(K1,J)=X21(K1+1,J)
      X21(32,1)=X22(1,1) ; X21(32,2)=X22(1,2)
      DO 2 K1=1,31
      DO 2 J=1,2
2     X22(K1,J)=X22(K1+1,J)
      DO 3 K1=1,31
      DO 3 J=1,2
3     X23(K1,J)=X23(K1+1,J)
      X23(32,1)=X24(1,1) ; X23(32,2)=X24(1,2)
      DO 4 K1=1,31
      DO 4 J=1,2
4     X24(K1,J)=X24(K1+1,J)
      CALL NTT(STAG,I,A11,A12,B11,B12,B01,B02,B03,B04,INV)
      DO 5 K1=1,31
      DO 5 J=1,2
5     X31(K1,J)=X31(K1+1,J)
      X31(32,1)=X32(1,1) ; X31(32,2)=X32(1,2)
      DO 6 K1=1,31
      DO 6 J=1,2
6     X32(K1,J)=X32(K1+1,J)
```

```
      DO 7 K1=1,31
      DO 7 J=1,2
    7 X33(K1,J)=X33(K1+1,J)
      X33(32,1)=X34(1,1) ; X33(32,2)=X34(1,2)
      DO 8 K1=1,31
      DO 8 J=1,2
    8 X34(K1,J)=X34(K1+1,J)
      X32(32,1)=B01 ; X32(32,2)=B02
      X34(32,1)=B03 ; X34(32,2)=B04
   10 CONTINUE
      RETURN
      END
C
C
C     SUBROUTINE TO SIMULATE THE BUFFER INTERCONNECTIONS
C     FOR THE STAGES 2 TO 7.
C
C
      SUBROUTINE ASTAG(X21,X22,X23,X24,X31,X32,X33,X
     134,NTT,STAG,INV)
      IMPLICIT INTEGER (A-H,O-Z)
      EXTERNAL NTT
      DIMENSION X21(32,2),X22(32,2),X23(32,2),X24(32,2),
     1X31(32,2),X32(32,2),X33(32,2),X34(32,2),
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      POS=1
      LSTAG=2**(STAG-2)
      MX=64/LSTAG/2
      DO 10 I=1,MX
      DO 2 K1=1,LSTAG
      A11=X21(1,1) ; A12=X21(1,2) ; B11=X22(1,1) ; B12=X22(1,2)
      DO 1 K2=1,31
      DO 1 J=1,2
      X21(K2,J)=X21(K2+1,J)
    1 X22(K2,J)=X22(K2+1,J)
      CALL NTT(STAG,POS,A11,A12,B11,B12,B01,B02,B03,B04,INV)
      POS=POS+1
      DO 3 K2=1,31
      DO 3 J=1,2
    3 X31(K2,J)=X31(K2+1,J)
      X31(32,1)=X32(1,1) ; X31(32,2)=X32(1,2)
      DO 4 K2=1,31
      DO 4 J=1,2
    4 X32(K2,J)=X32(K2+1,J)
      DO 5 K2=1,31
      DO 5 J=1,2
    5 X33(K2,J)=X33(K2+1,J)
      X33(32,1)=X34(1,1) ; X33(32,2)=X34(1,2)
      DO 6 K2=1,31
      DO 6 J=1,2
    6 X34(K2,J)=X34(K2+1,J)
      X32(32,1)=B01 ; X32(32,2)=B02
      X34(32,1)=B03 ;X34(32,2)=B04
    2 CONTINUE
      DO 8 K1=1,LSTAG
      A11=X23(1,1) ; A12=X23(1,2) ; B11=X24(1,1) ; B12=X24(1,2)
      DO 9 K2=1,31
      DO 9 J=1,2
      X23(K2,J)=X23(K2+1,J)
    9 X24(K2,J)=X24(K2+1,J)
      CALL NTT(STAG,POS,A11,A12,B11,B12,B01,B02,B03,B04,INV)
      POS=POS+1
      DO 12 K2=1,31
      DO 12 J=1,2
   12 X31(K2,J)=X31(K2+1,J)
      X31(32,1)=X32(1,1) ; X31(32,2)=X32(1,2)
      DO 13 K2=1,31
      DO 13 J=1,2
   13 X32(K2,J)=X32(K2+1,J)
      DO 14 K2=1,31
      DO 14 J=1,2
   14 X33(K2,J)=X33(K2+1,J)
      X33(32,1)=X34(1,1) ; X33(32,2)=X34(1,2)
      DO 15 K2=1,31
      DO 15 J=1,2
   15 X34(K2,J)=X34(K2+1,J)
      X32(32,1)=B01 ; X32(32,2)=B02
      X34(32,1)=B03 ;X34(32,2)=B04
    8 CONTINUE
   10 CONTINUE
      RETURN
      END
C
C
C     SUBROUTINE TO PERFORM THE BUTTERFLY CALCULATIONS.
C
C
      SUBROUTINE NTT(STAG,POS,A11,A12,B11,B12,B01,B02,B03,B04,
     1INV)
      IMPLICIT INTEGER (A-H,O-Z)
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      COMMON/BLK4/ MMOD,SUBM1,SUBM2,NP,MDOT1,MDOT2,IMDOT1,
     1IMDOT2,SUB,ALPSQ,N
      B01=A11+B11 ; B02=A12+B12
      B01=B01-MMOD*(B01/MMOD)
      B02=B02-MMOD*(B02/MMOD)
      E1=A11-B11 ; E2=A12-B12
      IF(E1.LT.0) E1=E1+MMOD
      IF(E2.LT.0) E2=E2+MMOD
      IF(INV.EQ.1) GO TO 220
      B03=E1*ALP1(STAG,POS)+E2*ALPSQ*ALP2(STAG,POS)
      B03=B03-MMOD*(B03/MMOD)
      B04=E1*ALP2(STAG,POS)+E2*ALP1(STAG,POS)
      B04=B04-MMOD*(B04/MMOD)
      GO TO 230
  220 B03=E1*ALPIN1(STAG,POS)+E2*ALPSQ*ALPIN2(STAG,POS)
      B03=B03-MMOD*(B03/MMOD)
      B04=E1*ALPIN2(STAG,POS)+E2*ALPIN1(STAG,POS)
      B04=B04-MMOD*(B04/MMOD)
  230 RETURN
```

```
      END

      SUBROUTINE TO SIMULATE THE BUFFER CONNECTIONS
      INPUT AND OUTPUT OPERATIONS.

      SUBROUTINE INOUT(X21,X22,X23,X24,IIN,OUT)
      IMPLICIT INTEGER (A-H,O-Z)
      DIMENSION X21(32,2),X22(32,2),X23(32,2),X24(32,2)
      DIMENSION IIN(128,2),OUT(128,2),IKK(128)
      DO 5 I=1,N
      OUT(I,1)=X21(I,1) ; OUT(I,2)=X21(I,2)
      DO 4 J=1,2
      DO 1 K1 1,31
    1 X21(K1,J)=X21(K1+1,J)
      X21(32,J)=X22(1,J)
      DO 2 K2=1,31
    2 X22(K2,J)=X22(K2+1,J)
      X22(32,J)=X23(1,J)
      DO 3 K3=1,31
    3 X23(K3,J)=X23(K3+1,J)
      X23(32,J)=X24(1,J)
      DO 4 K4=1,31
    4 X24(K4,J)=X24(K4+1,J)
      X24(32,1)=IIN(I,1) ; X24(32,2)=IIN(I,2)
    5 CONTINUE
      RETURN ; END

C
C     SUBROUTINE TO SIMULATE THE STRUCTURE OF AN EXTERNAL SERIAL
C     COMPLEX MULTIPLIER.
C
C
      SUBROUTINE MULTLY(X21,X22,X23,X24,MMI,MMF,X31,X32,X33,X34)
      IMPLICIT INTEGER (A-H,O-Z)
      DIMENSION X21(32,2),X22(32,2),X23(32,2),X24(32,2),X31(32,2),
     1X32(32,2),X33(32,2),X34(32,2)
      COMMON/BLK5/ OX(1,2),IN(1,2),H(128,2),X(1,2)
      COMMON/BLK2/ IND1(193),IND2(193),INV1(31,31),INV2(31,31)
      COMMON/BLK3/ ADD1(31,31),ADD2(31,31),SAD1(30,30),SAD2(31,31)
     1,RECT(30,31)
      COMMON/BLK4/ MMOD,SUBM1,SUBM2,NP,MDOT1,MDOT2,IMDOT1,
     1IMDOT2,SOD,ALPSO,N
      COMMON/BLK6/ STAG2,STAG3,STAG4,STAG5,STAG6,STAG7,STAG8,STAG9
     1,STAG10,STAG11,STAG12,STAG13,STAG14,STAG15,STAG16,STAG17,
     2STAG18,STAG19,STAG20,X1,X2,X3,X4
      LI=0
      INV1(31,31)=INV2(31,31)=0 ; ADD1(31,31)=30; ADD1(1,1)=0
      K1=4
      ISUB=0
      DO 10 I=MMI,MMF
      X(1,1)=X21(I,1) ; X(1,2)=X21(I,2)
      IN(1,1)=H(I,1) ; IN(1,2)=H(I,2)
      K1=1 KK=2
    2 STAG19=RECT(STAG17+1,STAG18+1)
      STAG18=ADD2(STAG8+1,STAG16+1)
      STAG17=ADD1(STAG7+1,STAG15+1)
      STAG16=INV2(STAG13+1,STAG14+1)
      STAG15=INV1(STAG13+1,STAG14+1)
      STAG14=ADD2(STAG11+1,STAG12+1)
      STAG13=ADD1(STAG9+1,STAG10+1)
      ISUB=ISUB+1
      IF(STAG13.EQ.30) STAG14=30
      IF(X4.EQ.0.OR.X3.EQ.0) GO TO 5
      STAG12=IND2(X4)
      STAG11=IND2(X3)
      STAG10=IND1(X4)
      STAG9=IND1(X3)
      GO TO 9
    5 STAG9=STAG10=30 ; STAG11=STAG12=0
    9 X4=X(1,1)
      X3=IN(1,K1)
      OX(1,K1)=STAG19
      K1=K1+1
      LI=LI+1
      IF(K1.GT.2) K1=1
      STAG8=INV2(STAG5+1,STAG6+1)
      STAG7=INV1(STAG5+1,STAG6+1)
      ISUB=ISUB+1
      IF(ISUB.NE.2) GO TO 15
      STAG6=SAD2(STAG3+1,STAG4+1)
      STAG5=SAD1(STAG20+1,STAG2+1)
      GO TO 16
   15 STAG6=ADD2(STAG3+1,STAG4+1)
      STAG5=ADD1(STAG20+1,STAG2+1)
   16 IF(STAG5.EQ.30) STAG6=30
      IF(X2.EQ.0.OR.X1.EQ.0) GO TO 7
      STAG4=IND2(X2)
      STAG3=IND2(X1)
      STAG2=IND1(X2)
      STAG20=IND1(X1)
      GO TO 6
    7 STAG20=STAG2=30 ; STAG3=STAG4=0
    6 X2=X(1,2)
      X1=IN(1,K2)
      LI=LI+1
      K2=K2-1
      IF(K2.LT.1) K2=1
      IF(ISUB.GE.4) ISUB=0
      IF(LI.LT.4) GO TO 2
      LI=0
      IF(KL.GT.3.OR.I.GE.65) GO TO 31
      DO 71 K1=1,31
      DO 71 J=1,2
   71 X31(K1,J)=X31(K1+1,J)
      X31(32,1)=X32(1,1) ; X31(32,2)=X32(1,2)
      DO 72 K1=1,31
      DO 72 J=1,2
   72 X32(K1,J)=X32(K1+1,J)
```

```
      X32(32,1)=X33(1,1) ; X32(32,2)=X33(1,2)
      DO 73 K1=1,31
      DO 73 J=1,2
   73 X33(K1,J)=X33(K1+1,J)
      X33(32,1)=X34(1,1) ; X33(32,2)=X34(1,2)
      DO 74 K1=1,31
      DO 74 J=1,2
   74 X34(K1,J)=X34(K1+1,J)
      X34(32,1)=0X(1,1) ; X34(32,2)=0X(1,2)
   31 DO 1 K1=1,31
      DO 1 J=1,2
    1 X21(K1,J)=X21(K1+1,J)
      X21(32,1)=X22(1,1) ; X21(32,2)=X22(1,2)
      DO 12 K1=1,31
      DO 12 J=1,2
   12 X22(K1,J)=X22(K1+1,J)
      X22(32,1)=X23(1,1) ; X22(32,2)=X23(1,2)
      DO 3 K1=1,31
      DO 3 J=1,2
    3 X23(K1,J)=X23(K1+1,J)
      X23(32,1)=X24(1,1) ; X23(32,2)=X24(1,2)
      DO 4 K1=1,31
      DO 4 J=1,2
    4 X24(K1,J)=X24(K1+1,J)
      IF(K1.LE.3.AND.I.LT.65) GO TO 32
      X24(32,1)=0X(1,1) ; X24(32,2)=0X(1,2)
   32 K1=K1+1
   10 CONTINUE
      RETURN
      END
C
C
C     SUBROUTINE TO GENERATE NECESSARY LOOK-UP TABLES.
C
C
      SUBROUTINE LTABLE
      IMPLICIT INTEGER (A-H,O-Z)
      COMMON/BLK2/ IND1(193),IND2(193),INV1(31,311),INV2(31,31)
      COMMON/BLK3/ ADD1(31,31),ADD2(31,31),SAD1(30,30),SAD2(31,31)
     1,RECT(30,31)
      DIMENSION I1(30),I2(31),INVD1(193),INVD2(193),IN(193)
      COMMON/BLK4/ MMOD, SUBM1,SUBM2,NP,MDOT1,MDOT2,IMDOT1,
     1IMDOT2,SOD,ALPSO,N
C
C
C     CALCULATION OF RECONSTRUCTION TABLE FROM MODS 30 AND 31 .
C
C
      DO 71 I=1,SUBM1
      K=I-1
      I1(I)=K*IMDOT1-SUBM1*(K*IMDOT1/SUBM1)
   71 I1(I)=I1(I)*SUBM2
      DO 72 I=1,SUBM2
      K=I-1
      I2(I)=K*IMDOT2-SUBM2*(K*IMDOT2/SUBM2)
   72 I2(I)=I2(I)*SUBM1
      DO 73 J=1,SUBM2
      DO 73 I=1,SUBM1
      RECT(I,J)=I1(I)+I2(J)
      RECT(I,J)=RECT(I,J)-SOD*(RECT(I,J)/SOD)
      RECT(I,J)=RECT(I,J)-MMOD*(RECT(I,J)/MMOD)
   73 CONTINUE
C
C
C     CALCULATION OF INDEX AND INVERSE INDEX LOOK-UP
C     TABLES AT MOD 30 & 31.
C
      IND1(1)=0
      IND1(3)=0
      IN(1)=1
      IN(193)=0
      INVD1(193)=0
      INVD2(193)=0
      MMOD1=MMOD-1
      DO 77 I=2,MMOD1
      IMDI=IN(I-1)
      IMD1=IMD1*NP-MMOD*(IMD1*NP/MMOD)
      IN(I)=IMD1
   77 IND1(IMD1)=I-1
      DO 78 I=1,MMOD1
      IT1=IND1(I)
      IT2=IN(I)
      IND1(I)=IT1-SUBM1*(IT1/SUBM1)
      IND2(I)=IT1-SUBM2*(IT1/SUBM2)
      INVD1(I)=IT2-SUBM1*(IT2/SUBM1)
      INVD2(I)=IT2-SUBM2*(IT2/SUBM2)
   78 CONTINUE
      DO 81 I=1,SUBM1
      DO 81 J=1,SUBM2
      IP=((I-1)*SUBM2*IMDOT1+(J-1)*SUBM1*IMDOT2)
      IF(IP.LT.SOD) GO TO 82
      L=IP/SOD ; IP=IP-L*SOD
   82 IF(IP.LT.MMOD1) GO TO 83
      LM=IP/MMOD1 ; IP=IP-LM*MMOD1
      IF(IP.GE.450) IP=1
   83 IP1=INVD1(IP+1)
      IP2=INVD2(IP+1)
      INV1(I,J)=IP1
   81 INV2(I,J)=IP2
C
C
C     CALCULATION OF ADDITION AND SUBTRACTION LOOK TABLE FOR MOD
C     30 AND 31 .
C
C
      DO 86 I=1,SUBM1
      DO 86 J=1,SUBM1
      II=I-1
      JJ=J-1
      LLL=III-JJ
```

```
      ADD1(I,J)=(I-1)*SUBM1+(L-1)/SUBM1)
   86 CONTINUE
      DO 87 I=1,SUBM2
      DO 87 J=1,SUBM2
      III=I-1
      JJJ=J-1
      L=(I-1)*SUBM2+JJJ
      ADD2(I,J)=(L-1)*SUBM2+(L-1)/SUBM2)
   87 CONTINUE
      DO 88 I=1,SUBM1
      DO 88 J=1,SUBM1
      IS1=ADD1(I,J)+IND1(ALPSQ)
      SAD1(I,J)=MOD(IS1,MMOD)
   88 CONTINUE
      DO 89 I=1,SUBM2
      DO 89 J=1,SUBM2
      IS2=ADD2(I,J)+IND2(ALPSQ)
      SAD2(I,J)=MOD(IS2,MMOD)
   89 CONTINUE
      RETURN
      END
C
C
C     SUBROUTINE TO GENETRATE LOOK-UP TABLES FOR POWERS
C     OF ALPH'S AND INVERSE ALPH'S.
C
C
      SUBROUTINE TWLF
      IMPLICIT INTEGER (A-H,O-Z)
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      COMMON/BLK4/ MMOD,SUBM1,SUBM2,NP,MDOT1,MDOT2,IMDOT1,
     1IMDOT2,SOD,ALPSQ,N
      DIMENSION A(128,2),IA(128,2)
      A(1,1)=IA(1,1)=1
      A(1,2)=IA(1,2)=0
      A(2,1)=0 ; A(2,2)=1
C
C
C     CALCULATION OF THE POWERS OF ALPH.
C
C
      DO 31 I=3,N
      A(I,1)=A(I-1,1)*A(2,1)+ALPSQ*A(I-1,2)*A(2,2)
      A(I,2)=A(I-1,1)*A(2,2)+A(I-1,2)*A(2,1)
      A(I,1)=A(I,1)-MMOD*(A(I,1)/MMOD)
   31 A(I,2)=A(I,2)-MMOD*(A(I,2)/MMOD)
      DO 32 I=2,N
      IA(I,1)=A(130-I,1)
   32 IA(I,2)=A(130-I,2)
C
C
C     CALCULATION OF POWERS OF INVERSE OF ALPH.
C
C
      DO 34 I=1,64
      ALP1(I,I)=A(I,1)
      ALP2(I,I)=A(I,2)
      ALPIN1(I,I)=IA(I,1)
      ALPIN2(I,I)=IA(I,2)
   34 CONTINUE
      ISTAG=2
   33 I1=1 ; L=I
      NSTAG=2**(ISTAG-1)
      MSTAG=64/NSTAG
      DO 36 I=1,MSTAG
      DO 35 K=1,NSTAG
      ALP1(ISTAG,I1)=A(L,1)
      ALP2(ISTAG,I1)=A(L,2)
      ALPIN1(ISTAG,I1)=IA(L,1)
      ALPIN2(ISTAG,I1)=IA(L,2)
   35 I1=I1+1
   36 L=L+NSTAG
      ISTAG=ISTAG+1
      IF(ISTAG.LE.7) GO TO 33.
      RETURN
      END
C
C
C     SUBROUTINE TO CONVERT THE REAL VALUED INPUT SEQUENCE
C     TO COMPLEX INPUT SEQUENCE FOR AN NIT PROCESSOR.
C
C
      SUBROUTINE OVLAP(IIN,ITS,IKN)
      IMPLICIT INTEGER (A-H,O-Z)
      DIMENSION IIN(128,2),ITS(64),IKN(128)
      DO 181 I1=1,64
      IIN(I1,1)=ITS(I1)
      IIN(I1,2)=IKN(I1)
      DO 182 I2=1,63
  182 ITS(I2)=ITS(I2+1)
      IST(64)=IKN(I1)
  181 CONTINUE
      DO 183 I1=1,64
      IIN(I1+64,1)=ITS(I1)
      IIN(I1+64,2)=IKN(I1+64)
      DO 184 I2=1,63
  184 ITS(I2)=ITS(I2+1)
      ITS(64)=IKN(I1+64)
  183 CONTINUE
      RETURN
      END
```

```
C     ***************************************************************
C     *                                                             *
C     *  MAIN LINE PROGRAM TO SIMULATE AN NTT PROCESSOR             *
C     *  MODULE 449.                                                *
C     *                                                             *
C     ***************************************************************
      IMPLICIT INTEGER (A-H,O-Z)
      EXTERNAL NTT
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      COMMON/BLK2/ IND1(449),IND2(449),INV1(31,31),INV2(31,31)
      COMMON/BLK3/ ADD1(31,31),ADD2(31,31),SAD1(30,30),SAD2(31,31)
     1,REC1(30,31)
      COMMON/BLK4/ MMOD,SUBM1,SUBM2,NP,MDOT1,MDOT2,IMDOT1,
     1IMDOT2,SOD,ALPSO,N
      COMMON/BLK5/ DX(1,2),DD(1,2),H(128,2),X(1,2)
      COMMON/BLK6/ STAG2,STAG3,STAG4,STAG5,STAG6,STAG7,STAG8,STAG9
     1,STAG10,STAG11,STAG12,STAG13,STAG14,STAG15,STAG16,STAG17,
     2STAG18,STAG19,STAG20,X1,X2,X3,X4
      DIMENSION BUF11(32,2),BUF12(32,2),BUF13(32,2),BUF14(32,2),
     1BUF21(32,2),BUF22(32,2),BUF23(32,2),BUF24(32,2),AOUT(64),
     2BUF31(32,2),BUF32(32,2),BUF33(32,2),BUF34(32,2),
     3BUF41(32,2),BUF42(32,2),BUF43(32,2),BUF44(32,2)
     4,OUT(128,2),IIN(120,2),ITS(64),IKN(120)
C
C
C     POWER ON RESET OF LATCHES AND BUFFERS.
C
      STAG2=STAG3=STAG4=STAG5=STAG6=STAG7=STAG8=STAG9=STAG10=
     2STAG11=STAG12=STAG13=STAG14=STAG15=STAG16=STAG17=STAG18=
     3STAG19=STAG20=X1=X2=X3=X4=0
      DO 2 I=1,32
      ITS(I)=ITS(I+32)=AOUT(I)=AOUT(I+32)=0
      DO 2 J=1,2
      BUF11(I,J)=BUF12(I,J)=BUF13(I,J)=BUF14(I,J)=0
      BUF21(I,J)=BUF22(I,J)=BUF23(I,J)=BUF24(I,J)=0
      BUF31(I,J)=BUF32(I,J)=BUF33(I,J)=BUF34(I,J)=0
      BUF41(I,J)=BUF42(I,J)=BUF43(I,J)=BUF44(I,J)=0
      IIN(I,J)=IIN(I+32,J+32)=IIN(I+64,J+64)=IIN(I+96,J+96)=0
    2 CONTINUE
C
C
C     DEFINING THE PARAMETERS FOR LOOK-UP TABLES CALCULATIONS.
C     N=TRANSFORM LENGTH.
C     MMOD=MAIN MODULUS .
C     ALPSO=SOURCE OF THE CYCLIC GROUP GENERATOR.
C     MIV=MULTIPLICATIVE INVERSE OF N.
C     NP=MULTIPLICATIVE INVERSE OF MMOD.
C     SUBM1 & SUBM2 = SUB MODULI.
C     IMDOT1 & IMDOT2 = MULTIPLICATIVE INVERSE OF SUBM1 & SUBM2.
C
C
      MMOD=449; SUBM1=30 ; SUBM2=31 ; SOD=SUBM1*SUBM2 ; NP=5
      IMDOT1=-1 ; IMDOT2=30 ;MIV=221; ALPSO=391; N=128
C
C
C     SUBROUTINE TWLF TO CALCULATE THE THE POWERS OF
C     CYCLIC GROUP GENERATOR ALPHA.
C
C
      CALL TWLF
C
C
C     SUBROUTINE TO CALCULATE THE VARIOUS LOOK-UP TABLES.
C
C
      CALL LTABLE
C
C
C     SUBROUTINE TO CALCULATE THE FILTER RESPOSE.
C
C
      CALL FILTER (H)
C
C
C     MULTIPLY THE FILTER COEFFIENTS BY MULTIPLICATIVE INVERSE
C     OF TRANSFORM LENGTH N.
C
C
      DO 997 I=1,N
      DO 997 J=1,2
      H(I,J)=MIRV*H(I,J)
  997 H(I,J)=H(I,J)-MMOD*(H(I,J)/MMOD)
C
C
C     SUBROUTINE TO GET INPUT SEQUENCE .
C
C
      CALL INFILE(IKN)
C
C
      DM4=1 ; DM2=0 ; IMUX1=2 ; DM3=1 ; IMUX22=1
      DO 100 K=1,8
C
C
C     SUBROUTINE TO ARRANGE THE INPUT SEQUENCE IN THE FORM
C     REQUIRED FOR OVER-LAP SAVE METHOD.
C
C
      CALL OVLAP(IIN,ITS,IKN)
      IF(DM4.NE.1.AND.DM2.EQ.1) GO TO 3
      CALL INOUT(BUF41,BUF42,BUF43,BUF44,IIN,OUT)
      OPMUX=2 ; IMUX22=1 ; IMUX24=0
      GO TO 4
    3 CALL INOUT(BUF21,BUF22,BUF23,BUF24,IIN,OUT)
      OPMUX=1 ; IMUX24=1 ; IMUX22=0
    4 IF(IMUX1.NE.2) GO TO 5
      MMI=65 ; MMF=128
      CALL MULTLY(BUF31,BUF32,BUF33,BUF34,MMI,MMF,BUF11,BUF12,
     1BUF13,BUF14)
```

```
      IMUX23=1 ; IMUX21=0
      GO TO 5
    5 MMI=25 ; MMF=128
      CALL MULTLY(BUF11,BUF12,BUF13,BUF14,MMI,MMF,BUF31,BUF32,
     1BUF33,BUF34)
      IMUX21=1 ; IMUX23=0
    6 IF(IMUX22.NE.1.AND.IMUX24.EQ.1) GO TO 7
      STAG=1 ; INV=0 ; DM1=1
      CALL STAG1(BUF21,BUF22,BUF23,BUF24,BUF11,BUF12,BUF13,BUF14,
     1NTT,STAG,INV)
    8 STAG=STAG+1 ; DM2=2
      CALL ASTAG(BUF11,BUF12,BUF13,BUF14,BUF21,BUF22,BUF23,BUF24,
     1NTT,STAG,INV)
      STAG=STAG+1 ; DM1=1
      CALL ASTAG(BUF21,BUF22,BUF23,BUF24,BUF11,BUF12,BUF13,BUF14,
     1NTT,STAG,INV)
      IF(STAG.NE.7) GO TO 8
      IMUX1=1 ; DM1=2
      GO TO 10
    7 STAG=1 ; INV=0 ; DM3=2
      CALL STAG1(BUF41,BUF42,BUF43,BUF44,BUF31,BUF32,BUF33,BUF34,
     1NTT,STAG,INV)
    9 STAG=STAG+1 ; DM4=2
      CALL ASTAG(BUF31,BUF32,BUF33,BUF34,BUF41,BUF42,BUF43,BUF44,
     1NTT,STAG,INV)
      STAG=STAG+1 ; DM3=2
      CALL ASTAG(BUF41,BUF42,BUF43,BUF44,BUF31,BUF32,BUF33,BUF34,
     1NTT,STAG,INV)
      IF(STAG.NE.7) GO TO 9
      IMUX1=2 ; DM3=1
   10 IF(IMUX1.NE.1) GO TO 15
      MMI=1 ; MMF=64
      CALL MULTLY(BUF11,BUF12,BUF13,BUF14,MMI,MMF,BUF31,BUF32,
     1BUF33,BUF34)
      IMUX1=1
      GO TO 13
   15 MMI=1 ; MMF=64
      CALL MULTLY(BUF31,BUF32,BUF33,BUF34,MMI,MMF,BUF11,BUF12,
     1BUF13,BUF14)
      IMUX1=2
   13 IF(IMUX23.NE.1.AND.IMUX21.EQ.1) GO TO 11
      STAG=1 ; INV=1 ; DM2=2
      CALL STAG1(BUF31,BUF32,BUF33,BUF34,BUF21,BUF22,BUF23,BUF24,
     1NTT,STAG,INV)
   12 STAG=STAG+1 ; DM3=2
      CALL ASTAG(BUF21,BUF22,BUF23,BUF24,BUF31,BUF32,BUF33,BUF34,
     1NTT,STAG,INV)
      STAG=STAG+1 ; DM2=2
      CALL ASTAG(BUF31,BUF32,BUF33,BUF34,BUF21,BUF22,BUF23,BUF24,
     1NTT,STAG,INV)
      IF(STAG.NE.7) GO TO 12
      DM4=1 ; DM4=0
      GO TO 16
   11 STAG=1 ; INV=1 ; DM4=2
      CALL STAG1(BUF11,BUF12,BUF13,BUF14,BUF41,BUF42,BUF43,BUF44,
     1NTT,STAG,INV)
   14 STAG=STAG+1 ; DM1=1
      CALL ASTAG(BUF41,BUF42,BUF43,BUF44,BUF11,BUF12,BUF13,BUF14,
     1NTT,STAG,INV)
      STAG=STAG+1 ; DM4=2
      CALL ASTAG(BUF11,BUF12,BUF13,BUF14,BUF41,BUF42,BUF43,BUF44,
     1NTT,STAG,INV)
      IF(STAG.NE.7) GO TO 14
      DM4=1 ; DM2=0
   16 PRINT 777,(OUT(I,1),I=65,N)
  777 FORMAT(3014)
      PRINT 778,(AOUT(I),I=1,64)
  778 FORMAT(3014)
      DO 541 I4=65,N
  541 AOUT(I)=OUT(I,2)
  100 CONTINUE
      STOP
      END
C
C
C     SUBROUTINE TO SIMULATE NECESSARY BUFFER INTERCONNECTIONS
C     FOR THE FIRST STAGE.
C
C
      SUBROUTINE STAG1(X21,X22,X23,X24,X31,X32,X33,X34,NTT,
     1STAG,INV)
      IMPLICIT INTEGER (A-H,O-Z)
      EXTERNAL NTT
      DIMENSION X21(32,2),X22(32,2),X23(32,2),X24(32,2),
     2X31(32,2),X32(32,2),X33(32,2),X34(32,2)
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      DO 10 I=1,64
      A11=X21(1,1) ; A12=X21(1,2) ; B11=X23(1,1) ; B12=X23(1,2)
      DO 1 K1=1,31
      DO 1 J=1,2
    1 X21(K1,J)=X21(K1+1,J)
      X21(32,1)=X22(1,1) ; X21(32,2)=X22(1,2)
      DO 2 K1=1,31
      DO 2 J=1,2
    2 X22(K1,J)=X22(K1+1,J)
      DO 3 K1=1,31
      DO 3 J=1,2
    3 X23(K1,J)=X23(K1+1,J)
      X23(32,1)=X24(1,1) ; X23(32,2)=X24(1,2)
      DO 4 K1=1,31
      DO 4 J=1,2
    4 X24(K1,J)=X24(K1+1,J)
      CALL NIT(STAG,I,A11,A12,B11,B12,B01,B02,B03,B04,INV)
      DO 5 K1=1,31
      DO 5 J=1,2
    5 X31(K1,J)=X31(K1+1,J)
      X31(32,1)=X32(1,1) ; X31(32,2)=X32(1,2)
      DO 6 K1=1,31
      DO 6 J=1,2
    6 X32(K1,J)=X32(K1+1,J)
```

```
      DO 7 K1=1,31
      DO 7 J=1,2
    7 X33(K1,J)=X33(K1+1,J)
      X33(32,1)=X34(1,1) ; X33(32,2)=X34(1,2)
      DO 8 K1=1,31
      DO 8 J=1,2
    8 X34(K1,J)=X34(K1+1,J)
      X34(32,1)=B01 ; X32(32,2)=B02
      X34(32,1)=B03 ; X34(32,2)=B04
   10 CONTINUE
      RETURN
      END
C
C
C     SUBROUTINE TO SIMULATE THE BUFFER INTERCONNECTIONS
C     FOR THE STAGES 2 TO 7.
C
C
      SUBROUTINE ASTAG(X21,X22,X23,X24,X31,X32,X33,X
     134,NTT,STAG,INV)
      IMPLICIT INTEGER (A-H,O-Z)
      EXTERNAL NTT
      DIMENSION X21(32,2),X22(32,2),X23(32,2),X24(32,2),
     1X31(32,2),X32(32,2),X33(32,2),X34(32,2)
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      POS=1
      LSTAG=2**(STAG-2)
      MX=64/LSTAG/2
      DO 10 I=1,MX
      DO 2 K1=1,LSTAG
      A11=X21(1,1) ; A12=X21(1,2) ; B11=X22(1,1) ; B12=X22(1,2)
      DO 1 K2=1,31
      DO 1 J=1,2
      X21(K2,J)=X21(K2+1,J)
    1 X22(K2,J)=X22(K2+1,J)
      CALL NTT(STAG,POS,A11,A12,B11,B12,B01,B02,B03,B04,INV)
      POS=POS+1
      DO 4 K2=1,31
      DO 3 J=1,2
    3 X31(K2,J)=X31(K2+1,J)
      X31(32,1)=X32(1,1) ; X31(32,2)=X32(1,2)
      DO 4 K2=1,31
      DO 4 J=1,2
    4 X32(K2,J)=X32(K2+1,J)
      DO 5 K2=1,31
      DO 5 J=1,2
    5 X33(K2,J)=X33(K2+1,J)
      X33(32,1)=X34(1,1) ; X33(32,2)=X34(1,2)
      DO 6 K2=1,31
      DO 6 J=1,2
    6 X34(K2,J)=X34(K2+1,J)
      X32(32,1)=B01 ; X32(32,2)=B02
      X34(32,1)=B03 ;X34(32,2)=B04
    2 CONTINUE
      DO 8 K1=1,LSTAG
      A11=X23(1,1) ; A12=X23(1,2) ; B11=X24(1,1) ; B12=X24(1,2)
      DO 9 K2=1,31
      DO 9 J=1,2
      X23(K2,J)=X23(K2+1,J)
    9 X24(K2,J)=X24(K2+1,J)
      CALL NTT(STAG,POS,A11,A12,B11,B12,B01,B02,B03,B04,INV)
      POS=POS+1
      DO 12 K2=1,31
      DO 12 J=1,2
   12 X31(K2,J)=X31(K2+1,J)
      X31(32,1)=X32(1,1) ; X31(32,2)=X32(1,2)
      DO 13 K2=1,31
      DO 13 J=1,2
   13 X32(K2,J)=X32(K2+1,J)
      DO 14 K2=1,31
      DO 14 J=1,2
   14 X33(K2,J)=X33(K2+1,J)
      X33(32,1)=X34(1,1) ; X33(32,2)=X34(1,2)
      DO 15 K2=1,31
      DO 15 J=1,2
   15 X34(K2,J)=X34(K2+1,J)
      X32(32,1)=B01 ; X32(32,2)=B02
      X34(32,1)=B03 ;X34(32,2)=B04
    8 CONTINUE
   10 CONTINUE
      RETURN
      END
C
C
C     SUBROUTINE TO PERFORM THE BUTTERFLY CALCULATIONS.
C
C
      SUBROUTINE NTT(STAG,POS,A11,A12,B11,B12,B01,B02,B03,B04,
     1INV)
      IMPLICIT INTEGER (A-H,O-Z)
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      COMMON/BLK4/ MMOD,SUBM1,SUBM2,NP,MDOT1,MDOT2,INDOT1,
     1INDOT2,SUD,ALPSQ,N
      B01=A11+B11 ; B02=A12+B12
      B01=B01-MMOD*(B01/MMOD)
      B02=B02-MMOD*(B02/MMOD)
      E1=A11-B11 ; E2=A12-B12
      IF(E1.LT.0) E1=E1+MMOD
      IF(E2.LT.0) E2=E2+MMOD
      IF(INV.EQ.1) GO TO 220
      B03=E1*ALP1(STAG,POS)+E2*ALPSQ*ALP2(STAG,POS)
      B03=B03-MMOD*(B03/MMOD)
      B04=E1*ALP2(STAG,POS)+E2*ALP1(STAG,POS)
      B04=B04-MMOD*(B04/MMOD)
      GO TO 230
  220 B03=E1*ALPIN1(STAG,POS)+E2*ALPSQ*ALPIN2(STAG,POS)
      B03=B03-MMOD*(B03/MMOD)
      B04=E1*ALPIN2(STAG,POS)+E2*ALPIN1(STAG,POS)
      B04=B04-MMOD*(B04/MMOD)
  230 RETURN
```

```
      END
C
C
C     SUBROUTINE TO SIMULATE THE BUFFER CONNECTIONS
C     INPUT AND OUTPUT OPERATIONS.
C
C
      SUBROUTINE INOUT(X21,X22,X23,X24,IIN,OUT)
      IMPLICIT INTEGER (A-H,O-Z)
      DIMENSION X21(32,2),X22(32,2),X23(32,2),X24(32,2)
      DIMENSION IIN(128,2),OUT(128,2),IKN(128)
      DO 5 I=1,N
      OUT(I,1)=X21(1,1) ; OUT(I,2)=X21(1,2)
      DO 4 J=1,2
      DO 1 K1=1,31
    1 X21(K1,J)=X21(K1+1,J)
      X21(32,J)=X22(1,J)
      DO 2 K2=1,31
    2 X22(K2,J)=X22(K2+1,J)
      X22(32,J)=X23(1,J)
      DO 3 K3=1,31
    3 X23(K3,J)=X23(K3+1,J)
      X23(32,J)=X24(1,J)
      DO 4 K4=1,31
    4 X24(K4,J)=X24(K4+1,J)
      X24(32,1)=IIN(I,1) ; X24(32,2)=IIN(I,2)
    5 CONTINUE
      RETURN ; END
C
C
C     SUBROUTINE TO SIMULATE THE STRUCTURE OF AN EXTERNAL SERIAL
C     COMPLEX MULTIPLIER.
C
C
      SUBROUTINE MULTLY(X21,X22,X23,X24,MMI,MMF,X31,X32,X33,X34)
      IMPLICIT INTEGER (A-H,O-Z)
      DIMENSION X21(32,2),X22(32,2),X23(32,2),X24(32,2),X31(32,2),
     1X32(32,2),X33(32,2),X34(32,2)
      COMMON/BLK5/ OX(1,2),IH(1,2),H(128,2),X(1,2)
      COMMON/BLK2/ IND1(449),IND2(449),INV1(31,31),INV2(31,31)
      COMMON/BLK3/ ADD1(31,31),ADD2(31,31),SAD1(30,30),SAD2(31,31)
     1,RECT(30,31)
      COMMON/BLK4/ MMOD,SUBM1,SUBM2,NP,MDOT1,MDOT2,IMDOT1,
     1IMDOT2,SOD,ALPSD,N
      COMMON/BLK6/ STAG2,STAG3,STAG4,STAG5,STAG6,STAG7,STAG8,STAG9
     1,STAG10,STAG11,STAG12,STAG13,STAG14,STAG15,STAG16,STAG17,
     2STAG18,STAG19,STAG20,X1,X2,X3,X4
      LI=0
      INV1(31,31)=INV2(31,31)=0 ; ADD1(31,31)=30; ADD1(1,1)=0
      KL=1
      ISUB=0
      DO 10 I=MMI,MMF
      X(1,1)=X21(1,1) ; X(1,2)=X21(1,2)
      IH(1,1)=H(I,1) ; IH(1,2)=H(I,2)
      K1=1 ;K2=2
    2 STAG19=RECT(STAG17+1,STAG18+1)
      STAG18=ADD2(STAG8+1,STAG16+1)
      STAG17=ADD1(STAG7+1,STAG15+1)
      STAG16=INV2(STAG13+1,STAG14+1)
      STAG15=INV1(STAG13+1,STAG14+1)
      STAG14=ADD2(STAG11+1,STAG12+1)
      STAG13=ADD1(STAG9+1,STAG10+1)
      ISUB=ISUB+1
      IF(STAG13.EQ.30) STAG14=30
      IF(X4.EQ.0.OR.X3.EQ.0) GO TO 5
      STAG12=IND2(X4)
      STAG11=IND2(X3)
      STAG10=IND1(X4)
      STAG9=IND1(X3)
      GO TO 9
    5 STAG9=STAG10=30 ; STAG11=STAG12=0
    9 X4=X(1,1)
      X3=IH(1,K1)
      OX(1,K1)=STAG19
      K1=K1+1
      LI=LI+1
      IF(K1.GT.2) K1=1
      STAG8=INV2(STAG5+1,STAG6+1)
      STAG7=INV1(STAG5+1,STAG6+1)
      ISUB=ISUB+1
      IF(ISUB.NE.2) GO TO 15
      STAG6=SAD2(STAG3+1,STAG4+1)
      STAG5=SAD1(STAG20+1,STAG2+1)
      GO TO 16
   15 STAG6=ADD2(STAG3+1,STAG4+1)
      STAG5=ADD1(STAG20+1,STAG2+1)
   16 IF(STAG5.EQ.30) STAG6=30
      IF(X2.EQ.0.OR.X1.EQ.0) GO TO 7
      STAG4=IND2(X2)
      STAG3=IND2(X1)
      STAG2=IND1(X2)
      STAG20=IND1(X1)
      GO TO 6
    7 STAG20=STAG2=30 ; STAG3=STAG4=0
    6 X2=X(1,2)
      X1=IH(1,K2)
      LI=LI+1
      K2=K2-1
      IF(K2.LT.1) K2=1
      IF(ISUB.GE.4) ISUB=0
      IF(LI.LT.4) GO TO 2
      LI=0
      IF(KL.GT.3.OR.I.GE.65) GO TO 31
      DO 71 K1=1,31
      DO 71 J=1,2
   71 X31(K1,J)=X31(K1+1,J)
      X31(32,1)=X32(1,1) ; X31(32,2)=X32(1,2)
      DO 72 K1=1,31
      DO 72 J=1,2
   72 X32(K1,J)=X32(K1+1,J)
```

```
      X32(32,1)=X33(1,1) ; X32(32,2)=X33(1,2)
      DO 73 K1=1,31
      DO 73 J=1,2
   73 X33(K1,J)=X33(K1+1,J)
      X33(32,1)=X34(1,1) ; X33(32,2)=X34(1,2)
      DO 74 K1=1,31
      DO 74 J=1,2
   74 X34(K1,J)=X34(K1+1,J)
      X34(32,1)=OX(1,1) ; X34(32,2)=OX(1,2)
   31 DO 1 K1=1,31
      DO 1 J=1,2
    1 X21(K1,J)=X21(K1+1,J)
      X21(32,1)=X22(1,1) ; X21(32,2)=X22(1,2)
      DO 12 K1=1,31
      DO 12 J=1,2
   12 X22(K1,J)=X22(K1+1,J)
      X22(32,1)=X23(1,1) ; X22(32,2)=X23(1,2)
      DO 3 K1=1,31
      DO 3 J=1,2
    3 X23(K1,J)=X23(K1+1,J)
      X23(32,1)=X24(1,1) ; X23(32,2)=X24(1,2)
      DO 4 K1=1,31
      DO 4 J=1,2
    4 X24(K1,J)=X24(K1+1,J)
      IF(KL.LE.3.AND.I.LT.65) GO TO 32
      X24(32,1)=OX(1,1) ; X24(32,2)=OX(1,2)
   32 KL=KL+1
   10 CONTINUE
      RETURN
      END
C
C
C     SUBROUTINE TO GENERATE NECESSARY LOOK-UP TABLES.
C
C
      SUBROUTINE LTABLE
      IMPLICIT INTEGER (A-H,O-Z)
      COMMON/BLK2/ IND1(449),IND2(449),INV1(31,311),INV2(31,31)
      COMMON/BLK3/ ADD1(31,31),ADD2(31,31),SAD1(30,30),SAD2(31,31)
     1,RECT(30,31)
      DIMENSION T1(30),T2(31),INVD1(449),INVD2(449),IN(449)
      COMMON/BLK4/ MMOD, SUBM1,SUBM2,NP,MDOT1,MDOT2,IMDOT1,
     1IMDOT2,SOD,ALPSO,N
C
C
C     CALCULATION OF RECONSTRUCTION-TABLE FROM MODS 30 AND 31 .
C
C
      DO 71 I=1,SUBM1
      K=I-1
      T1(I)=K*IMDOT1-SUBM1*(K*IMDOT1/SUBM1)
   71 T1(I)=T1(I)*SUBM2
      DO 72 I=1,SUBM2
      K=I-1
      T2(I)=K*IMDOT2-SUBM2*(K*IMDOT2/SUBM2)
   72 T2(I)=T2(I)*SUBM1
      DO 73 J=1,SUBM2
      DO 73 I=1,SUBM1
      RECT(I,J)=T1(I)+T2(J)
      RECT(I,J)=RECT(I,J)-SOD*(RECT(I,J)/SOD)
      RECT(I,J)=RECT(I,J)-MMOD*(RECT(I,J)/MMOD)
   73 CONTINUE
C
C
C     CALCULATION OF INDEX AND INVERSE INDEX LOOK-UP
C     TABLES AT MOD 30 & 31.
C
      IND1(1)=0
      IND1(3)=0
      IN(1)=1
      IN(449)=0
      INVD1(449)=0
      INVD2(449)=0
      MMOD1=MMOD-1
      DO 77 I=2,MMOD1
      IMD1=IN(I-1)
      IMD1=IMD1*NP-MMOD*(IMD1*NP/MMOD)
      IN(I)=IMD1
   77 IND1(IMD1)=I-1
      DO 78 I=1,MMOD1
      IT1=IND1(I)
      IT2=IN(I)
      IND1(I)=IT1-SUBM1*(IT1/SUBM1)
      IND2(I)=IT1-SUBM2*(IT1/SUBM2)
      INVD1(I)=IT2-SUBM1*(IT2/SUBM1)
      INVD2(I)=IT2-SUBM2*(IT2/SUBM2)
   78 CONTINUE
      DO 81 I=1,SUBM1
      DO 81 J=1,SUBM2
      IP=((I-1)*SUBM2*IMDOT1+(J-1)*SUBM1*IMDOT2)
      IF(IP.LT.SOD) GO TO 82
      L=IP/SOD ; IP=IP-L*SOD
   82 IF(IP.LT.MMOD1) GO TO 83
      LM=IP/MMOD1 ; IP=IP-LM*MMOD1
      IF(IP.GE.450) IP=1
   83 IP1=INVD1(IP+1)
      IP2=INVD2(IP+1)
      INV1(I,J)=IP1
   81 INV2(I,J)=IP2
C
C
C     CALCULATION OF ADDITION AND SUBTRACTION LOOK TABLE FOR MOD
C     30 AND 31 .
C
C
      DO 86 I=1,SUBM1
      DO 86 J=1,SUBM1
      II=I-1
      JJ=J-1
      LLL=II+JJ
```

```
      ADD1(I,J)=LLL-SUBM1*(LLL/SUBM1)
   86 CONTINUE
      DO 87 I=1,SUBM2
      DO 87 J=1,SUBM2
      III=I-1
      JJJ=J-1
      LLLL=III+JJJ
      ADD2(I,J)=LLLL-SUBM2*(LLLL/SUBM2)
   87 CONTINUE
      DO 88 I=1,SUBM1
      DO 88 J=1,SUBM1
      IS1=ADD1(I,J)*MMOD(ALPSO)
      SAD1(I,J)=IS1-MMOD*
   88 CONTINUE
      DO 89
      DO 89 J=
      IS2=ADD2(                ALPSO)
      SAD2(I,J)=                ,MMOD)
   89 CONTINUE
      RETURN
      END
C
C
C     SUBROUTINE TO GENETRATE LOOK-UP TABLES FOR POWERS
C     OF ALPH'S AND INVERSE ALPH'S.
C
C
      SUBROUTINE TWLF
      IMPLICIT INTEGER (A-H,O-Z)
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      COMMON/BLK4/ MMOD,SUBM1,SUBM2,NP,MDOT1,MDOT2,INDOT1,
     1INDOT2,SOD,ALPSO,N
      DIMENSION A(128,2),IA(128,2)
      A(1,1)=IA(1,1)=1
      A(1,2)=IA(1,2)=0
      A(2,1)=0 ; A(2,2)=1
C
C
C     CALCULATION OF THE POWERS OF ALPH.
C
C
      DO 31 I=3,N
      A(I,1)=A(I-1,1)*A(2,1)+ALPSO*A(I-1,2)*A(2,2)
      A(I,2)=A(I-1,1)*A(2,2)+A(I-1,2)*A(2,1)
      A(I,1)=A(I,1) MMOD*(A(I,1)/MMOD)
   31 A(I,2)=A(I,2) MMOD*(A(I,2)/MMOD)
      DO 32 I=3,N
      IA(I,1)=A(130-I,1)
   32 IA(I,2)=A(130-I,2)
C
C
C     CALCULATION OF POWERS OF INVERSE OF ALPH.
C
C
      DO 34 I=1,64
      ALP1(1,I)=A(I,1)
      ALP2(1,I)=A(I,2)
      ALPIN1(1,I)=IA(I,1)
      ALPIN2(1,I)=IA(I,2)
   34 CONTINUE
      ISTAG=2
   33 I1=1 ; L=1
      KSTAG=2**(ISTAG-1)
      MSTAG=64/KSTAG
      DO 36 I=1,MSTAG
      DO 35 K=1,KSTAG
      ALP1(ISTAG,I1)=A(L,1)
      ALP2(ISTAG,I1)=A(L,2)
      ALPIN1(ISTAG,I1)=IA(L,1)
      ALPIN2(ISTAG,I1)=IA(L,2)
   35 I1=I1+1
   36 L=L+KSTAG
      ISTAG=ISTAG+1
      IF(ISTAG.LE.7) GO TO 33
      RETURN
      END
C
C
C     SUBROUTINE TO CONVERT THE REAL VALUED INPUT SEQUENCE
C     TO COMPLEX INPUT SEQUENCE FOR AN NTT PROCESSOR.
C
C
      SUBROUTINE OVLAP(IIN,ITS,IKN)
      IMPLICIT INTEGER (A-H,O-Z)
      DIMENSION IIN(128,2),ITS(64),IKN(128)
      DO 181 I1=1,64
      IIN(I1,1)=ITS(I1)
      IIN(I1,2)=IKN(I1)
      DO 182 I2=1,63
  182 ITS(I2)=ITS(I2+1)
      IST(64)=IKN(I1)
  181 CONTINUE
      DO 183 I1=1,64
      IIN(I1+64,1)=ITS(I1)
      IIN(I1+64,2)=IKN(I1+64)
      DO 184 I2=1,63
  184 ITS(I2)=ITS(I2+1)
      I1S(64)=IKN(I1+64)
  183 CONTINUE
      RETURN
      END
```

```
C     ***********************************************************
C     *                                                         *
C     *  MAIN LINE PROGRAM TO SIMULATE AN NTT PROCESSOR         *
C     *  MODULE 191.                                            *
C     *                                                         *
C     ***********************************************************
      IMPLICIT INTEGER (A-H,O-Z)
      EXTERNAL NTT
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      COMMON/BLK2/ IND1(191),IND2(191),INV1(31,31),INV2(31,31)
      COMMON/BLK3/ ADD1(31,31),ADD2(31,31),SUB1(30,30),SUB2(31,31)
     1,REC(30,31)
      COMMON/BLK4/ MMOD,SUBM1,SUBM2,NP,MDOT1,MDOT2,IMDOT1,
     1IMDOT2,SOD,ALPS01,ALPS02,N
      COMMON/BLK5/ DX(1,2),IH(1,2),H(128,2),X(1,2)
      COMMON/BLK6/ STAG2,STAG3,STAG4,STAG5,STAG6,STAG7,STAG8,STAG9
     1,STAG10,STAG11,STAG12,STAG13,STAG14,STAG15,STAG16,STAG17,
     2STAG18,STAG19,STAG20,X1,X2,X3,X4
      DIMENSION BUF11(32,2),BUF12(32,2),BUF13(32,2),BUF14(32,2),
     1BUF21(32,2),BUF22(32,2),BUF23(32,2),BUF24(32,2),AOUT(64),
     2BUF31(32,2),BUF32(32,2),BUF33(32,2),BUF34(32,2),
     3BUF41(32,2),BUF42(32,2),BUF43(32,2),BUF44(32,2)
     4,OUT(128,2),IIN(128,2),ITS(64),IKN(128)
C
C     POWER ON RESET OF LATCHES AND BUFFERS.
C
C
      STAG2=STAG3=STAG4=STAG5=STAG6=STAG7=STAG8=STAG9=STAG10=
     2STAG11=STAG12=STAG13=STAG14=STAG15=STAG16=STAG17=STAG18=
     3STAG19=STAG20=X1=X2=X3=X4=0
      DO 2 I=1,32
      ITS(I)=ITS(I+32)=AOUT(I)=AOUT(I+32)=0
      DO 2 J=1,2
      BUF11(I,J)=BUF12(I,J)=BUF13(I,J)=BUF14(I,J)=0
      BUF21(I,J)=BUF22(I,J)=BUF23(I,J)=BUF24(I,J)=0
      BUF31(I,J)=BUF32(I,J)=BUF33(I,J)=BUF34(I,J)=0
      BUF41(I,J)=BUF42(I,J)=BUF43(I,J)=BUF44(I,J)=0
      IIN(I,J)=IIN(I+32,J+32)=IIN(I+64,J+64)=IIN(I+96,J+96)=0
    2 CONTINUE
C
C     DEFINING THE PARAMETERS FOR LOOK-UP TABLES CALCULATIONS.
C     N-TRNSFORM LENGTH.
C     MMOD-MAIN MODULUS ;
C     ALPS01=REAL PART OF CYCLIC GROUP GENERATOR.
C     ALPS02=IMAGINARY PART OF CYCLIC GROUP GENERATOR
C     MIV-MULTIPLICATIVE INVERSE OF N.
C     NP-MULTIPLICATIVE INVERSE OF MMOD.
C     SUBM1 & SUBM2 = SUB MODULI.
C     IMDOT1 & IMDOT2 = MULTIPLICATIVE INVERSE OF SUBM1 & SUBM2.
C
C
      MMOD=191; SUBM1=30 ; SUBM2=31 ; SOD=SUBM1*SUBM2 ; NP=19
      IMDOT1=1 ; IMDOT2=30 ;MIV=97; ALPS01=66; ALPS02=6; N=128
C
C
C     SUBROUTINE TWLF TO CALCULATE THE THE POWERS OF
C     CYCLIC GROUP GENERATOR ALPHA.
C
C
      CALL TWLF
C
C     SUBROUTINE TO CALCULATE THE VARIOUS LOOK-UP TABLES.
C
C
      CALL LTABLE
C
C
C     SUBROUTINE TO CALCULATE THE FILTER RESPOSE.
C
C
      CALL FILTER (H)
C
C
C     MULTILPY THE FILTER COEFFIENTS BY MULTIPLICATIVE INVERSE
C     OF TRANSFOM LENGTH N.
C
C
      DO 997 I=1,N
      DO 997 J=1,2
      H(I,J)=MIV*H(I,J)
  997 H(I,J)=H(I,J)-MMOD*(H(I,J)/MMOD)
C
C
C     SUBROUTINE TO GET INPUT SEQUENCE ;
C
C
      CALL INFILE(IKN)
C
C
      DM4=1 ; DM2=0 ; IMUX1=2 ; DM3=1 ; IMUX22=1
      DO 100 K=1,8
C
C
C     SUBROUTINE TO ARRANGE THE INPUT SEQUENCE IN THE FORM
C     REQUIRED FOR OVER-LAP SAVE METHOD.
C
C
      CALL OVLAP(IIN,ITS,IKN)
      IF(DM4.NE.1.AND.DM2.EQ.1) GO TO 3
      CALL INOUT(BUF41,BUF42,BUF43,BUF44,IIN,OUT)
      OPMUX=2 ; IMUX22=1 ; IMUX24=0
      GO TO 4
    3 CALL INOUT(BUF21,BUF22,BUF23,BUF24,IIN,OUT)
      OPMUX=1 ; IMUX24=1 ; IMUX22=0
    4 IF(IMUX1.NE.2) GO TO 5
      MMI=65 ; MMF=128
      CALL MULTLY(BUF31,BUF32,BUF33,BUF34,MMI,MMF,BUF11,BUF12,
```

```
     1BUF13,BUF14)
      IMUX34=1 ; IMUX21=0
      GO TO 6
 5    MM1=65 ; MMF=128
      CALL MULTY(BUF11,BUF12,BUF13,BUF14,MM1,MMF,BUF31,BUF32,
     1BUF33,BUF34)
      IMUX21=1 ; IMUX34=0
 6    IF(IMUX22.NE.1.AND.IMUX24.EQ.1) GO TO 7
      STAG=1 ; INV=0 ; DM1=1
      CALL STAG1(BUF21,BUF22,BUF23,BUF24,BUF11,BUF12,BUF13,BUF14,
     1NTT,STAG,INV)
 8    STAG=STAG+1 ; DM2=2
      CALL ASTAG(BUF11,BUF12,BUF13,BUF14,BUF21,BUF22,BUF23,BUF24,
     1NTT,STAG,INV)
      STAG=STAG+1 ; DM1=1
      CALL ASTAG(BUF21,BUF22,BUF23,BUF24,BUF11,BUF12,BUF13,BUF14,
     1NTT,STAG,INV)
      IF(STAG.NE.7) GO TO 8
      IMUX1=1 ; DM1=2
      GO TO 10
 7    STAG=1 ; INV=0 ; DM3=2
      CALL STAG1(BUF41,BUF42,BUF43,BUF44,BUF31,BUF32,BUF33,BUF34,
     1NTT,STAG,INV)
 9    STAG=STAG+1 ; DM4=2
      CALL ASTAG(BUF31,BUF32,BUF33,BUF34,BUF41,BUF42,BUF43,BUF44,
     1NTT,STAG,INV)
      STAG=STAG+1 ; DM3=2
      CALL ASTAG(BUF41,BUF42,BUF43,BUF44,BUF31,BUF32,BUF33,BUF34,
     1NTT,STAG,INV)
      IF(STAG.NE.7) GO TO 9
      IMUX1=2 ; DM3=1
10    IF(IMUX1.NE.1) GO TO 15
      MM1=1 ; MMF=64
      CALL MULTY(BUF11,BUF12,BUF13,BUF14,MM1,MMF,BUF31,BUF32,
     1BUF33,BUF34)
      IMUX1=1
      GO TO 16
15    MM1=1 ; MMF=64
      CALL MULTY(BUF31,BUF32,BUF33,BUF34,MM1,MMF,BUF11,BUF12,
     1BUF13,BUF14)
      IMUX1=2
16    IF(IMUX23.NE.1.AND.IMUX21.EQ.1) GO TO 11
      STAG=1 ; INV=1 ; DM2=2
      CALL STAG1(BUF31,BUF32,BUF33,BUF34,BUF21,BUF22,BUF23,BUF24,
     1NTT,STAG,INV)
12    STAG=STAG+1 ; DM3=2
      CALL ASTAG(BUF21,BUF22,BUF23,BUF24,BUF31,BUF32,BUF33,BUF34,
     1NTT,STAG,INV)
      STAG=STAG+1 ; DM2=2
      CALL ASTAG(BUF31,BUF32,BUF33,BUF34,BUF21,BUF22,BUF23,BUF24,
     1NTT,STAG,INV)
      IF(STAG.NE.7) GO TO 12
      DM2=1 ; DM4=0
      GO TO 16
11    STAG=1 ; INV=1 ; DM4=2
      CALL STAG1(BUF11,BUF12,BUF13,BUF14,BUF41,BUF42,BUF43,BUF44,
     1NTT,STAG,INV)
14    STAG=STAG+1 ; DM1=1
      CALL ASTAG(BUF41,BUF42,BUF43,BUF44,BUF11,BUF12,BUF13,BUF14,
     1NTT,STAG,INV)
      STAG=STAG+1 ; DM4=2
      CALL ASTAG(BUF11,BUF12,BUF13,BUF14,BUF41,BUF42,BUF43,BUF44,
     1NTT,STAG,INV)
      IF(STAG.NE.7) GO TO 14
      DM4=1 ; DM2=0
16    PRINT 777,(OUT(I,1),I=65,N)
777   FORMAT(3014)
      PRINT 778,(AOUT(I),I=1,64)
778   FORMAT(3014)
      DO 541 I4=65,N
541   AOUT(I)=OUT(I,2)
100   CONTINUE
      STOP
      END
C
C
C     SUBROUTINE TO SIMULATE NECESSARY BUFFER INTERCONNECTIONS
C     FOR THE FIRST STAGE.
C
C
      SUBROUTINE STAG1(X21,X22,X23,X24,X31,X32,X33,X34,NTT,
     1STAG,INV)
      IMPLICIT INTEGER (A-H,O-Z)
      EXTERNAL NTT
      DIMENSION X21(32,2),X22(32,2),X23(32,2),X24(32,2),
     2X31(32,2),X32(32,2),X33(32,2),X34(32,2)
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      DO 10 I=1,64
      A11=X21(1,1) ; A12=X21(1,2) ; B11=X23(1,1) ; B12=X23(1,2)
      DO 1 K1=1,31
      DO 1 J=1,2
1     X21(K1,J)=X21(K1+1,J)
      X21(32,1)=X22(1,1) ; X21(32,2)=X22(1,2)
      DO 2 K1=1,31
      DO 2 J=1,2
2     X22(K1,J)=X22(K1+1,J)
      DO 3 K1=1,31
      DO 3 J=1,2
3     X23(K1,J)=X23(K1+1,J)
      X23(32,1)=X24(1,1) ; X23(32,2)=X24(1,2)
      DO 4 K1=1,31
      DO 4 J=1,2
4     X24(K1,J)=X24(K1+1,J)
      CALL NTT(STAG,I,A11,A12,B11,B12,B01,B02,B03,B04,INV)
      DO 5 K1=1,31
      DO 5 J=1,2
5     X31(K1,J)=X31(K1+1,J)
      X31(32,1)=X32(1,1) ; X31(32,2)=X32(1,2)
      DO 6 K1=1,31
      DO 6 J=1,2
```

```
      X32(K1,J)=X32(K1+1,J)
      DO 7 K1=1,31
      DO 7 J=1,2
    7 X33(K1,J)=X33(K1+1,J)
      X33(32,1)=X34(1,1) ; X33(32,2)=X34(1,2)
      DO 8 K1=1,31
      DO 8 J=1,2
    8 X34(K1,J)=X34(K1+1,J)
      X32(32,1)=B01 ; X32(32,2)=B02
      X34(32,1)=B03 ; X34(32,2)=B04
   10 CONTINUE
      RETURN
      END


C
C     SUBROUTINE TO SIMULATE THE BUFFER INTERCONNECTIONS
C     FOR THE STAGES J TO Z.
C
C
      SUBROUTINE BSTAG(X21,X22,X23,X24,X31,X32,X33,X
     134,NI1,STAG,INV)
      IMPLICIT INTEGER (A-H,O-Z)
      EXTERNAL NI1
      DIMENSION X21(32,2),X22(32,2),X23(32,2),X24(32,2),
     1X31(32,2),X32(32,2),X33(32,2),X34(32,2)
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      POS=1
      ISTAG=2**(STAG-2)
      MX=64/ISTAG/2
      DO 10 I=1,MX
      DO 2 K1=1,ISTAG
      A11=X21(1,1) ; A12=X21(1,2) ; B11=X22(1,1) ; B12=X22(1,2)
      DO 1 K2=1,31
      DO 1 J=1,2
      X21(K2,J)=X21(K2+1,J)
    1 X22(K2,J)=X22(K2+1,J)
      CALL NI1(STAG,POS,A11,A12,B11,B12,B01,B02,B03,B04,INV)
      POS=POS+1
      DO 3 K2=1,31
      DO 3 J=1,2
    3 X31(K2,J)=X31(K2+1,J)
      X31(32,1)=X32(1,1) ; X31(32,2)=X32(1,2)
      DO 4 K2=1,31
      DO 4 J=1,2
    4 X32(K2,J)=X32(K2+1,J)
      DO 5 K2=1,31
      DO 5 J=1,2
    5 X33(K2,J)=X33(K2+1,J)
      X33(32,1)=X34(1,1) ; X33(32,2)=X34(1,2)
      DO 6 K2=1,31
      DO 6 J=1,2
    6 X34(K2,J)=X34(K2+1,J)
      X32(32,1)=B01 ; X32(32,2)=B02
      X34(32,1)=B03 ;X34(32,2)=B04
      CONTINUE
      DO 8 K1=1,ISTAG
      A11=X23(1,1) ; A12=X23(1,2) ; B11=X24(1,1) ; B12=X24(1,2)
      DO 9 K2=1,31
      DO 9 J=1,2
      X23(K2,J)=X23(K2+1,J)
    9 X24(K2,J)=X24(K2+1,J)
      CALL NI1(STAG,POS,A11,A12,B11,B12,B01,B02,B03,B04,INV)
      POS=POS+1
      DO 12 K2=1,31
      DO 12 J=1,2
   12 X31(K2,J)=X31(K2+1,J)
      X31(32,1)=X32(1,1) ; X31(32,2)=X32(1,2)
      DO 13 K2=1,31
      DO 13 J=1,2
   13 X32(K2,J)=X32(K2+1,J)
      DO 14 K2=1,31
      DO 14 J=1,2
   14 X33(K2,J)=X33(K2+1,J)
      X33(32,1)=X34(1,1) ; X33(32,2)=X34(1,2)
      DO 15 K2=1,31
      DO 15 J=1,2
   15 X34(K2,J)=X34(K2+1,J)
      X32(32,1)=B01 ; X32(32,2)=B02
      X34(32,1)=B03 ;X34(32,2)=B04
    8 CONTINUE
   10 CONTINUE
      RETURN
      END
C
C
C     SUBROUTINE TO PERFORM THE BUTTERFLY CALCULATIONS.
C
C
      SUBROUTINE NI1(STAG,POS,A11,A12,B11,B12,B01,B02,B04,B04,
     1INV)
      IMPLICIT INTEGER (A-H,O-Z)
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      COMMON/BLK4/ MMOD,SUBM1,SUBM2,NP,MOD11,MOD12,IMOD11,
     1IMOD12,SOD,ALPS01,ALPS02,N
      B01=A11+B11 ; B02=A12+B12
      B01=B01-MMOD*(B01/MMOD)
      B02=B02-MMOD*(B02/MMOD)
      E1=A11-B11 ; E2=A12-B12
      IF(E1.LT.0) E1=E1+MMOD
      IF(E2.LT.0) E2=E2+MMOD
      IF(INV.EQ.1) GO TO 220
      B03=E1*ALP1(STAG,POS)-E2*ALP2(STAG,POS)
      B03=B03-MMOD*(B03/MMOD)
      IF(B03.LT.0) B03=B03+MMOD
      B04=E1*ALP2(STAG,POS)+E2*ALP1(STAG,POS)
      B04=B04-MMOD*(B04/MMOD)
      GO TO 230
  220 B03=E1*ALPIN1(STAG,POS)-E2*ALPIN2(STAG,POS)
      B03=B03-MMOD*(B03/MMOD)
      IF(B03.LT.0) B03=B03+MMOD
```

```
      B04=FLOAT(IN2(STAG,POS))*FLOAT(IN1(STAG,POS))
      B04=B04-AMOD(B04/AMOD)
  250 RETURN
      END
C
C
C     SUBROUTINE TO SIMULATE THE BUFFER CONNECTIONS
C     INPUT AND OUTPUT OPERATIONS.
C
C
      SUBROUTINE INOUT(X21,X22,X23,X24,IIN,IOUT)
      IMPLICIT INTEGER (A-H,O-Z)
      DIMENSION X21(32,2),X22(32,2),X23(32,2),X24(32,2)
      DIMENSION IIN(128,2),IOUT(128,2),IAN(128)
      DO 5 I=1,N
      OUT(I,1)=X21(1,1) ; OUT(I,2)=X21(1,2)
      DO 4 J=1,2
      DO 1 K1=1,31
    1 X21(K1,J)=X21(K1+1,J)
      X21(32,J)=X22(1,J)
      DO 2 K2=1,31
    2 X22(K2,J)=X22(K2+1,J)
      X22(32,J)=X23(1,J)
      DO 3 K3=1,31
    3 X23(K3,J)=X23(K3+1,J)
      X23(32,J)=X24(1,J)
      DO 4 K4=1,31
    4 X24(K4,J)=X24(K4+1,J)
      X24(32,1)=IIN(I,1) ; X24(32,2)=IIN(I,2)
    5 CONTINUE
      RETURN ; END
C
C
C     SUBROUTINE TO SIMULATE THE STRUCTURE OF AN EXTERNAL SERIAL
C     COMPLEX MULTIPLIER.
C
C
      SUBROUTINE MULTL(X21,X22,X23,X24,MM1,MMF,X31,X32,X33,X34)
      IMPLICIT INTEGER (A-H,O-Z)
      DIMENSION X21(32,2),X22(32,2),X23(32,2),X24(32,2),X31(32,2),
     1X32(32,2),X33(32,2),X34(32,2)
      COMMON/BLK5/ OX(1,2),IH(128,2),X(1,2)
      COMMON/BLK2/ IND1(191),IND2(191),INV1(31,31),INV2(31,31)
      COMMON/BLK3/ ADD1(31,31),ADD2(31,31),SUB1(30,30),SUB2(31,31)
     1,REC1(30,31)
      COMMON/BLK4/ MMOD,SUBM1,SUBM2,NP,MDOT1,MDOT2,IMDOT1,
     1IMDOT2,SUB,ALPSU1,ALPSU2,N
      COMMON/BLK5/ STAG2,STAG3,STAG4,STAG5,STAG6,STAG7,STAG8,STAG9
     1,STAG10,STAG11,STAG12,STAG13,STAG14,STAG15,STAG16,STAG17,
     2STAG18,STAG19,STAG20,X1,X2,X3,X4
      LI=0
      INV1(31,31)=INV2(31,31)=0 ; ADD1(31,31)=30; ADD1(1,1)=0
      K1=1
      DO 10 I=MM1,MMF
      X(1,1)=X21(1,1) ; X(1,2)=X21(1,2)
      IH(1,1)=H(1,1) ; IH(1,2)=H(1,2)
      K1=1 ;K2=2
    2 STAG19=REC1(STAG17+1,STAG18+1)
      IF(LI.EQ.0) GO TO 15
      STAG18=SUB2(STAG8+1,STAG16+1)
      STAG17=SUB1(STAG7+1,STAG15+1)
      GO TO 16
   15 STAG18=ADD2(STAG8+1,STAG16+1)
      STAG17=ADD1(STAG7+1,STAG15+1)
   16 STAG16=INV2(STAG13+1,STAG14+1)
      STAG15=INV1(STAG13+1,STAG14+1)
      STAG14=ADD2(STAG11+1,STAG12+1)
      STAG13=ADD1(STAG9+1,STAG10+1)
      IF(STAG13.EQ.30) STAG14=30
      IF(X4.EQ.0.OR.X3.EQ.0) GO TO 5
      STAG12=IND2(X4)
      STAG11=IND2(X3)
      STAG10=IND1(X4)
      STAG9=IND1(X3)
      GO TO 9
    5 STAG9=STAG10=30 ; STAG11=STAG12=0
    9 X4=X(1,1)
      X3=IH(1,K1)
      OX(1,K1)=STAG19
      K1=K1+1
      LI=LI+1
      IF(K1.GT.2) K1=1
      STAG8=INV2(STAG5+1,STAG6+1)
      STAG7=INV1(STAG5+1,STAG6+1)
      STAG6=ADD2(STAG3+1,STAG4+1)
      STAG5=ADD1(STAG20+1,STAG2+1)
      IF(STAG5.EQ.30) STAG6=30
      IF(X2.EQ.0.OR.X1.EQ.0) GO TO 7
      STAG4=IND2(X2)
      STAG3=IND2(X1)
      STAG2=IND1(X2)
      STAG20=IND1(X1)
      GO TO 6
    7 STAG20=STAG2=30 ; STAG3=STAG4=0
    6 X2=X(1,2)
      X1=IH(1,K2)
      LI=LI+1
      K2=K2-1
      IF(K2.LT.1) K2=1
      IF(LI.LT.4) GO TO 2
      LI=0
      IF(KL.GT.3.OR.I.GE.65) GO TO 31
      DO 71 K1=1,31
      DO 71 J=1,2
   71 X31(K1,J)=X31(K1+1,J)
      X31(32,1)=X32(1,1) ; X31(32,2)=X32(1,2)
      DO 72 K1=1,31
      DO 72 J=1,2
   72 X32(K1,J)=X32(K1+1,J)
      X32(32,1)=X33(1,1) ; X32(32,2)=X33(1,2)
```

```
      DO 73 K1=1,31
      DO 73 J=1,2
   73 X33(K1,J)=X33(K1+1,J)
      X33(32,1)=X33(1,1) + X33(32,2)-X34(1,2)
      DO 74 K1=1,31
      DO 74 J=1,2
   74 X34(K1,J)=X34(K1+1,J)
      X34(32,1)=DX(1,1) + X34(32,2)-DX(1,2)
   31 DO 1 K1=1,31
      DO 1 J=1,2
    1 X21(K1,J)=X21(K1+1,J)
      X21(32,1)=X22(1,1) + X21(32,2)=X22(1,2)
      DO 12 K1=1,31
      DO 12 J=1,2
   12 X22(K1,J)=X22(K1+1,J)
      X22(32,1)=X23(1,1) + X22(32,2)=X23(1,2)
      DO 3 K1=1,31
      DO 3 J=1,2
    3 X23(K1,J)=X23(K1+1,J)
      X23(32,1)=X24(1,1) + X23(32,2)=X24(1,2)
      DO 4 K1=1,31
      DO 4 J=1,2
    4 X24(K1,J)=X24(K1+1,J)
      IF(K1.EQ.3.AND.I.LT.65) GO TO 32
      X24(32,1)=DX(1,1) + X24(32,2)=DX(1,2)
   32 K1=K1+1
   10 CONTINUE
      RETURN
      END
C
C
C     SUBROUTINE TO GENERATE NECESSARY LOOK-UP TABLES.
C
C
      SUBROUTINE LTABLE
      IMPLICIT INTEGER (A-H,O-Z)
      COMMON/BLK2/ IND1(191),IND2(191),INV1(31,311),INV2(31,31)
      COMMON/BLK3/ ADD1(31,31),ADD2(31,31),SUB1(30,30),SUB2(31,31)
     1,RECT(30,31)
      DIMENSION T1(30),T2(31),INVD1(191),INVD2(191),IN(191)
      COMMON/BLK4/ MMOD, SUBM1,SUBM2,NP,MDOT1,MDOT2,IMDOT1,
     1IMDOT2,SOD,ALPSO1,ALPSO2,N
C
C
C     CALCULATION OF RECONSTRUCTION TABLE FROM MODS 30 AND 31 .
C
C
      DO 71 I=1,SUBM1
      K=I-1
      T1(I)=K*IMDOT1-SUBM1*(K*IMDOT1/SUBM1)
   71 T1(I)=T1(I)*SUBM2
      DO 72 I=1,SUBM2
      K=I-1
      T2(I)=K*IMDOT2-SUBM2*(K*IMDOT2/SUBM2)
   72 T2(I)=T2(I)*SUBM1
      DO 73 J=1,SUBM2
      DO 73 I=1,SUBM1
      RECT(I,J)=T1(I)+T2(J)
      RECT(I,J)=RECT(I,J)-SOD*(RECT(I,J)/SOD)
      RECT(I,J)=RECT(I,J)-MMOD*(RECT(I,J)/MMOD)
   73 CONTINUE
C
C
C     CALCULATION OF INDEX AND INVERSE INDEX LOOK-UP
C     TABLES AT MOD 30 & 31.
C
      IND1(1)=0
      IND1(NP)=0
      IN(1)=1
      IN(191)=0
      INVD1(191)=0
      INVD2(191)=0
      MMOD1=MMOD-1
      DO 77 I=2,MMOD1
      IND1=IN(I-1)
      IND1=IND1*NP-MMOD*(IND1*NP/MMOD)
      IN(I)=IND1
   77 IND1(IND1)=I-1
      DO 78 I=1,MMOD1
      IT1=IND1(I)
      IT2=IN(I)
      IND1(I)=IT1-SUBM1*(IT1/SUBM1)
      IND2(I)=IT1-SUBM2*(IT1/SUBM2)
      INVD1(I)=IT2-SUBM1*(IT2/SUBM1)
      INVD2(I)=IT2-SUBM2*(IT2/SUBM2)
   78 CONTINUE
      DO 81 I=1,SUBM1
      DO 81 J=1,SUBM2
      IP=((I-1)*SUBM2*IMDOT1+(J-1)*SUBM1*IMDOT2)
      IF(IP.LT.SOD) GO TO 82
      L=IP/SOD  ; IP=IP-L*SOD
   82 IF(IP.LT.MMOD1) GO TO 83
      LM=IP/MMOD1 ; IP=IP-LM*MMOD1
      IF(IP.GE.450) IP=1
   83 IP1=INVD1(IP+1)
      IP2=INVD2(IP+1)
      INV1(I,J)=IP1
   81 INV2(I,J)=IP2
C
C
C     CALCULATION OF ADDITION AND SUBTRACTION LOOK TABLE FOR MOD
C     30 AND 31 .
C
C
      DO 86 I=1,SUBM1
      DO 86 J=1,SUBM1
      II=I-1
      JJ=J-1
      LLL=II+JJ
      ADD1(I,J)=LLL-SUBM1*(LLL/SUBM1)
```

```
      KK II=J,J
      IF(KK.LT.0) KK=KK+SUBM1
      SUB1(I,J)=KK
   86 CONTINUE
      DO 87 I=1,SUBM2
      DO 87 J=1,SUBM2
      III=I+1
      JJJ=J+1
      IF(III.EQ.SUBM2+1) III=SUBM2*(I/SUBM2)
      SUB2(I,J)=III+SUBM2*(JJJ/SUBM2)
      KKK=III+JJJ
      IF(KKK.LT.0) KKK=KKK+SUBM2
      SUB3(I,J)=KKK
   87 CONTINUE
      RETURN
      END
C
C
C     SUBROUTINE TO GENERATE LOOK-UP TABLES FOR POWERS
C     OF ALPH'S AND INVERSE ALPH'S.
C
C
      SUBROUTINE TWLF
      IMPLICIT INTEGER (A-H,O-Z)
      COMMON/BLK1/ ALP1(7,64),ALP2(7,64),ALPIN1(7,64),ALPIN2(7,64)
      COMMON/BLK2/ MMOD,SUBM1,SUBM2,NP,MDOT1,MDOT2,IMDOT1,
     1IMDOT2,SUB,ALPS01,ALPS02,N
      DIMENSION A(128,2),IA(128,2)
      A(1,1)=IA(1,1)=1
      A(1,2)=IA(1,2)=0
      A(2,1)=ALPS01 ; A(2,2)=ALPS02
C
C
C     CALCULATION OF THE POWERS OF ALPH.
C
C
      DO 31 I=3,N
      A(I,1)=A(I-1,1)*A(2,1)-A(I-1,2)*A(2,2)
      IAA=A(I,1) ; A(I,1)=MOD(IAA,MMOD)
      IF(A(I,1).LT.0) A(I,1)=A(I,1)+MMOD
      A(I,2)=A(I-1,1)*A(2,2)+A(I-1,2)*A(2,1)
   31 A(I,2)=A(I,2)-MMOD*(A(I,2)/MMOD)
      DO 32 I=2,N
      IA(I,1)=A(130-I,1)
   32 IA(I,2)=A(130-I,2)
C
C
C     CALCULATION OF POWERS OF INVERSE OF ALPH.
C
C
      DO 34 I=1,64
      ALP1(1,I)=A(I,1)
      ALP2(1,I)=A(I,2)
      ALPIN1(1,I)=IA(I,1)
      ALPIN2(1,I)=IA(I,2)
   34 CONTINUE
      ISTAG=2
   33 II=1
      L=1
      KSTAG=2**(ISTAG-1)
      MSTAG=64/KSTAG
      DO 36 I=1,MSTAG
      DO 35 K=1,KSTAG
      ALP1(ISTAG,II)=A(L,1)
      ALP2(ISTAG,II)=A(L,2)
      ALPIN1(ISTAG,II)=IA(L,1)
      ALPIN2(ISTAG,II)=IA(L,2)
   35 II=II+1
   36 L=L+KSTAG
      ISTAG=ISTAG+1
      IF(ISTAG.LE.7) GO TO 33
      RETURN
      END
C
C
C     SUBROUTINE TO CONVERT THE REAL VALUED INPUT SEQUENCE
C     TO COMPLEX INPUT SEQUENCE FOR AN NTT PROCESSOR.
C
C
      SUBROUTINE OVLAP(IIN,ITS,IKN)
      IMPLICIT INTEGER (A-H,O-Z)
      DIMENSION IIN(128,2),ITS(64),IKN(128)
      DO 181 II=1,64
      IIN(II,1)=ITS(II)
      IIN(II,2)=IKN(II)
      DO 182 I2=1,63
  182 ITS(I2)=ITS(I2+1)
      ITS(64)=IKN(II)
  181 CONTINUE
      DO 183 II=1,64
      IIN(II+64,1)=ITS(II)
      IIN(II+64,2)=IKN(II+64)
      DO 184 I2=1,63
  184 ITS(I2)=ITS(I2+1)
      ITS(64)=IKN(II+64)
  183 CONTINUE
      RETURN
      END
```

APPENDIX B

PROGRAMS TO GENERATE LOOK-UP TABLES

FOR 2708 EPROMS ON INTEL 220 SYSTEM

```
;       ****************************************
;       PROGRAME TO GENERATE INDEX AND
;       INVERSE INDEX LOOK-UP TABLE AT MODULI
;       30 & 31.
;       IND30 & IND31 REFERS TO INDEX TABLE AT
;       MODULI 30 & 31 RESPECTIVELY.
;       INV30 & INV31 REFERS TO INVERSE INDEX
;       TABLE AT MODULI 30 & 31 RESPECTIVELY.
;       SUBROUTINE REQUIRED ARE MULT & RECT.
;       ****************************************
;INVERSE INDEX TABLE FOR MOD 193
        CSEG
START:: MVI A,00H
        MVI C,00H
        MVI D,00H
        MVI B,5 ;LOAD GENERATOR IN B
        MVI E,1
        LXI H,5000H
        MOV M,E
ST1:    PUSH H
        CALL MULT ;CALL MULTIPLICATION SUBROUTIN
DONE:   MOV A,E
        CPI 193
        JNC LOC1
        MOV A,D
        CPI 0
        JZ LOC2
LOC1:   LXI H,-193 ;MODULO 193 REDUCTION
        DAD D
        XCHG
        JMP DONE
LOC2:   POP H
        INX H
        MOV M,E
        MOV A,L
        CPI 192
        JNZ ST1
;INDEX LOOKUP TABLE FOR MOD 193
        LXI H,5000H
        MVI B,0
LOC4:   LXI D,6000H
        MOV A,M
        ADD E
        MOV E,A
        MOV A,B
        STAX D
        INX H
        INR B
        MOV A,L
        CPI 192
        JNZ LOC4
; INDEX LOOKUP TABLE FOR MOD 30
        LXI D,5100H
        MVI A,31
        STAX D
        INX D
        LXI H,6001H
LOC7:   MOV A,M
LOC6:   CPI 30
        JC LOC5
        SUI 30  ;MODULO 30 REDUCTION
        JMP LOC6
LOC5:   STAX D
        INX H
        INX D
        MOV A,L
        CPI 193
        JNZ LOC7
;INDEX LOOKUP TABLE FOR MOD 31
        LXI D,5200H
        MVI A,31
        STAX D
        INX D
        LXI H,6001H
LOC9:   MOV A,M
LOC10:  CPI 31
        JC LOC11
        SUI 31  ;MODULO 31 REDUCTION
        JMP LOC10
LOC11:  STAX D
        INX H
        INX D
        MOV A,L
        CPI 193
        JNZ LOC9
;INVERSE INDEX TABLE FOR MOD 193 AT MOD 30
        LXI H,5000H
        LXI B,5400H
LOC14:  MOV A,M
LOC13:  CPI 30
        JC LOC12
        SUI 30  ;MODULO 30 REDUCTION
        JMP LOC13
LOC12:  STAX B
        INX H
        INX B
        MOV A,L
        CPI 193
        JNZ LOC14
;INVERSE INDEX TABLE FOR MOD 193 AT MOD 31
        LXI H,5600H
        LXI B,5500H
LOC17:  MOV A,M
LOC16:  CPI 31
        JC LOC15
        SUI 31  ;MODULO 31 REDUCTION
        JMP LOC16
LOC15:  STAX B
        INX H
```

```
;       ****************************************
;       PROGRAME TO COMPUTE ADDITION LOOK-UP
;       TABLE FOR MODULUS 30 FOR R*X*Y
;        TABLE IS STORED FROM 7400H TO 77FFH.
;       ****************************************
        CSEG
START:  MVI A,00H
        LXI H,7400H ; STARTING ADDRESS OF  TABLE
        MVI D,00H
        MVI B,00H
LP2:    MVI C,00H
LP1:    MOV A,C
        ADD B
;       FOR MODIFIED ADD TABLE
;       TO IMPLEMENT R*X*Y.
        ADI 3
        CPI 30
        JC LOC1
        SUI 30  ;MODULO 30 REDUCTION.
LOC1:   MOV D,A
        MOV A,C
        CPI 30
        JC LOC5
        MVI D,31
LOC5:   MOV M,D
        INR C   ;INCREMENT THE ADDEND.
        INX H
        MOV A,C
        CPI 32
        JNZ LP1
        INR B   ;INCREMENT THE ADDER.
        MOV A,B
        CPI 30
        JNZ LP2
        MVI E,00H
LOC7:   INR E
        MVI M,31
        INX H
        MOV A,E
        CPI 63
        JMP ADRSSH
        END START
```

```
;       ****************************************
;       PROGRAME TO COMPUTE ADDITION LOOK-UP TAB
;       FOR MODULUS 31 FOR R*X*Y.
;       TABLE IS STORE FROM 7400H TO 77FFH
;       ****************************************
        CSEG
START:  MVI A,00H
        LXI H,7400H ;STARTING ADDRESS OF  TABLE.
        MVI D,00H
        MVI B,00H
LP2:    MVI C,00H
LP1:    MOV A,C
        ADD B
;       FOR MODIFIED ADD TABLE
;       TO IMPLEMENT R*X*Y.
        ADI 3
        CPI 31
        JC LOC1
        SUI 31  ;MODULO 31 REDUCTION.
LOC1:   MOV D,A
        MOV A,C
        CPI 31
        JC LOC5
        MVI D,31
LOC5:   MOV M,D
        INR C   ;INCREMENT THE ADDEND.
        INX H
        MOV A,C
        CPI 32
        JNZ LP1
        INR B   ;INCREMENT THE ADDER.
        MOV A,B
        CPI 31
        JNZ LP2
        MVI E,00H
LOC7:   INR E
        MVI M,31
        INX H
        MOV A,E
        CPI 31
        JNZ LOC7
        JMP ADRSSH
        END START
```

```
1  ;       ****************************************
2  ;       PROGRAM TO GENERATE RECONSTRUCTION TABLE
3  ;       RECT(3H,31). TABLE IS STORED AT LOCATION
4  ;       7400H TO 77FFH.
5  ;       ****************************************
6  ;LOOKUP TABLE FOR RECONSTRUCTION OF MOD 30 & 31
7          CSEG
8  START:  MVI A,00H
9          MVI B,00H
10         MVI C,00H
11         MVI D,00H
12         LXI H,7000H
13 LOK1:   MVI E,31
14         PUSH B
15         CALL MULT ;CALL TO MULTIPLICATION ROUTIN
16         MOV M,E
17         INX H
18         MOV M,D
19         POP B
20         INX H
21         INR B
22         MOV A,B
23         CPI 30
24         JNZ LOK1
25         MVI B,00H
26         MVI A,00H
27         MVI C,00H
28         MVI D,00H
29         LXI H,7100H
30 LOK3:   MVI E,30
31         PUSH B
32         PUSH H
33         CALL MULT ;CALL TO MULTIPLICATION ROUTIN
34 LOK4:   MOV A,E
35         CPI 31
36         JNC LOK5
37         MOV A,D
38         CPI 0
39         JZ LOK6
40 LOK5:   LXI H,-31 ;MODULO 31 REDUCTION
41         DAD D
42         XCHG
43         JMP LOK4
44 LOK6:   POP H
45         POP B
46         MOV M,E
47         INX H
48         INR B
49         MOV A,B
50         CPI 31
51         JNZ LOK3
52         MVI A,00H
53         LXI H,7100H
54         MVI C,00H
55         LXI B,7200H
56 LOK7:   PUSH B
57         MVI E,30
58         MOV D,M
59         CALL MULT ;CALL TO MULTIPLICATION ROUTIN
60         POP B
61         MOV A,E
62         STAX B
63         INX B
64         MOV A,D
65         STAX B
66         INX H
67         INX B
68         MOV A,L
69         CPI 31
70         JNZ LOK7
71         LXI D,8000H
72         LXI B,7200H
73 LOK9:   LXI H,7000H
74 LOK8:   XRA A
75         PUSH B
76         LDAX B
77         ADC M
78         STAX D
79         INX B
80         INX H
81         INX D
82         LDAX B
83         ADC M
84         STAX D
85         POP B
86         INX H
87         INX D
88         MOV A,L
89         CPI 60
90         JNZ LOK8
91         INX B
92         INX B
93         MOV A,C
94         CPI 62
95         JNZ LOK9
96         LXI B,3720
97         LXI H,8000H
98 ADS3:   PUSH H
99         MOV E,M
100        INX H
101        MOV D,M
102 ADS2:  MOV A,D
103        CPI 3
104        JC ADS1
105        JNZ ADS4
106        MOV A,E
107        CPI 162
108        JC ADS1
109 ADS4:  LXI H,-930
```

```
110             DAD D
111             XCHG
112             JMP ADS2
113 ADS1:       POP H
114             MOV M,E
115             DCX B
116             INX H
117             MOV M,D
118             DCX B
119             INX H
120             MOV A,B
121             CPI 0
122             JNZ ADS3
123             MOV A,C
124             CPI 0 /
125             JNZ ADS3
126             LXI D,931
127             LXI B,8000H
128             LXI H,8000H
129 LK1:        PUSH D
130             PUSH H
131             MOV E,M
132             INX H
133             MOV D,M
134             XOR A,D
135             CPI 1
136             JC LK2
137             JNZ LK9
138             MOV A,E
139             CPI 209
140             JC LK2
141 LK9:        LXI H,930 ;MODULO 30*31 REDUCTION
142             XRA A
143             MOV A,L
144             SBB E
145             MOV E,A
146             MOV A,H
147             SBB D
148             MOV D,A
149 LK3:        MOV A,E
150             CPI 193
151             JNC LK7
152             JZ LK7
153             MOV A,D
154             CPI 0
155             JZ LK8
156 LK7:        LXI H,-193 ;MODULO 193 REDUCTION
157             DAD D
158             XCHG
159             JMP LK3
160 LK8:        MVI A,193
161             SUB E
162             CPI 193
163             JNZ LK10
164             SUI 193
```

```
165 LK10:       MOV E,A
166             JMP LK4
167 LK2:        MOV A,E
168             CPI 193
169             JNC LK5
170             JZ LK5
171             MOV A,D
172             CPI 0
173             JZ LK4
174 LK5:        LXI H,-193 ;MODULO 193 REDUCTION
175             DAD D
176             XCHG
177             JMP LK2
178 LK4:        POP H
179             MOV A,E
180             STAX B
181             INX H
182             INX H
183             INX B
184             POP D
185             DCX D
186             MOV A,D
187             CPI 0
188             JNZ LK1
189             MOV A,E
190             CPI 0
191             JNZ LK1
192             LXI H,7400H
193 ;           STARTING LOCATION OF REC1(30,31) TABLE
194             LXI D,8000H
195             MVI B,-1
196 ADRS4:      MVI C,00
197             INR B
198 ADRS3:      MOV A,C
199             CPI 30
200             JZ ADRS1
201             LDAX D
202             MOV M,A
203             INX H
204             INX D
205             INR C
206             JMP ADRS3
207 ADRS1:      MVI M,0FFH
208             INX H
209             MVI M,0FFH
210             INX H
211             MOV A,B
212             CPI 31
213             JNZ ADRS4
214             MVI D,0
215             JMP E2
216 ADRS5:      MVI M,0FFH
217             INX H
218             INR D
219             CPI 85
```

```
220             JNZ ADRSS
221             JMP E2
222 ;           SUBROUTINE TO MULTIPLY TWO NUMBERS
223 MULT:       MVI D,00H
224             MVI C,9
225 MULT0:      MOV A,E
226             RAR
227             MOV E,A
228             DCR C
229             RZ
230             MOV A,D
231             JNC MULTI
232             ADD B
233 MULTI:      RAR
234             MOV D,A
235             JMP MULT0
236 E2:         JMP PERSSH
237             END START
```

```
  1 ;           ****************************************
  2 ;           PROGRAM TO COMPUTE CONTROLS FOR
  3 ;           THE MULTIPLIER.THE LOOK-UP TABLE
  4 ;           IS STORED FROM 7400H TO 77FFH.
  5 ;           ONLY LEAST SIGNIFICANT SIX BITS
  6 ;           ARE USED.BIT PATTER FOR BUF3 IS
  7 ;           22,27,23 AND 42.FOR BUF1 IS
  8 ;           20,25,21 AND 40.
  9 ;           ****************************************
 10 ;
 11             CSEG
 12 START:      MVI     B,0H
 13             LXI     H,7400H
 14             MVI     E,00H
 15 ;           CONTROLS FOR BUF3.
 16 ;
 17 LP1:        MVI     M,22
 18             INX     H
 19             MVI     M,27
 20             INX     H
 21             MVI     M,23
 22             INX     H
 23             MVI     M,42
 24             INX     H
 25             INR     B
 26             MOV     A,B
 27             CPI     64
 28             JNZ     LP1
 29 ;           CONTROLS FOR BUF1
 30 ;
 31             MVI     B,00H
 32 LP2:        MVI     M,20
 33             INX     H
 34             MVI     M,25
 35             INX     H
 36             MVI     M,21
 37             INX     H
 38             MVI     M,40
 39             INX     H
 40             INR     B
 41             MOV     A,B
 42             CPI     128
 43             JNZ     LP2
 44 ;           CONTROL FOR BUF3
 45 ;
 46             MVI     B,0EH
 47 LP3:        MVI     M,22
 48             INX     H
 49             MVI     M,27
 50             INX     H
 51             MVI     M,23
 52             INX     H
 53             MVI     M,42
 54             INX     H
```

```
                ;*****************************************
                ;PROGRAM TO COMPUTE CONTROLS FOR THE
                ;BUTTERFLY INPUT MULTIPLEXING UNIT.
                ;THE TABLE IS STORED FROM 7400H TO
                ;77FFH. LEAST SIGIFICANT THREE BITS
                ;CONTIN INFORMATION ABOUT NTT
                ;STAGE AND LEAST SIGNIFICAT FOURTH
                ;BIT IS FOR FORWARTD OR INVERSE NTT.
                ;*****************************************
                ;
                CSEG
START:          MVI     D,00H;
                MVI     E,00H
                MVI     C,00H
                LXI     H,7400H ;STARTING ADDRESS OF TAE
                CALL    LOOP
                MVI     C,8H
                LXI     D,80H
                MVI     B,00H
                CALL    LOOP
                LXI     D,00H
                LXI     B,00H
                CALL    LOOP
                MVI     C,8H.
                LXI     D,80H
                MVI     B,00
                CALL    LOOP
                JMP     E1
LOOP:           XRA     A
LP2:            MOV     B,E
LP1:            MOV     M,C
                INR     B
                INX     H
                MOV     A,B
                CPI     64
                JNZ     LP1
                INR     C
                INR     D
                MOV     A,D
                CPI     7
                JNZ     LP2
                RET
E1:             JMP     0F855H
                END     START
```

```
                INR     B
                MOV     A,B
                CPI     64
                JNZ     LP3
                INR     E
                MOV     A,E
                CPI     4
                JNZ     LP1
                JMP     0F855H
                END     START
```

```
1 ;     ****************************************
2 ;     PROGRAME TO COMPUTE ADDITION LOOK-UP
3 ;     TABLE FOR MODULUS 30.
4 ;      TABLE IS STORED FROM 7400H TO 77FFH.
5 ;     ****************************************
6       CSEG
7 START:  MVI A,00H
8         LXI H,7400H ; STARTING ADDRESS OF  TABLE
9         MVI D,00H
10        MVI B,00H
11 LP2:   MVI C,00H
12 LP1:   MOV A,C
13        ADD B
14        CPI 30
15        JC LOC1
16        SUI 30  ;MODULO 30 REDUCTION.
17 LOC1:  MOV D,A
18        MOV A,C
19        CPI 30
20        JC LOC5
21        MVI D,31
22 LOC5:  MOV M,D
23        INR C   ;INCREMENT THE ADDEND.
24        INX H
25        MOV A,C
26        CPI 32
27        JNZ LP1
28        INR B   ;INCREMENT THE ADDER.
29        MOV A,B
30        CPI 30
31        JNZ LP2
32        MVI E,00H
33 LOC7:  INR E
34        MVI M,31
35        INX H
36        MOV A,E
37        CPI 63
38        JMP 0F855H
39        END START
```

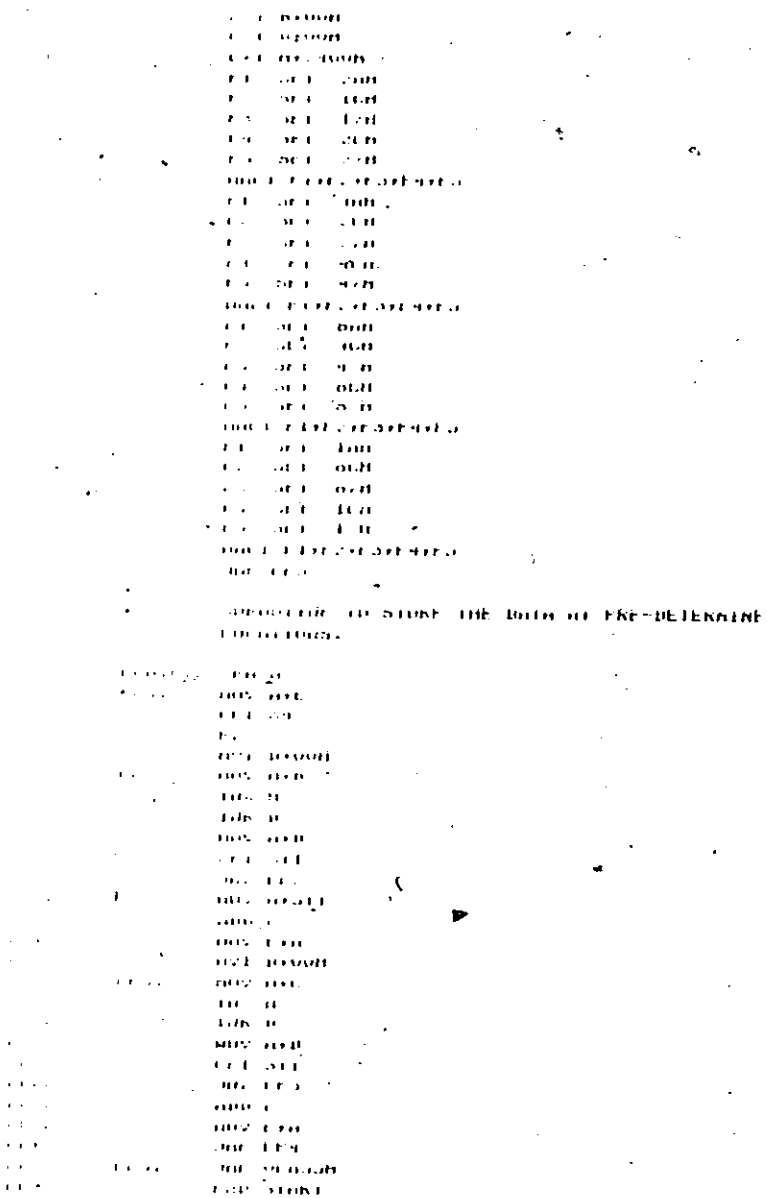
```
1 ;     ****************************************
2 ;     PROGRAME TO COMPUTE ADDITION LOOK-UP TAB
3 ;     FOR MODULUS 31.
4 ;     TABLE IS STORE FROM 7400H TO 77FFH
5 ;     ****************************************
6       CSEG
7 START:  MVI A,00H
8         LXI H,7400H ;STARTING ADDRESS OF  TABLE.
9         MVI D,00H
10        MVI B,00H
11 LP2:   MVI C,00H
12 LP1:   MOV A,C
13        ADD B
14        CPI 31
15        JC LOC1
16        SUI 31  ;MODULO 31 REDUCTION.
17 LOC1:  MOV D,A
18        MOV A,C
19        CPI 31
20        JC LOC5
21        MVI D,31
22 LOC5:  MOV M,D
23        INR C   ;INCREMENT THE ADDEND.
24        INX H
25        MOV A,C
26        CPI 32
27        JNZ LP1
28        INR B   ;INCREMENT THE ADDER.
29        MOV A,B
30        CPI 31
31        JNZ LP2
32        MVI E,00H
33 LOC7:  INR E
34        MVI M,31
35        INX H
36        MOV A,E
37        CPI 31
38        JNZ LOC7
39        JMP 0F855H
40        END START
```

```
110             INX B
111             MOV A,L
112             CPI 193
113             JNZ LC17
114 ;INVERSE LOOKUP TABLE FOR MOD 30 (30,31)
115             CALL RECT ;GET RECONSTRUCTION TABLE
116             LXI H,8000H
117             LXI B,7400H
118             MVI D,00
119 LC24:       MVI E,00
120 LC23:       MOV A,E
121             CPI 30
122             JNC LC20
123             MOV A,M
124             PUSH H
125             LXI H,5400H
126             ADD L
127             MOV L,A
128             MOV A,M
129             STAX B
130             POP H
131             INX H
132             JMP LC25
133 LC20:       MVI A,0
134             STAX B
135 LC25:       INX B
136             INR E
137             MOV A,E
138             CPI 32
139             JNZ LC23
140             INR D
141             MOV A,D
142             CPI 30
143             JNZ LC24
144             MVI E,00
145 LC26:       INR E
146             MVI A,00
147             STAX B
148             INX B
149             MOV A,E
150             CPI 62
151             JNZ LC26
152             JMP E1
153 ;INVERSE INDEX TABLE FOR MOD 31 (30,31)
154             CALL RECT ;GET RECONSTRUCTION TABLE
155             LXI H,8000H
156             LXI B,9500H
157             MVI D,00
158 LC34:       MVI E,00
159 LC33:       MOV A,E
160             CPI 30
161             JNC LC30
162             MOV A,M
163             PUSH H
164             LXI H,5500H
165             ADD L
166             MOV L,A
167             MOV A,M
168             STAX B
169             POP H
170             INX B
171             JMP LC35
172 LC30:       MVI A,0
173             STAX B
174 LC35:       INX B
175             INR E
176             MOV A,E
177             CPI 32
178             JNZ LC33
179             INR D
180             MOV A,D
181             CPI 31
182             JNZ LC34
183             MVI E,00
184 LC36:       INR E
185             MVI A,00
186             STAX B
187             INX B
188             MOV A,E
189             CPI 31
190             JNZ LC36
191             JMP E1
192 ;           SUBROUTINE TO COMPUTE RECONSTRUCTION TAB
193 RECT:       MVI B,00H
194 ;LOOKUP TABLE FOR RECONSTRUCTION OF MOD 30 & 31
195 START:      MVI A,00H
196             MVI B,00H
197             MVI C,00H
198             MVI D,00H
199             LXI H,7000H
200 LOK1:       MVI E,31
201             PUSH B
202             CALL MULT ;CALL TO MULTIPLICATION TABLE
203             MOV M,E
204             INX H
205             MOV M,D
206             POP B
207             INX H
208             INR B
209             MOV A,B
210             CPI 30
211             JNZ LOK1
212             MVI B,00H
213             MVI A,00H
214             MVI C,00H
215             MVI D,00H
216             LXI H,7100H
217 LOK3:       MVI E,30
218             PUSH B
219             PUSH H
```

```
        CALL MULT ;CALL TO MULTIPLICATION ROUTIN
LOK4:   MOV A,E
        CPI 31
        JNC LOK5
        MOV A,D
        CPI 0
        JZ LOK6
LOK5:   LXI H,-31 ;MODULO 31 REDUCTION
        DAD D
        XCHG
        JMP LOK4
LOK6:   POP H
        POP B
        MOV M,E
        INX H
        INR B
        MOV A,B
        CPI 31
        JNZ LOK3
        MVI A,00H
        LXI H,7100H
        MVI D,00H
        LXI B,7200H
LOK7:   PUSH B
        MVI E,30
        MOV B,M
        CALL MULT ;CALL TO MULTIPLICATION ROUTIN
        POP B
        MOV A,E
        STAX B
        INX B
        MOV A,D
        STAX B
        INX H
        INX B
        MOV A,L
        CPI 31
        JNZ LOK7
        LXI D,8000H
        LXI B,7200H
LOK9:   LXI H,7000H
LOK8:   XRA A
        PUSH B
        LDAX B
        ADC M
        STAX D
        INX B
        INX H
        INX D
        LDAX B
        ADC M
        STAX D
        POP B
        INX H
        INX B
        MOV A,L
        CPI 60
        JNZ LOK8
        INX B
        INX B
        MOV A,C
        CPI 62
        JNZ LOK9
        LXI B,3720
        LXI H,8000H
ADS3:   PUSH H
        MOV E,M
        INX H
        MOV D,M
ADS2:   MOV A,D
        CPI 3
        JC ADS1
        JNZ ADS4
        MOV A,E
        CPI 162
        JC ADS1
ADS4:   LXI H,-930 ;MODULO 30*31 REDUCTION
        DAD D
        XCHG
        JMP ADS2
ADS1:   POP H
        MOV M,E
        INX B
        INX H
        MOV M,D
        DCX B
        INX H
        MOV A,B
        CPI 0
        JNZ ADS3
        MOV A,C
        CPI 0
        JNZ ADS3
        LXI B,8000H
        LXI H,8000H
        LXI D,930
ADS8:   PUSH D
        PUSH H
        MOV E,M
        INX H
        MOV D,M
ADS7:   MOV A,E
        CPI 192
        JNC ADS5
        MOV A,D
        CPI 0
        JZ ADS6
ADS5:   LXI H,-192 ;MODULO 192 REDUCTION
        DAD D
        XCHG
```

```
        JMP ADS7
ADS6:   POP H
        INX H
        INX H
        MOV A,E
        STAX B
        INX B
        POP D
        DCX D
        MOV A,D
        CPI 0
        JNZ ADS8
        MOV A,E
        CPI 8
        JNZ ADS8
        RZ
;       SUBROUTINE TO MULTIPLY TWO NUMBERS.
MULT:   MVI D,00H
        MVI C,9
MULT0:  MOV A,E
        RAR
        MOV E,A
        DCR C
        RZ
        MOV A,D
        JNC MULT1
        ADD B
MULT1:  RAR
        MOV D,A
        JMP MULT0
E1:     JMP 0F855H
        END START
```

```
;       ****************************************************************
;       PROGRAM TO COMPUTE THE CONTROL LOOKUP TABLE FOR THE
;       MULTIPLEX INPUT MULTIPLEXING UNIT. THE TABLE IS STORED
;       FROM ADDRESS  00H UPWARDS. THE 8-BIT CONTROL BIT
;       PATTERN IS STORED.
;       ****************************************************************
        ORG
;
;       MACRO TO LOAD REQUIRED BIT PATTERN
;
MULT    MACRO   P1,P2,P3,P4,P5
        STI     SET 01H
        LXI     B,00H
        LXI     D,00H
        MVI     B,F1
        MVI     C,F1
        CALL    LOGIC
        STI     SET 01H
        LXI     B,00H
        LXI     D,00H
        MVI     B,F2
        MVI     C,F3
        CALL    LOGIC
        STI     SET 02H
        LXI     B,00H
        LXI     D,00H
        MVI     B,F1
        MVI     C,F3
        CALL    LOGIC
        STI     SET 04H
        LXI     B,00H
        LXI     D,00H
        MVI     B,F2
        MVI     C,F3
        CALL    LOGIC
        STI     SET 08H
        LXI     B,00H
        LXI     D,00H
        MVI     B,F4
        MVI     C,F5
        CALL    LOGIC
        STI     SET 10H
        LXI     B,00H
        LXI     D,00H
        MVI     B,F2
        MVI     C,F3
        CALL    LOGIC
        STI     SET 20H
        LXI     B,00H
        LXI     D,00H
        MVI     B,F4
        MVI     C,F5
        CALL    LOGIC
        ENDM
;
;       MAIN PROGRAM.
;
START.  MVI     A,00H
```



```
1.      ;       ************************************************************
2.      ;       PROGRAM TO COMPUTE THE LOOK-UP TABLE FOR THE
3.      ;       DISTRIBUTOR UNIT, BUFFER INTER-CONNECTIONS
4.      ;       AND OUTPUT MULTIPLEXING UNIT. THE FIVE LEAST
5.      ;       SIGNIFICANT BITS STORED THE CONTROLS FOR THE
6.      ;       MEMORY BUFFER INTER-CONNECTIONS. SIXTH AND
7.      ;       SEVENTH BIT STORES THE CONTROL FOR THE DISTRIBUTOR
8.      ;       UNIT. THE MOST SIGNIFICANT BIT IS FOR OUTPUT
9.      ;       MULTIPLEXING UNIT.
10.     ;       ************************************************************
11.             CSEG
12.     ;       MACROS LOGIC, RECV1 AND RECV2 LOAD THE REQUIRED
13.     ;       BIT PATTERN.
14.     LOGIC   MACRO   STN,LH
15.             CALL  MAC
16.             ENDM
17.     RECV1   MACRO       I1
18.             MVI   M,56H
19.             INX   H
20.             MVI   M,0BH
21.             INX   H
22.             REPT  I1
23.             MVI   M,56H
24.             INX   H
25.             ENDM
26.             ENDM
27.     RECV2   MACRO         I2
28.             REPT   I2
29.             MVI   M,5EH
30.             INX   H
31.             ENDM
32.             ENDM
33.     ;
34.     ;       MAIN PROGRAM.
35.     ;
36.     START:  XRA   A
37.             LXI   H,7400H
38.             CALL  MUL
39.             CALL  STAGD
40.             LXI   B,00H
41.             LXI   D,00H
42.             MVI   B,6DH
43.             MVI   C,6DH
44.     STN     SET   01H
45.     LH      SET   56H
46.             CALL  MAC
47.             LXI   B,00H
48.             LXI   D,00H
49.             LXI   H,7800H
50.             MVI   M,0FFH
51.             INX   H
52.             MVI   M,0FFH
53.             INX   H
54.             MVI   B,0BH
55.             MVI   C,0BH
```

```
STN     SET     01H
LN      SET     80H
        CALL    MAC
        LXI     H,7800H
LG      SET     78H
STN     SET     04H
K1      SET     02H
K2      SET     04H
        RECV1   K1
        RECV2   K2
        MVI     B,56H
        MVI     C,5EH
        LXI     D,00H
        CALL    MAC
        LXI     H,7900H
        MVI     M,0FFH
        INX     H
        MVI     M,0FFH
        INX     H
STN     SET     04H
LN      SET     78H
        MVI     C,0BH
        MVI     B,0BH
        LXI     D,00H
        CALL    MAC
        LXI     H,7980H
STN     SET     08H
J1      SET     06H
J2      SET     08H
LN      SET     70H
        LXI     D,00H
        MVI     B,56H
        MVI     C,5EH
        RECV1   J1
        RECV2   J2
        CALL    MAC
        LXI     7A00H
        MVI     M,0FFH
        INX     H
        MVI     M,0FFH
STN     INX     H
STN     SET     08H
LN      SET     70H
        LXI     D,00H
        MVI     B,0BH
        MVI     C,0BH
        CALL    MAC
        LXI     H,7A80H
L1      SET     1EH
L2      SET     28H
        RECV1   L1
        RECV2   L2
        LXI     H,7B00H
        MVI     M,0FFH
        INX     H
        MVI     M,0FFH
        INX     H
STN     SET     01H
LN      SET     80H
        LXI     D,00H
        MVI     B,0BH
        MVI     C,0BH
        CALL    MAC
        LXI     H,7B80H
        LXI     D,00H
        MVI     B,56H
        MVI     C,5EH
STN     SET     02H
LN      SET     7CH
T1      SET     00H
T2      SET     00H
        RECV1   T1
        RECV2   T2
        CALL    MAC
        LXI     H,7C00H
        MVI     M,0FFH
        INX     H
        MVI     M,0FFH
        INX     H
        LXI     D,00H
        LXI     B,0BH
STN     SET     08H
LN      SET     20H
        CALL    MAC
        LXI     H,7C80H
        LXI     D,00H
        MVI     B,56H
        MVI     C,5EH
STN     SET     08H
LN      SET     20H
R1      SET     06H
R2      SET     08H
        RECV1   R1
        RECV2   R2
        CALL    MAC
        LXI     H,7D00H
        CALL    MUL
        JMP     E2
;
;       SUBROUTINES MAC, MUL AND STAGD STORE THE DATA
;       AT THE PRE DETERMINED LOCATIONS.
;
;
MAC:    XRA     A
LK4:    MOV     A,E
        CPI     LN
        RZ
        MVI     D,00H
LK2:    MOV     M,B
        INX     H
```

```
                INR   D
                MOV   A,B
                CPI   STN
                JNZ   LK2
                MVI   A,STN
                ADD   C
                MOV   E,A
                MVI   D,00H
LK2:            MOV   H,C
                INX   H
                INR   D
                MOV   A,B
                CPI   STN
                JNZ   LK3
                MVI   A,STN
                ADD   C
                MOV   E,A
                JMP   LK4
;
;
STAGD:    XRA   A
                LXI   D,00H
                LXI   B,00H
                MVI   B,28H
                MVI   C,23H
AR2:      MVI   E,06H
AR1:      MOV   H,B
                INX   H
                INR   C
                MOV   A,C
                CPI   7
                JNZ   LK1
                MOV   H,C
                INX   H
                MOV   H,B
                INX   H
                MOV   H,B
                INX   H
                MOV   H,B
                INX   H
                MOV   H,B
                INX   H
                INR   D
                MOV   A,D
                CPI   3
                JNZ   AR2
                RET
;
;
MUL:      LXI   D,00H
                LXI   D,00H
LP2:      MVI   B,00H
LP1:      MVI   H,35H
                INX   H
                INR   B
                MOV   A,B
                CPI   7
                JNZ   LP1
LP3:      MVI   H,35H
                INX   H
                INR   B
                MOV   A,B
                CPI   0EH
                JNZ   LP3
                INR   E
                MOV   A,E
                CPI   64.
                JNZ   LP3
                RET
E2:       JMP   0FB55H
END  START
```

VITA AUCTORIS

1956 Born on the 15th of August in Bombay, India

1972 Completed High School at Parle Tilak Vidyalaya, Bombay, India

1978 Graduated from University of Bombay, Bombay India, with the degree of Bachelor of Engineering in Electrical Engineering

1979 Worked as Trainee Systems Engineer at Tata Burroughs Limited, Bombay, India

1982 Candidate for the degree of M.A.Sc. in Electrical Engineering at the University of Windsor, Windsor, Ontario, Canada