1-1-1981

A read-only-memory oriented implementation of the number theoretic transform butterfly unit.

Mahmood Akhtar
University of Windsor

A READ-ONLY-MEMORY ORIENTED IMPLEMENTATION OF THE NUMBER THEORETIC TRANSFORM BUTTERFLY UNIT

by

MAHMOOD AKHTAR

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

Windsor, Ontario, Canada

1981

UMI Number: EC54737

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI®

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

763008

ABSTRACT

This thesis is concerned with the design of a hardware implementation of a Number Theoretic Transform butterfly structure. The butterfly is being used as the computational element in a Number Theoretic Transform processor suitable for digital signal processing operations. The butterfly has been realized using arrays of read-only-memory (ROM) and table look-up techniques. All mathematical operations performed by the Number Theoretic Transform butterfly have been carried out using the Residue Number System. The ROM oriented structure lends itself to an efficient realization using very large scale integration (VLSI) technology. The use of high density EPROMS in a pipeline configuration results in a structure suitable for real time signal processing applications.

ACKNOWLEDGEMENT

I would like to express my sincere thanks to my supervisor, Dr. W.C. Miller for many valuable discussions and constructive criticism on this thesis. I am also very thankful to Dr. G.A. Jullien for his advice and assistance throughout the study period. Thanks are due to the other members of the Department and my fellow graduate students, especially Mr. H.K. Nagpal who helped me in various ways.

To my parents, I extend my sincere gratitude. Without their help and love, though far away, this work would not have started.

Thanks are also due to Mrs. Marion Campeau for her diligence in typing this thesis.

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

LIST OF FIGURES

Figure		Page

CHAPTER 1

INTRODUCTION

1.1 PREAMBLE

This thesis describes a hardware realization of a number theoretic transform butterfly. 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 authors responsibility in this project was to design an NTT butterfly that can be multiplexed with a memory support structure to ultimately provide a digital filtering capability.

1.2 NUMBER THEORETIC TRANSFORM

Finite digital convolution has many practical applications in digital signal processing. It can be used to implement non-recursive digital filters. It can also be used to carry out auto and cross correlation, as well as, polynomial multiplication. The direct method of computing a convolution sum involves a number of multiplications proportional to the product of the length of the two inputs [14]. Multiplication in a digital system, is a relatively slow operation and therefore techniques were investigated to minimize the number of multiplications in the convolution sum. The use of transform techniques to compute convolution is quite popular and the savings in multiplication time over direct method depends upon the transform length.

The characteristic of these transforms are such that the convolution in time domain is equivalent to pointwise multiplication in transform domain.

The discrete Fourier transform (DFT) is defined in the complex number field and is one of the transforms that exhibits the cyclic convolution property. The DFT is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j.2\pi/N.nk}, \quad k = 0,1,\ldots,N-1 \qquad (1.1)$$

The DFT becomes very attractive to use as it can be implemented efficiently using the Fast Fourier Transform (FFT) type algorithm [15]. The two main disadvantages associated with the FFT are the multiplication by irrational coefficients and the inherent number growth. Both of the above introduce truncation and/or round-off errors when implemented on a finite wordlength machine.

Pollard [4] has shown that transforms defined in a finite ring also exhibit the cyclic convolution property. These transforms are named as Number Theoretic Transforms (NTT) because number theoretic concepts are used in their definition. The number theoretic transform is defined as

$$X(k) = \left| \sum_{n=0}^{N-1} x(n)\, \alpha^{nk} \right|_M \quad k = 0,1,2,\ldots,N-1 \qquad (1.2)$$

where α is the cyclic generator of order N. These transforms are implemented using an integer number system. Since these transforms are defined in finite rings, the number growth problem is inherently solved. The value of M is chosen such that the result of the convolution is within the defined range. Whenever the result of an operation exceeds M, the number is reduced modulo M and if the final result is within the dynamic range, the intermediate overflows can be ignored. Thus the computation is exact and truncation-roundoff errors do not arise.

The proposed implementation of the NTT requires a supporting memory structure and a computational unit commonly known as the butterfly unit (BF). The operations performed by the butterfly unit are addition, subtraction and multiplication, but no division. The complexity of the BF unit depends upon the choice of the field and also the form of the generator, which is used to define the number theoretic transform.

1.3 THE NTT BUTTERFLY UNIT

The binary operations in the BF unit are performed modulo an integer M, which is used in the definition of the NTT. Modulo reduction is not an easy operation unless the modulus M has a simpler form, preferably a power of two for the Binary number system implementation of the BF unit. Radar [6] used the Mersenne number and Agarwal and Burrus [7] used the Fermat numbers to ease of the computation in the BF unit using the binary number system to perform the required arithmetic operations modulo M. McClellan [16] has built hardware for implementing the Fermat number transform and used adders-subtractors to implement the BF unit. The generator was chosen such that the multiplications by twiddle factors were replaced by bit shiftings.

These adders-subtractors and the bit shifting were arranged in a pipeline configuration for a high throughput rate.

In using an array of ROMS, rather than adder-subtractor, etc., an extremely simple structure emerges that offers identical characteristics for any required operation and is inherently simple to pipeline. The use of ROM arrays for implementing BF unit also relaxes the constraints on the choice of the parameters for NTT and they can be chosen freely on purely number theoretic basis to maximize the transform length.

1.4 OBJECTIVE AND OUTLINE OF THE WORK

The use of NTT to compute convolution is very attractive because of its error free computation. The heart of the processor is the computational unit or the Butterfly unit. The orientation in this work is to utilize the advancement in memory fabrication technology and build up a butterfly unit using arrays of look up tables arranged in a pipeline configuration. The look up table approach is quite attractive because of the fact that multiplication can be performed by accessing the data from the tables and thus the multiplication time is reduced to the access time of the ROMS.

Normally the dynamic range assocaited with an NTT processor would be too large to allow an efficient realization based on table look up techniques. In this work the Residue Number System has been employed so that a problem with a large dynamic range can be converted to a number of parallel operations with small dynamic ranges. In this manner a realization based on array of ROM is not only practical but desirable as it is able to exploit the rapidly evolving VLSI technology associated with memory fabrication.

The present work was divided into three phases. The first phase of the work consisted of a literature survey to establish the theoretical basis for the design of the NTT processor. Pollard [4] has defined transforms in finite rings/field and has showed the cyclic convolution property (ccp) of the transforms. Agarwal and Burrus [7] have established the necessary conditions for the transforms to exhibit the ccp. Baraniecka [8] has proposed the look-up table approach using the residue number system to implement the computational unit of Number Theoretic Transform (NTT) processor. The use of look-up tables relaxes the constraints on the choice of the parameters of the NTT. Baraniecka [8] also outlined the procedure for selecting the NTT parameters for look-up table implementation.

Pease [9] has presented a procedure for the design of the memory organization of a FFT processor and Corinthois [10]-[11] has used this idea as the basis for a proposed memory organization for a FFT processor. The same memory organization is used for the FNTT processor because of the similar structure of the two transforms.

The second phase of the work was to design a complete read-only-memory oriented hardware implementation of the NTT Butterfly unit. The design utilizes the table look-up approach and employes a pipeline configuration. A computer simulation of the hardware structure of the NTT bufferly and the associated memory organization was carried out on the IBM 370/3031 facility to verify the validity of the proposed structure. The simulation consisted of generating the look-up tables and then arranging them in the pipeline configuration to check the operations of the pipeline. A convolution of sequences was performed to establish the right selection of the parameters.

The final phase of the work was to actually build a prototype computational unit using 2708 Eproms and 8212 as registers arranged in a pipeline fashion. The registers are necessary for storing the intermediate data to keep the pipeline full. This unit was then tested for real time application.

1.5 THESIS ORGANIZATION

Chapter 2 provides a review of the basic modular arithmetic used in the design. The advantage of using the RNS for a look-up table implementation, especially for multiplication, is established. Binary operations using sub-moduli techniques are also described and the implementation of addition-subtraction using look up tables is shown. An efficient way of performing multiplication for large primes is also described in this chapter.

Chapter 3 starts with an introduction to digital convolution and its implementation using transform techniques. Decimation in time (DIT) and decimation in frequency (DIF) forms of the FFT algorithm are presented in detail.

The choice of the parameter for the NTT and the construction of the 2nd degree extension Galois fields are reviewed. A suitable choice of parameters for an RNS based implementation of the Number Theoretic Transform is discussed.

The concept of an NTT processor is provided in Chapter 4. A memory structure for real time applications is described and a suitable memory organization is suggested. The selection of the primes for an efficient hardware realization of the NTT butterfly unit is discussed and a final design of the butterfly structures for both kind of primes is presented. These butterfly structures were simulated on an IBM 370 computer and the details of the simulation are included in this chapter.

The butterfly unit for 4n + 1 type primes was then implemented in hardware using 2708 Eproms and 8212 latches. The simplicity of the structure using ROM arrays is obvious from the hardware design. The generation of the look up tables on an Intel 220 system and the other relevant material is discussed, and the clock circuitry for running the pipeline is given.

Chapter 5 summarizes the work presented in the thesis and Chapter 6 presents the conclusions that can be reached regarding this work.

CHAPTER 2

LOOK UP TABLE IMPLEMENTATION OF RESIDUE ARITHMETIC

2.1 INTRODUCTION

The look up table approach offers the potential for a ROM oriented high speed realization. This approach is particularly advantageous in realizing multiplication operations, which now become as simple and fast as addition. The use of the Residue Number System (RNS) to implement addition, subtraction and multiplication in look up tables provides a great saving in hardware and is more efficient than the BNS. The RNS is also an inherently carry-borrow free system and does not introduce internal delays due to carry-borrow digit propagation.

In this chapter a detailed discussion of the residue number system and its implementation using look up tables is presented. The concepts developed here will be applied to the number theoretic transform (NTT) in the next chapter.

The residue number system is an integer number system and in the following discussion, all the variables take on integer values only.

2.2 MODULAR ARITHMETIC

If two integers, a and m, are related by the following equation

$$a = q \cdot m + r \qquad (2.1)$$

where q and r are integers and $r \in 0, 1, \ldots\ldots, m-1$, then r is the residue of a, modulo m, and is represented as:

$$r = |\ a\ |_m \tag{2.2}$$

From eq. (2.1) it is clear that q is the quotient and r is the least positive remainder of $\frac{a}{m}$.

Definition 1: If two integers have the same residue then they are called congruent and represented as:

$$a \equiv b \mod m \tag{2.3}$$

such that

$$|\ a\ |_m = |\ b\ |_m = r \tag{2.4}$$

This also implies that (a-b) is divisible by m and written as $m|(a-b)$. Thus all integers are congruent mod m to some integer in the finite set {0,1,2,...,m-1} and are said to belong to one of the m classes. The residue classes mod m form a commutative ring with identity with respect to modulo m addition and multiplication and is denoted by Z_m. For example, if m=7, there are seven distinct classes and the integers belonging to these are

{0} = ..., -14, -7, 0, 7, 14, ...
{1} = ..., -13, -6, 1, 8, 15, ...
{2} = ..., -12, -5, 2, 9, 16, ...
{3} = ..., -11, -4, 3, 10, 17, ...
{4} = ..., -10, -3, 4, 11, 18, ...
{5} = ..., - 9, -2, 5, 12, 19, ...
{6} = ..., - 8, -1, 6, 13, 20, ...

e.g. 13 and 27 belong to the same class as $|13|_7 = |27|_7 = 6$ or $13 \equiv 27$ mod 7.

The following basic arithmetic operations are permissible with modulo arithmetic

a) addition: $8 + 12 = 20 \equiv 3 \mod 17$

b) negation: $-7 \equiv (-7 + 17 = 10) \mod 17$

c) subtraction: $7 - 12 = 7 + (-12) \equiv (7 + 5 = 12) \mod 17$

d) multiplication: $7 \times 12 = 84 \equiv 16 \mod 17$

e) division: $\frac{a}{b}$ exists if b has a multiplicative inverse and b divides a

2.3 RESIDUE NUMBER SYSTEM (RNS)

2.3.1 Representation of Numbers

The representation of an integer in the residue number system takes the form of an n-tuple

$$a = (a_1, a_2, \ldots, a_n) \tag{2.5}$$

of the least positive residue with respect to the set of moduli $(m_1, m_2, \ldots, m_n)$.

The residues, a_i, are formally written $a_i = |a|_{m_i}$. The residue representation of a number is unique. The converse of this statement is true only if the numbers considered are in the range of 0 to M-1 where

$$M = \prod_{i=1}^{n} m_i \tag{2.6}$$

and all the m_i's are relatively prime. If negative numbers are to be represented in this system, then the number range can be divided into two parts. The first part represent positive numbers and second, negative numbers.

For M = even

$$x = +\text{ve no. if } x \in \{0,1,2,\ldots,\frac{M}{2}-1\}$$
$$= -\text{ve no. if } x \in \{\frac{M}{2}, \frac{M}{2}+1, \ldots, M-1\}$$

For M = odd

$$x = +\text{ve no. if } x \in \{0,1,2,\ldots,\frac{M-1}{2}\}$$
$$= -\text{ve no. if } x \in \{\frac{M+1}{2}, \ldots, M-1\}$$

Example 1:

for $n = 3$

and $m_1 = 5$; $m_2 = 7$; $m_3 = 9$

$$M = \prod_{i=1}^{3} m_i = 5 \cdot 7 \cdot 9 = 315$$

positive numbers $\in \{0,1,\ldots,157\}$ †

negative numbers $\in \{158,\ldots,314\}$

2.3.2 Basic Arithmetic Operations in the RNS

Definition 2: A binary operation defined on a set s of elements is a rule that assigns to each pair of elements from s a unique element from s.

Definition 3: A set s is closed with respect to binary operations if

$$a \square b = c \tag{2.7}$$

where a, b and c are any element in s and is the binary operation. The residue number system is, in general, not closed under the binary operation of

† The conversion from the residue number system to signed number system is explained in Sect. 2.3.3 by giving an example.

division as the result of division may not be an integer.

The residue number system is inherently a carry/borrow free system. The binary operations under which the system is closed can be performed by independent operations on the respective digits, i.e.,

$$Z = x \square y \text{ implies } Z_i = |x_i \square y_i|_{m_i} \tag{2.8}$$

where $\square$ represents the allowed binary operations.

It is useful to be familiar with the idea of the multiplicative inverse before considering division in the residue number system.

Assume it is desired to divide x by y in the real number system, then $\frac{x}{y}$ can be written as $\frac{x}{y} = x \cdot \frac{1}{y}$ where $\frac{1}{y}$ is the multiplicative inverse of y, and thus division by y can be replaced by multiplication with $\frac{1}{y}$.

If $\frac{x}{y}$ is not an integer in the real number system, then it can not be represented in the residue number system and division of x by y is not defined in the RNS. But for $\frac{x}{y}$ an integer, in other words, when x is a multiple of y, the idea of a multiplicative inverse can be used to perform division.

Definition 4: If $0 < a < m$ and $|ab|_m = 1$, then a is called the multiplicative inverse of b mod m and is denoted by $a = |\frac{1}{b}|_m$.

The quantity $|\frac{1}{b}|_m$ exists if and only if $(b,m) = 1$ and $|b|_m \neq 0$. In this case $|\frac{1}{b}|_m$ is unique and division can be performed as

$$|\tfrac{x}{y}|_m = |x \cdot |\tfrac{1}{y}|_m|_m \tag{2.9}$$

2.3.3 Conversion From RNS Using Chinese Remainder Theorem (CRT)

In this section conversion from the RNS to any other number system is discussed. This conversion is made possible using a theorem from number theory [1] called the Chinese Remainder Theorem.

Given the residue representation $(r_1, r_2, \ldots, r_n)$ of x, the Chinese Remainder Theorem makes it possible to determine $|x|_M$, provided the greatest common divisor of any pair of moduli is one or moduli are pairwise relatively prime. $|x|_M$ is then given by the following equation:

$$|x|_M = \left| \sum_{j=1}^{n} \hat{m}_j \left| \frac{r_j}{\hat{m}_j} \right|_{m_j} \right|_M \qquad (2.10)$$

where $M = \prod_{i=1}^{n} m_i$, $\hat{m}_i = \frac{M}{m_i}$ and $(m_i, m_j) = 1$ for $i \neq j$

$\left| \frac{1}{\hat{m}_j} \right|_{m_j}$ represents the multiplicative inverse of $\hat{m}_j$ mod m_j.

The following example illustrates the procedure to convert a number from its residue representation using Chinese Remainder Theorem.

Example 2:

let $m_1 = 5$, $m_2 = 7$, $m_3 = 9$

then $M = \prod_{i=1}^{3} m_i = 5 \cdot 7 \cdot 9 = 315$

$\hat{m}_1 = 63$, $\hat{m}_2 = 45$, $\hat{m}_3 = 35$.

$\left| \frac{1}{\hat{m}_1} \right|_{m_1} = \left| \frac{1}{63} \right|_5 = 2$ since $|63 \times 2|_5 = 1$

$$\left|\frac{1}{\hat{m}_2}\right|_{m_2} = \left|\frac{1}{45}\right|_7 = 5 \qquad \text{since} \quad |45 \times 5|_7 = 1$$

$$\left|\frac{1}{\hat{m}_3}\right|_{m_3} = \left|\frac{1}{35}\right|_9 = 8 \qquad \text{since} \quad |35 \times 8| = 1$$

Chinese Remainder Theorem

$$|x|_M = \left| \hat{m}_1 \left|\frac{r_1}{\hat{m}_1}\right|_{m_1} + \hat{m}_2 \left|\frac{r_2}{\hat{m}_2}\right|_{m_2} + \hat{m}_3 \left|\frac{r_3}{\hat{m}_3}\right|_{m_3} \right|_M \tag{2.11}$$

or

$$|x|_M = |63 \cdot |r_1 \cdot 2|_5 + 45 \, |r_2 \cdot 5|_7 + 35 \, |r_3 \cdot 8|_9|_{315} \tag{2.12}$$

Addition

	moduli	5	7	9	
$x = 173$	$\longrightarrow$	3	5	2	
$+y = 94$	$\longrightarrow$	4	3	4	+
$\lvert 267\rvert_{315} = 267$	$\longrightarrow$	2	1	6	

using equation (2.12) where $r_1 = 2$, $r_2 = 1$ and $r_3 = 6$

$|x|_M = |63.4 + 45.5 + 35.3|_{315} = |582|_{315} = 267$

which is the correct result of addition.

Subtraction

	moduli	5	7	9	
$x = 173$	$\longrightarrow$	3	5	2	
$-y = 94$	$\longrightarrow$	4	3	4	-
79	$\longrightarrow$	4	2	7	

using eq. (2.12) for (4,2,7) as (r_1, r_2, r_3)

$|63.3 + 45.3 + 35.2|_{315} = 79.$

If -ve nos. are also to be represented then the number range, 0 to 314, is divided as

0, 1, 2, ..., 157 positive numbers

158, 159,..., 314 negative numbers.

The following example explains the procedure when the result of subtraction is negative

		moduli 5	7	9
x = 94	⟶	4	3	4
-y = 173	⟶	3	5	2
-79	⟶	1	5	2

using equation (2.12), (1,5,2) → 236 since the result lies in the negative number range, it is a negative result therefore subtract 315 from this, 236 - 315 = -79 which is the correct result of subtraction in signed number representation.

Multiplication

Choose the numbers such that the result of multiplication is contained in the dynamic range

		moduli 5	7	9	
x = 41	⟶	1	6	5	
y = 6	⟶	1	6	6	x
246	⟶	1	1	3	

using (2.12) (1,1,3) → 246.

Division

$x = 312 = (2,4,6)$

$y = \ 13 = (3,6,4)$

First find the multiplicative inverse of y_i's

$$\left| \frac{1}{y_1} \right|_{m_1} = \left| \frac{1}{3} \right|_5 = 2 \qquad \text{since } |3\text{x}2|_5 = 1$$

$$\left| \frac{1}{y_2} \right|_{m_2} = \left| \frac{1}{6} \right|_7 = 6 \qquad \text{since } |6\text{x}6|_7 = 1$$

$$\left| \frac{1}{y_3} \right|_{m_3} = \left| \frac{1}{4} \right|_9 = 7 \qquad \text{since } |4\text{x}7|_9 = 1$$

Division can now be performed by multiplying x_i's with multiplicative inverses of y_i's

		moduli	5	7	9	
$x = 312$	⟶		2	4	6	
$\frac{1}{y}$	⟶		2	6	7	x
			4	3	6	

using equation (2.12), $(4,3,6) \rightarrow 24$ which is $\frac{312}{13}$.

To verify that division in RNS will not produce the closest integer value if x is not divisible by y, take

$x = 311 \ = \ (1,3,5)$

$y = \ 13 \ = \ (3,6,4)$

$$\left| \frac{x}{y} \right|_{315} = (1,3,5) \ . \ (2,6,7) = (2,4,8)$$

$(2,4,8) \rightarrow 242 \neq [\frac{x}{y}]_R = 24$

where $[.]_R$ indicates rounding to nearest integer. Note that there is no relation between $\frac{x}{y}$ and $|\frac{x}{y}|_{315}$. The reason is quite obvious. $\frac{x}{y}$ is not an integer and so $|\frac{x}{y}|_{315}$ has no meaning in the RNS. Two conclusions can be drawn from the above examples: (i) The RNS is not a weighted magnitude representation. The residue representation does not give any idea of magnitude and sign of the number represented. (ii) Division is not a simple operation. (iii) Operations on a pair of residues is independent of other residue operations.

2.4 IMPLEMENTATION OF RNS USING LOOK UP TABLES

Recent advances in high density memory technology have made it possible to implement the RNS operations using look-up tables stored in ROMS. The results of the operations can be precalculated and stored in the locations addressed by the input data. Binary operations are then reduced to the accessing of data from the stored tables. This is particularly advantageous in multiplication which becomes as simple and fast as addition. Speed of operation is then dependent only on the access time of the ROMS.

For a given modulus, $m_i \leq 32$, the operation of multiplication and addition modulo m_i of the two numbers can be computed by looking up the result in a 1k x 8 bits commercially available ROMS. Using the same approach, operation moduli m_i, $32 < m_i \leq 64$, would require a 4k x 8 bits ROM or four 1k x 8 bits ROMS and so on.

The RNS is more efficient than the binary number system for look up table implementation as it requires less memory for the same dynamic range. For example, with a wordlength of B bits, 2^B numbers can be represented and therefore a total of $2^B \cdot 2^B = 2^{2B}$ entries are required to store the result of operations in look up tables. For the same dynamic range, m_i's can be choosen such that $\prod_{i=1}^{n} m_i > 2^B$, then each m_i requires m_i^2 entries in the table. Hence a total of $\sum_{i=1}^{n} m_i^2$ entries are needed as compared to the direct implementation which requires $2^{2B} \approx \prod_{i=1}^{n} m_i^2$ and for a reasonable value of n and m_i's

$$\sum_{i=1}^{n} m_i^2 << 2^{2B} .$$

As an example of an RNS implementation using look up table, Fig. 2.1 illustrates a residue multiplier for modulo 31, followed by a residue adder to implement the function $\left| |a.b|_{31} + |c.d|_{31} \right|_{31}$. The input to each table, modulo 31, can be represented by a maximum of 5 bits and the total of the two inputs require ten address lines, the output is five bits and so commercially available 1k x 8 bits ROMS can be used to implement this function. A total of three 1k x 8 ROMS and two stages are required to compute the result. From Fig. 2.1, it is noted that ROM arrays offer the possibility of easy pipelining for high throughput. The data from each ROM is latched and used as a partial address for the next ROM. The only control function required is a latch pulse. For every latch pulse, new input is accepted and a new output is generated. The throughput rate of the system is equal to the inverse of the access time of ROM plus latch settling time.

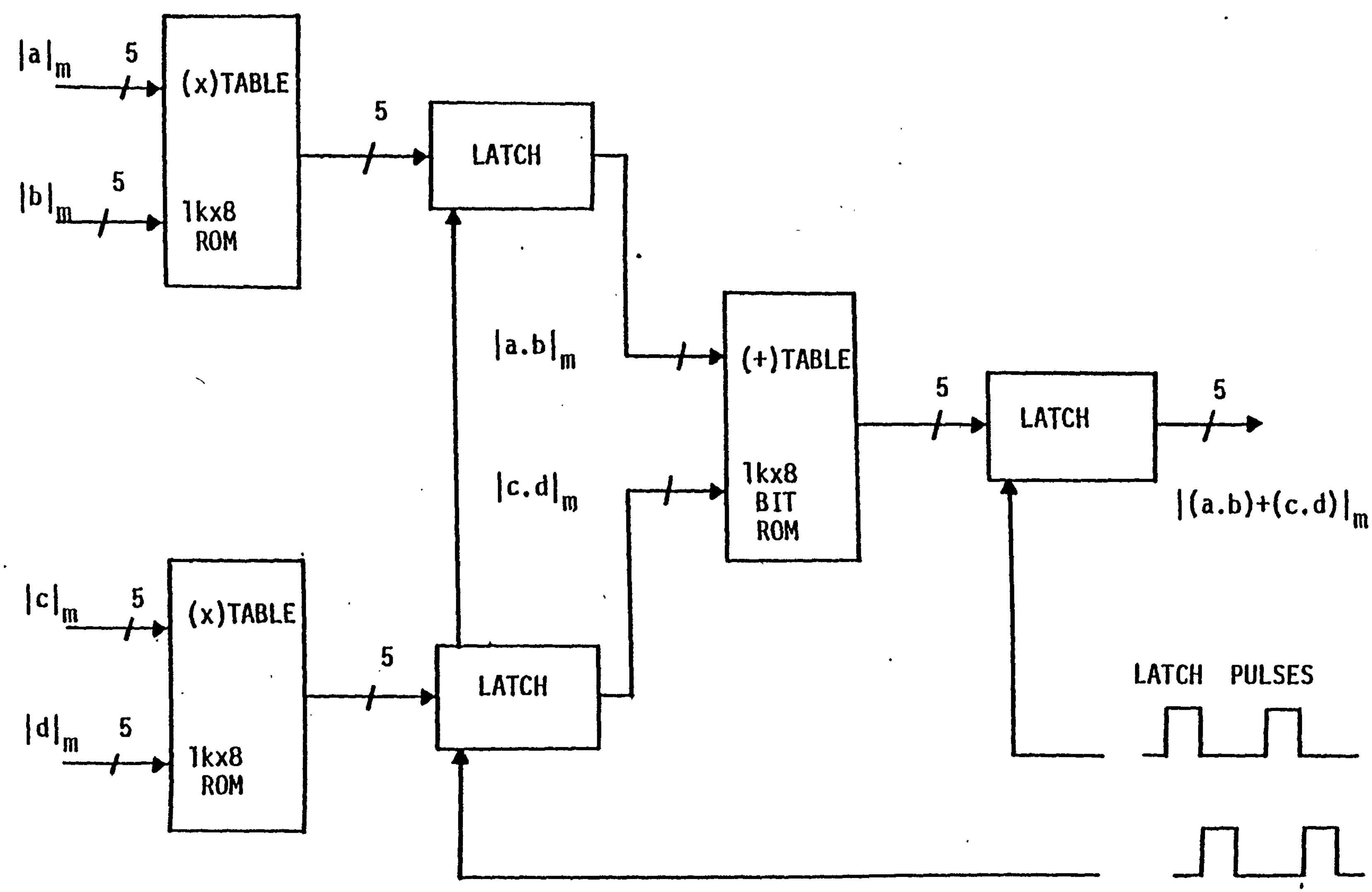


Fig. 2.1 PIPELINE ARRAY FOR THE FUNCTION $|(a.b)+(c.d)|_m$ for $16 < m \leq 32$

Another advantage of the look-up table is that it does not require any extra hardware for addition or multiplication with a constant. The constant can be pre-multiplied or added and can be stored along with the result of the operation.

Example 3:

For modulus m = 9 compute

$Z = \left| 5\,|a.b|_9 + 3\,|c.d|_9 \right|_9$ with a=3, b=4, c=6, d=8.

The result of the computation using residue arithmetic is 6. Fig. 2.2 shows the entries and the interconnections between the look up tables.

Two multiplication and one addition table is required to compute Z. The first multiplication table generates the result of multiplication per-multiplied by 5, modulo 9, and second table generatesthe result of the second multiplication pre-multiplied by 3, modulo 9. Note that multiplication by 3 and 5 does not require any extra storage and does not introduce any extra delay.

2.4.1 Addition/Subtraction Using Sub-Moduli

As mentioned earlier, commerically available ROMS can be used to store tables for the RNS arithmetic, but this imposes an upper limit on the largest modulus to be used. To implement arithmetic modulo $m_i \le 32$, 1k x 8 bits ROMS can be used, operation modulo $32 < m_i \le 64$ would require a 4k x 8 bits ROMS or four 1k x 8 bits ROMS and operation modulo $64 < m_i \le 128$ would require 16k x 8 bits ROM or sixteen of 1k x 8 bits ROMS and so on. As will be explained in the next chapter, prime moduli, $64 < m_i < 512$, are required to implement a pratical NTT, the use

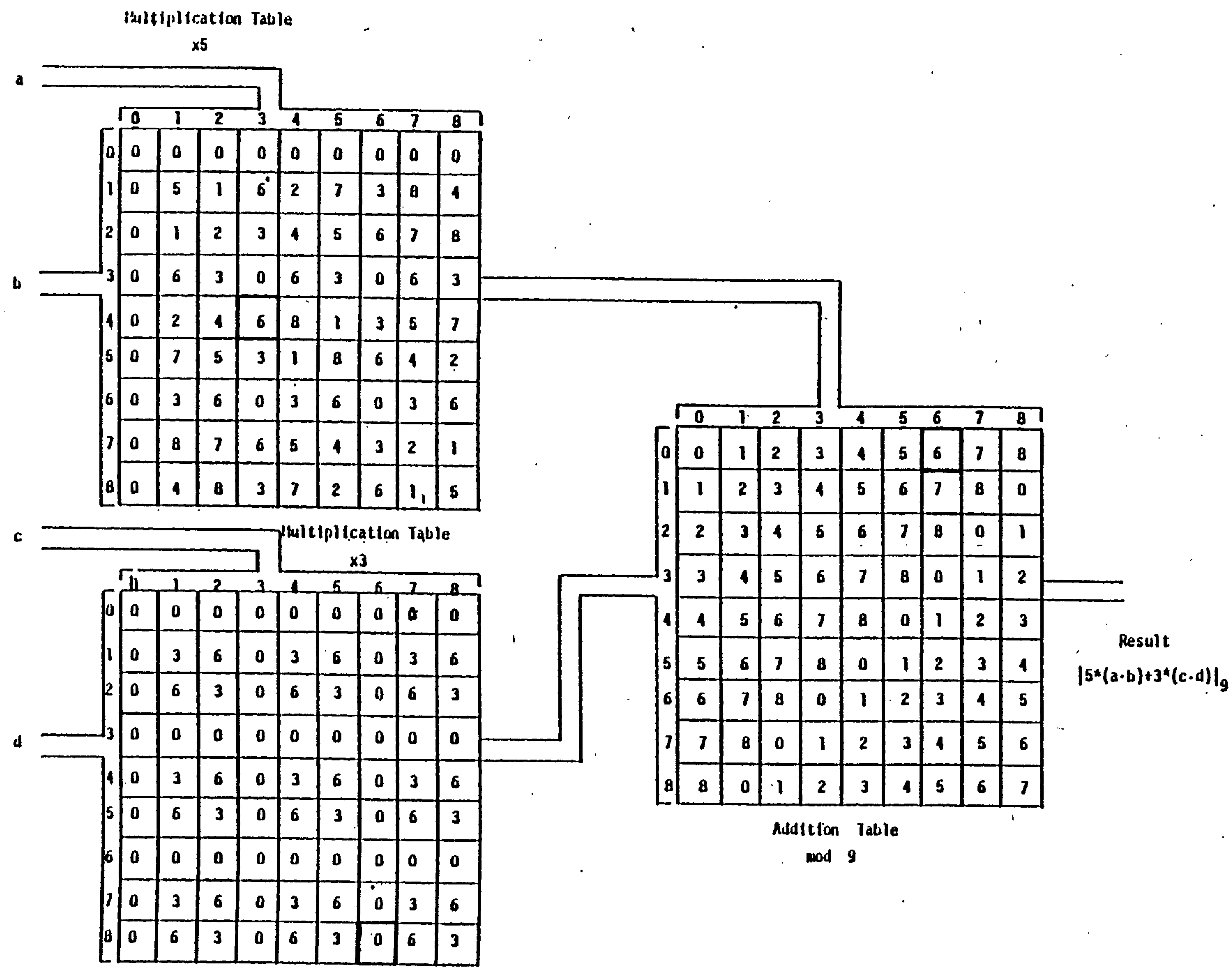


Fig. 2.2 MODULO 9 OPERATIONS WITH PRE-MULTIPLIED CONSTANT

of sixteen or more ROMS does not seem a very efficient approach. In order to increase the implementation efficiency, the same technique of breaking a large dynamic range into smaller moduli can be used to implement the addition/subtraction modulo a large modulus. The only constraints on the choice of sub-moduli is that they should be large enough to contain the result of the operation modulo main modulus and should be relatively prime. For example, if main modulus is m_i, then the maximum number which can occur is $m_i - 1$. The maximum result of addition is $2(m_i - 1)$ and therefore the sub-moduli should be chosen such that their product is greater than $2(m_i - 1)$. Mathematically the condition can be represented as

$$m_{1i} \times m_{2i} > 2(m_i - 1) \qquad (2.13)$$

where m_{1i} and m_{2i} are the sub-moduli.

Multiplication can not be implemented efficiently using the sub-moduli approach as more than two sub-moduli are required to contain the result of multiplication, modulo the main modulus. However, for prime moduli, there exists an efficient method to implement multiplication utilizing the sub-moduli approach and will be dealt with later.

Fig. 2.3 illustrates the addition modulo 19 using 6 and 7 as sub-moduli. First note that 6.7 > 2(19-1) and so these are appropriate sub-moduli, which will produce the correct result of addition modulo 19.

This example is clearly not an efficient one as only one ROM would be necessary to implement addition modulo 19 but this explicitly shows the

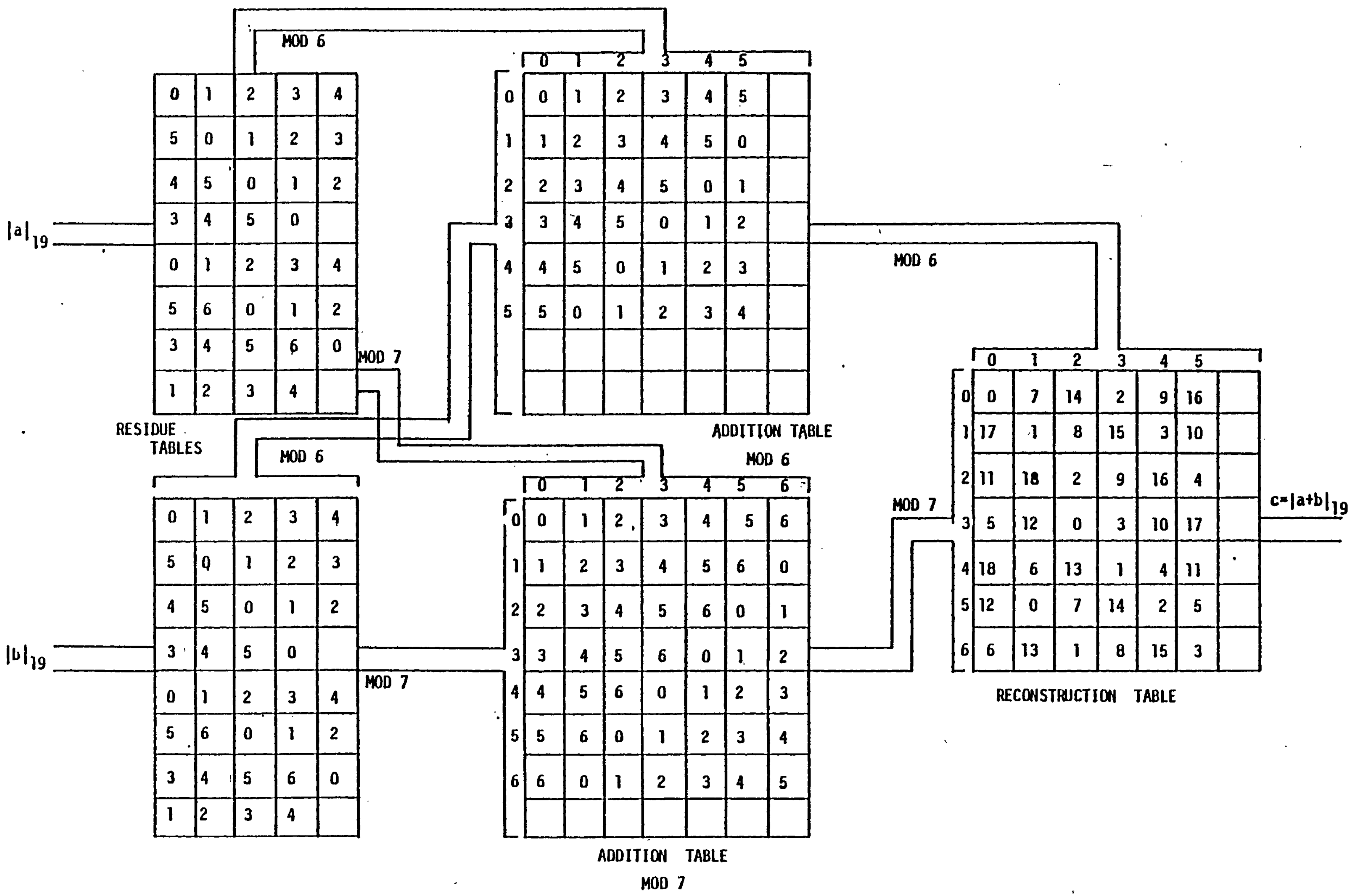


Fig. 2.3 ADDITION MODULO 19 USING 6 and 7 as SUB-MODULI

implementation using sub-moduli.

Example 4:

Assume 1k x 8 bits ROMS are available to implement addition/ subtraction modulo 191. Numbers from 0 to 190 can be represented by 8 bits and hence a total of 16 input (address) lines are required and therefore the memory needed is 64k x 8 bits or 64 ROMS of 1k x 8 bits each. The maximum value of the sub-moduli that can be chosen is 31 which have five bit representation and the look up table will require a total of 10 address lines and so 1k x 8 ROMS can be used to store the tables. Fig. 2.4 shows the implementation using sub-moduli 17 and 23, both have five bit representation. In the first stage, the numbers to be added are reduced modulo 17 and 23. In the next stage, addition modulo 17 and 23 is performed and in the final stage, the result is reconstructed and corrected using chinese remainder theorem to produce the result modulo 191.

A total of seven 1k x 8 ROMS are required to implement addition/ subtraction. It is obvious from this example that sub-moduli scheme saves a lot of memory at the cost of increasing the time of operation. It requires three stages to compute the result whereas direct implementation would have required only one stage but the tremendous saving in hardware is obviously more advantageous.

For implementing subtraction, the same scheme is used except that subtraction tables are required in the 2nd stage of Fig. 2.4 and the entries in reconstruction tables are different.

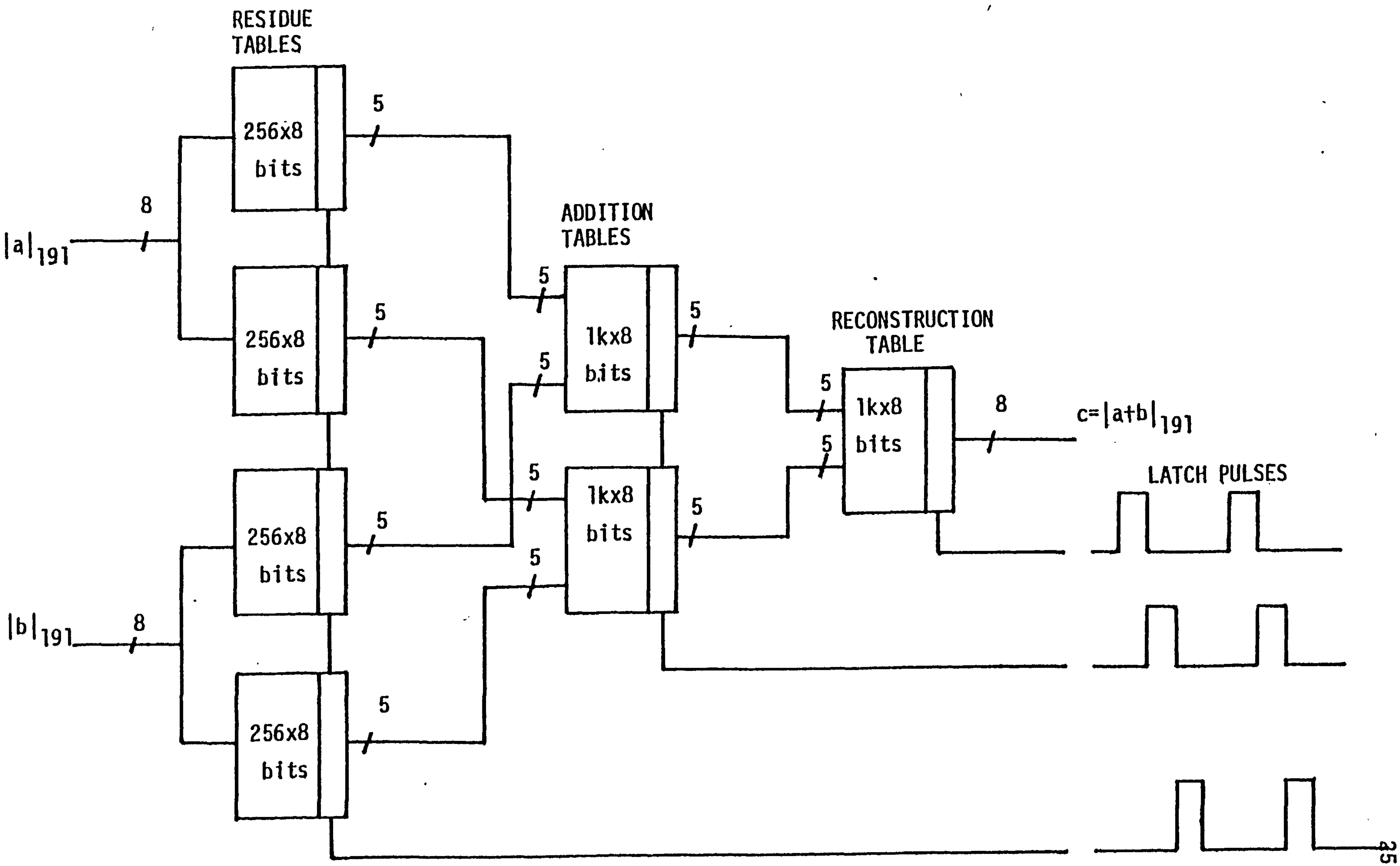


Fig. 2.4 ADDITION USING SUB-MODULI APPROACH

2.4.2 Multiplication Modulo A Prime Number

As explained in the previous section, look up tables speed up the operation of addition-multiplication, if they can be implemented efficiently in hardware. For moduli $m_i \leq 32$, commercially available 1k x 8 ROMS can be used to store the tables of addition/multiplication. For large moduli, addition/subtraction can be implemented efficiently using the sub-moduli approach. For multiplication, however, the direct application of the sub-moduli scheme does not offer an efficient way. Taylor [2] recently proposed a scheme to implement multiplication modulo $(2^n \pm 1, 2^n)$. Jullien [3] presented an efficient scheme to implement multiplication modulo a prime number. For practical NTT's, moduli of interest are primes and therefore Jullien's scheme can be used to implement multiplication. A complete description of the scheme is as follows.

The residue classes (mod m) form a commutative ring with identity with respect to addition and multiplication modulo m, traditionally known as the ring of integers modulo m or the residue ring and denoted by Z_m. The ring of residue classes (mod m) contains exactly m distinct elements. The ring of the residue classes (mod m) is a field if and only if m is a prime number. Thus the non-zero classes of Z_m form a cyclic multiplication group of order m-1, {1,2,..,m-1}, with multiplication modulo m, isomorphic to the addition group {0,1,2,..,m-2} with addition modulo m-1.

This property of isomorphism can be used to implement multiplication and is analogous to multiplication using logarithms.

For a prime modulus, there exists a set of integers, called primitive roots, whose repeated multiplications generates all the elements of the multiplicative group.

$$|\alpha^t|_m = a \in \{1,2,..,m-1\} \tag{2.14}$$

where α is the primitive root and t is the index of a. For different values of t, distinct elements of the field are generated. Note that zero does not have an index and therefore multiplication by zero needs extra care. However in look up table implementation, multiplication by zero can be taken care of easily.

Example 5:

For modulus 11, the primitive root is 2. Table (2.1) shows the element and the respective indices of the field. Multiplication $|6x10|_{11}=5$ can be mapped into addition of indices $|9+5|_{10}=4$. 4 is the index of 5 and the correct result of multiplication is obtained. In this way

x	ind_2 x
1	0
2	1
3	8
4	2
5	4
6	9
7	7
8	3
9	6
10	5

Table 2.1: Index of the elements mod 11.

multiplication is replaced by addition and can be implemented using the sub-moduli approach for large moduli.

The following steps are required to perform multiplication using the index method.

(i) Find the indices of the numbers to be multiplied.

(ii) Add indices mod m-1.

(iii) Perform inverse index operation.

Our main interest is in look up table implementation and therefore a sub-modular ROM adder can be considered. Here the modulus is decomposed into two relatively prime moduli and the addition is carried out within this two moduli system. The final result is reconstructed using another look up table. This reconstruction table can include:

(i) sub-moduli reconstruction using chinese remainder theorem.

(ii) Modulus over flow correction.

(iii) Inverse index look up.

The following example illustrates the complete procedure. Consider the operation, $|x.y|_{19}=Z$ and choose sub-moduli 6 and 7 which gives a composite modulus 6x7 = 42 > 2x19. Fig. (2.5) shows the required tables and appropriate interconnection. Multiplication by zero is invalid using the index method, an invalid index (in this case,7), is stored as the index of zero. In the inverse look up, knowing that 7 will never occur except by multiplication of zero, zero is stored to give the correct result of multiplication. Consider x=13 and y=15, the result is $|13x15|_{19}=5$.

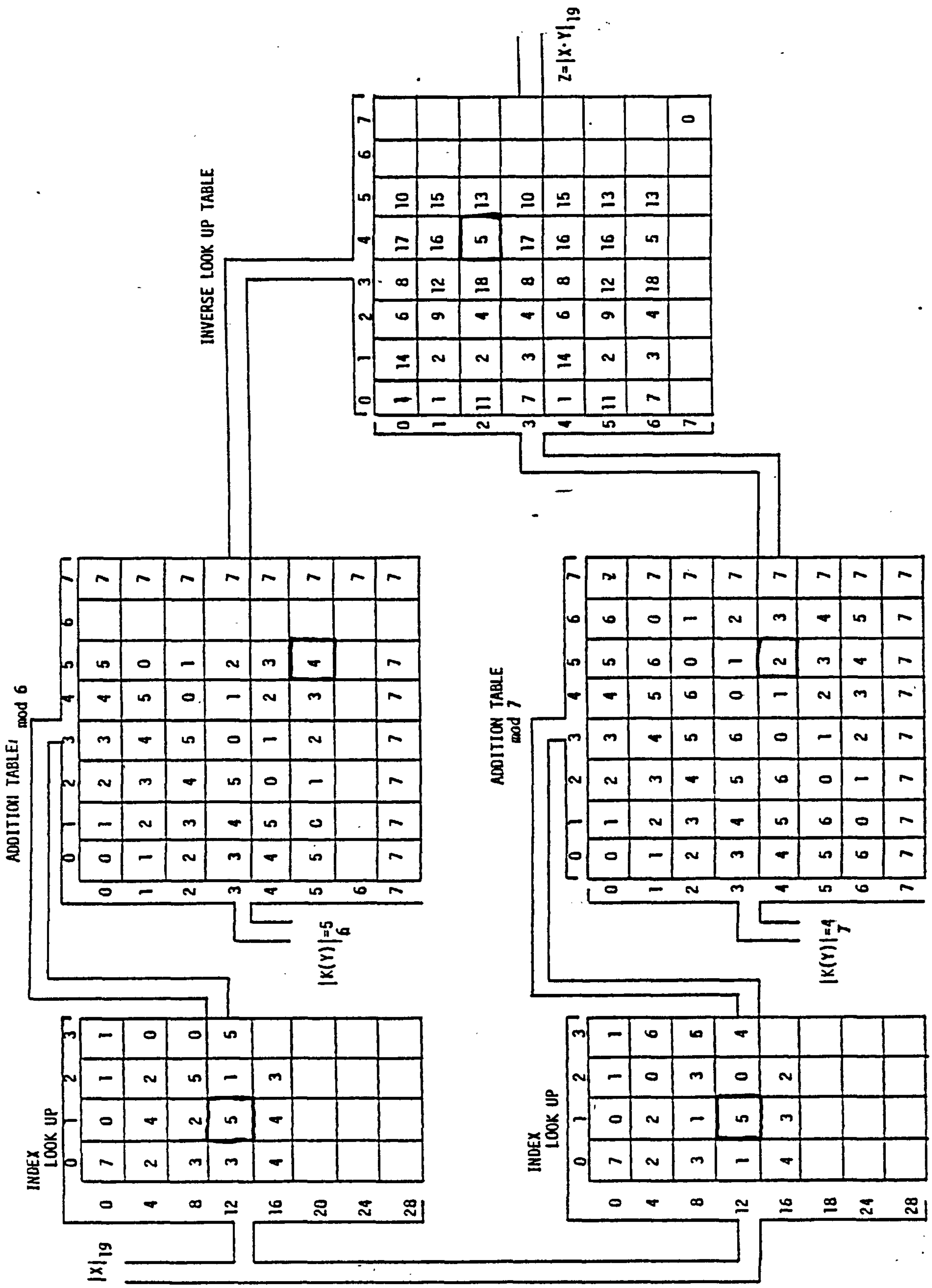


Fig. 2.5 MULTIPLICATION USING INDEX ADDITION AND SUB-MODULI

If the look up tables of Fig. (2.5) are followed (result at every stage is in square) the correct result is obtained.

Fig. (2.6) shows the block diagram for multiplication modulo 191 using sub-moduli 30 and 31. Note the similarity between Fig. (2.4) to perform addition and Fig. (2.6) to perform multiplication. Both operations now take the same time, number of stages and same number of ROMS.

2.5 SUMMARY

In this chapter the basic idea of modular arithmetic was presented. The residue number system was described and was applied to perform binary operations namely addition, subtraction, multiplication and division. The method was clearly illustrated by using examples. The adoptibility of the RNS for a look up table implementation of multiplication and addition was shown.

From the discussion in this chapter it can now be concluded that the RNS is an efficient and fast way of performing addition,subtraction and multiplication since it is inherently a carry borrow free system and there is no interdigit dependence. Division is possible only in certain cases.

The RNS also offers the best result for hardward implementation using look up tables. Multiplication modulo a prime number can be efficiently implemented and offers the same speed of operation as addition.

The ideas will now be used in the next chapter for the definition and implementation of NTTs.

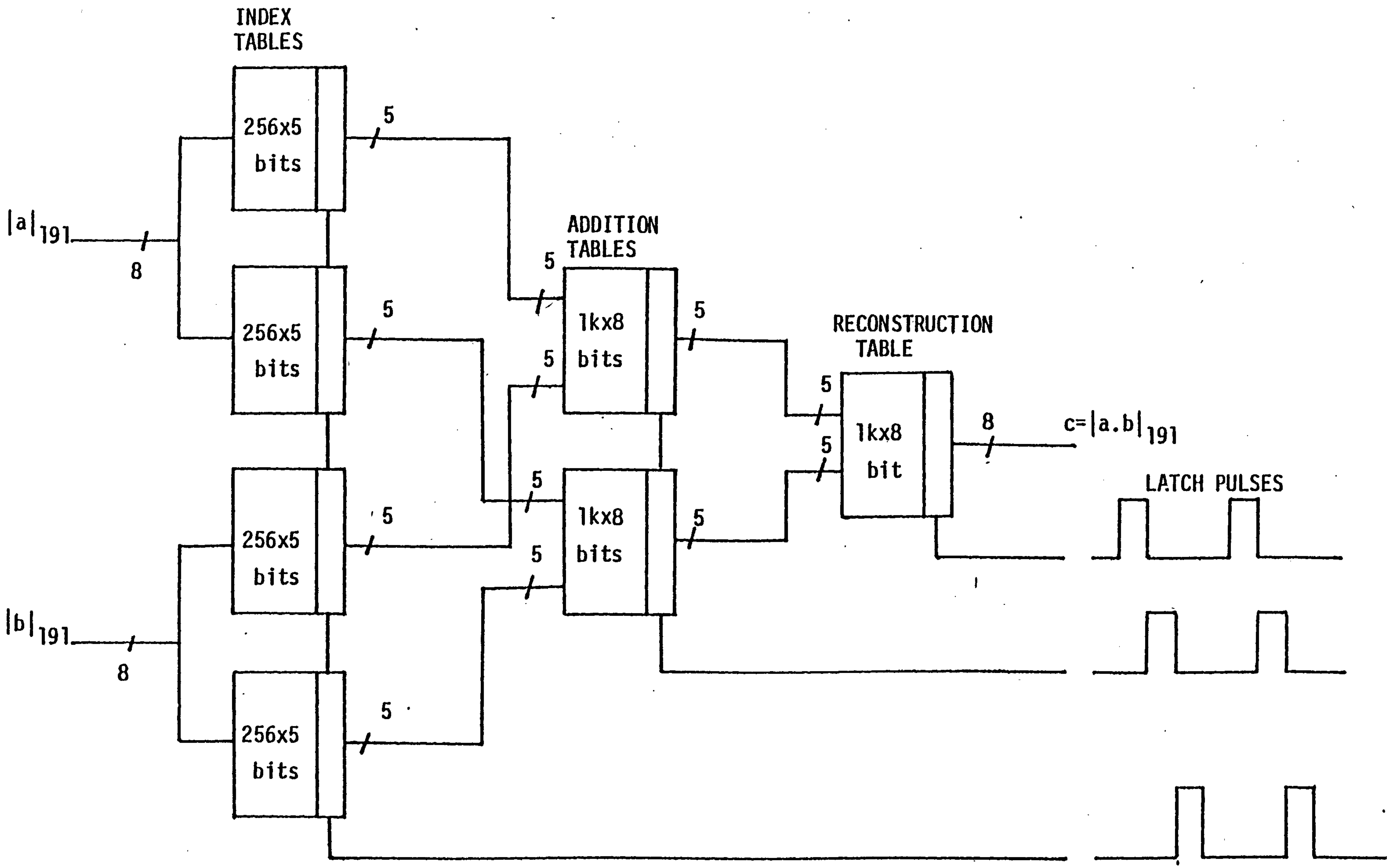


Fig. 2.6 MULTIPLICATION USING INDEX ADDITION MODULO 191

CHAPTER 3

DIGITAL CONVOLUTION AND IMPLEMENTATION USING TRANSFORM TECHNIQUES

3.1 INTRODUCTION TO DIGITAL CONVOLUTION

Finite digital convolution has many powerful applications in digital signal processing. It is used to implement non-recursive or finite impulse response digital filters. It is also used to carry out auto and cross correlation as well as for computation such as polynomial multiplication [4].

3.1.1 Finite Linear Convolution

Finite linear discrete convolution of two sequences is mathematically represented as

$$y(n) = \sum_{m=0}^{N_1+N_2-1} h(n-m)\, x(m) \qquad n=0,1,2,\ldots(N_1+N_2-1) \tag{3.1}$$

where x(n), h(n) and y(n) are the finite digital sequences of length N_1, N_2 and N_1+N_2-1 respectively. Fig. 3.1 shows a simple pictorial representation of how linear convolution is carried out in practice. Fig. 3.1(a) shows a typical sequence x(n) that is non-zero in the range $0 \leq n \leq 4$. Fig. 3.1(b) shows the sequence h(n) that is non-zero for $0 \leq n \leq 7$. Fig. 3.1(c) shows the mirror image of h(n) along the y-axis. Fig. 3.1(d) to (f) show simultaneous plots of x(m) and h(n-m) for n=1,4,11. Clearly for n < 0 and n > 11, there is no overlap between x(m) and h(n-m), therefore y(n) is exactly zero. Finally Fig. 3.1(g)

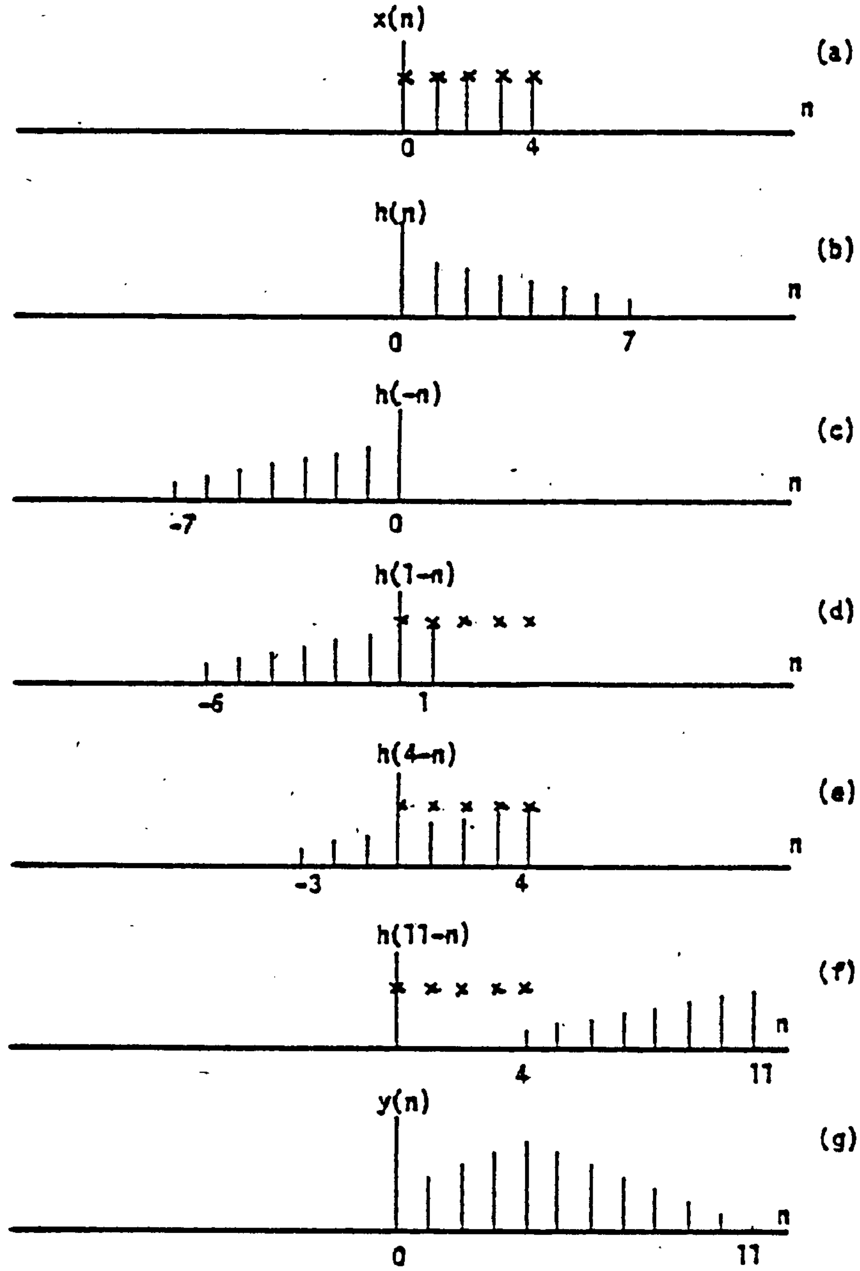


Fig. 3.1 Explanation of linear convolution

shows $y(n)$, which is the desired convolution.

3.1.2 Periodic or Cyclic Convolution

If $h(n)$ represents one period of the periodic sequence $h_p(n)$, and $x(n)$ represents that of $x_p(n)$, of both period N samples, then the periodic or cyclic convolution of $h(n)$ and $x(n)$ is defined as

$$y(n) = \sum_{m=0}^{N-1} x(m)\ h|n-m|_N \quad \text{for } n = 0,1,..,N-1 \tag{3.2}$$

and is represented as $y(n) = x(n) * h(n)$. Because of the periodicity, sequences $x_p(n)$ and $h_p(n-m)$ are considered only in the interval $0 \leq m \leq N-1$.

As the samples of $h_p(n-m)$ slide past $m=N-1$, the identical samples appear at $m=0$. Thus the term cyclic convolution is a description of the convolution of two sequences defined on a circle. When two periodic sequences are convolved, the output sequence is periodic and of the same period.

3.1.3 Linear Convolution Via Cyclic Convolution

Consider two finite duration sequences $x(n)$ and $h(n)$. The duration of $x(n)$ is N_1 and the duration of $h(n)$ is N_2. The linear convolution of $x(n)$ and $h(n)$ yields the sequence $y(n)$ of duration N_1+N_2-1. To obtain this sequence using cyclic convolution, both input sequences should also be of period N_1+N_2-1. Zeros can be appended to these input sequences to make them of duration N_1+N_2-1 and then circular convolution can be used to obtain $y(n)$.

3.2 DISCRETE FOURIER TRANSFORM

Finite digital convolution can be implemented using transforms having the cyclic convolution property (ccp). The characteristics of these transforms are such that the transform of convolution in the time domain is equal to the term by term product in the transform domain.

One of the transforms that exhibit ccp is the Discrete Fourier Transform (DFT) and is given by

$$\text{DFT} \qquad X(k) = \sum_{n=0}^{N-1} x(n)\, W^{nk}, \quad k = 0,1,\ldots,N-1 \tag{3.3}$$

where $W = \exp\left(-j\,\frac{2\pi}{N}\right)$.

The inverse transform (IDFT) is given by

$$\text{IDFT} \qquad x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\, W^{-nk}, \; n = 0,1,\ldots,N-1 \tag{3.4}$$

Then the cyclic convolution property is given as

$$\begin{aligned} &\text{If} \quad y(n) = x(n) \; (*) \; h(n) \\ &\text{then} \quad Y(k) = X(k) \,.\, H(k) \end{aligned} \tag{3.5}$$

where X, H and Y are the respective transforms of x, h and y.

To prove the ccp of DFT, take

$$y_p(n) = \sum_{l=0}^{N-1} x_p(l)\, h_p(n-l) \tag{3.6}$$

Take the transform of both sides of equation (3.6)

$$Y_p(k) = \sum_{n=0}^{N-1} \left\{ \sum_{l=0}^{N-1} x_p(l)\, h(n-l) \right\} e^{-j\frac{2\pi}{N} \cdot nk}$$

$$= \sum_{l=0}^{N-1} x_p(l) \underbrace{\{ \sum_{n=0}^{N-1} h(n-l) \cdot e^{-j \frac{2\pi}{N} (n-l) \cdot k} \}}_{H_p(k)} e^{-j \frac{2\pi}{N} lk}$$

$$= H(K) \cdot \underbrace{\sum_{l=0}^{N-1} x_p(l) \; e^{-j \frac{2\pi}{N} \cdot lk}}_{X_p(k)}$$

or

$Y_p(k) = X_p(k) \cdot H_p(k)$ which is the desired result.

Using the ccp of DFT, convolution can be implemented in the following way

i) take the DFT of both the input sequences

ii) obtain the term by term product in transform domain

iii) perform the inverse DFT to obtain the output sequence.

The block diagram of Fig. 3.2 shows the complete procedure to perform convolution.

3.3 FAST FOURIER TRANSFORM (FFT)

The term FFT refers to a number of algorithms that employ a number of methods for reducing the computation time required to compute a DFT. They make use of the symmetry and periodicity of the exponential factors, W, used in the defination of DFT, to de-

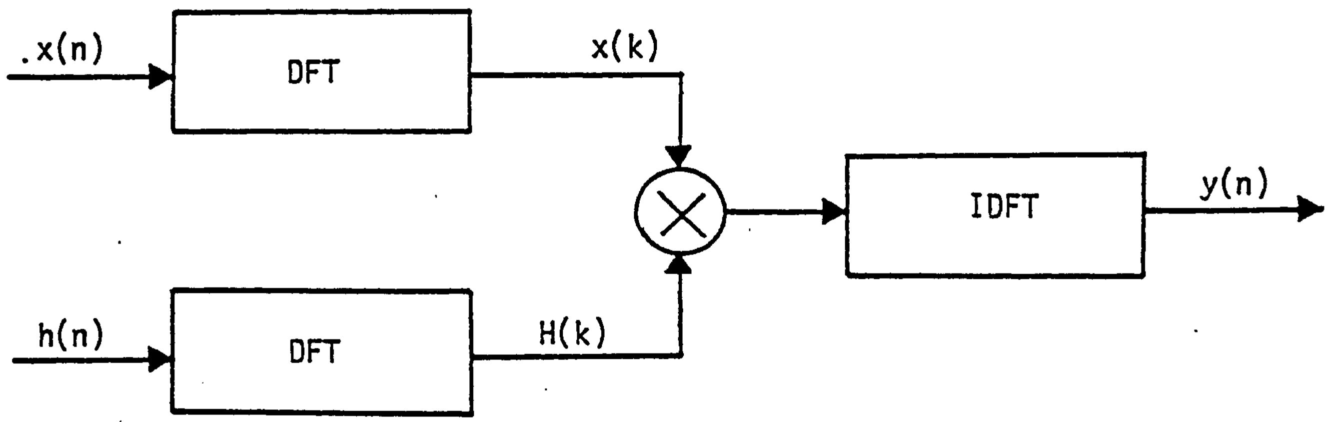


Fig. 3.2 Convolution using DFT method

compose a long DFT computation into smaller length DFT computations. To compute an N point DFT, a total of $(N-1)^2$ complex multiplications and N(N-1) additions are required while using the FFT for the same transform requires approximately $\frac{N}{2}\log_2 N$ multiplication and $N \log_2 N$ addition for radix 2 algorithm. Basically there are two types of FFT algorithms, called decimation in time (DIT) and decimation in frequency (DIF).

3.3.1 Decimation in Time Algorithm (DIT)

The algorithm in which the input sequence (time domain) is decomposed into smaller sequences is called a DIT algorithm. The procedure is illustrated for an N point sequence where $N = 2^r$, r is an integer.

By definition:

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

Define two $\frac{N}{2}$ point sequences $x_1(n)$ and $x_2(n)$ as the even and odd members of x(n).

$$\begin{aligned} x_1(n) &= x(2n) \\ x_2(n) &= x(2n+1) \end{aligned} \qquad n = 0,1,2,\ldots,\frac{N}{2}-1$$

Then N-point DFT is

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

$$\text{where } W_N^2 = e^{-j\frac{2\pi}{N}\cdot 2} = e^{-j\, 2\pi/N/2} = W_{N/2}$$

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

$$= X_1(k) + W_N^k\, X_2(k)$$

where $X_1(k)$ and $X_2(k)$ are $\frac{N}{2}$ point DFT's, and of period $\frac{N}{2}$. Therefore,

$$X(k) = X_1(k) + W_N^k\, X_2(k) \qquad 0 \le k \le \frac{N}{2}-1$$

$$= X_1(k-\tfrac{N}{2}) + W_N^k\, X_2(k-\tfrac{N}{2}) \qquad \frac{N}{2} \le k \le N-1 \quad .$$

As mentioned, for direct evaluation of an N point DFT, N^2 multiplications are required. Similarly, direct evaluation of an $\frac{N}{2}$ point DFT, requires $(\frac{N}{2})^2$ multiplications. If the above procedure is used to compute an N point DFT, a total of

$(\frac{N}{2})^2 \cdot 2 + N$ multiplications are required and

for $\frac{N^2}{2} >> N$ approximately $\frac{N^2}{2}$ multiplication are required and

a 50% saving over the direct evaluation of an N point DFT is obtained for a reasonably large N. The procedure is repeatedly applied to each of the successive subsequences, until only two point DFT's are left to be evaluated.

A flow graph representing the basic operation of the decimation in time algorithm is called a butterfly and has inputs A and B that are combined to give two outputs x and y via the operation

$$x = A + W_N^k B$$

$$y = A - W_N^k B.$$

Fig. 3.3 shows the butterfly unit and Fig. 3.4 shows the flow graph for 8 point DIT algorithm.

3.3.2 Decimation in Frequency Algorithm DIF

In this version of the FFT, the input sequence x(n) is partitioned into two sequence each of length $\frac{N}{2}$ in the following manner. The first sequence $x_1(n)$ consists of first $\frac{N}{2}$ points of x(n) and the second sequence $x_2(n)$ consists of the last $\frac{N}{2}$ points of x(n). Thus

$$x_1(n) = x(n) \qquad n = 0,1,2,\ldots,\frac{N}{2}-1$$

$$x_2(n) = x(n + \frac{N}{2}) \qquad n = 0,1,2,\ldots,\frac{N}{2}-1.$$

The N- point DFT of x(n) is then

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

$$= \sum_{n=0}^{\frac{N}{2}-1} (x_1(n) + e^{-j\,\pi k}\, x_2(n))\, W_N^{nk}$$

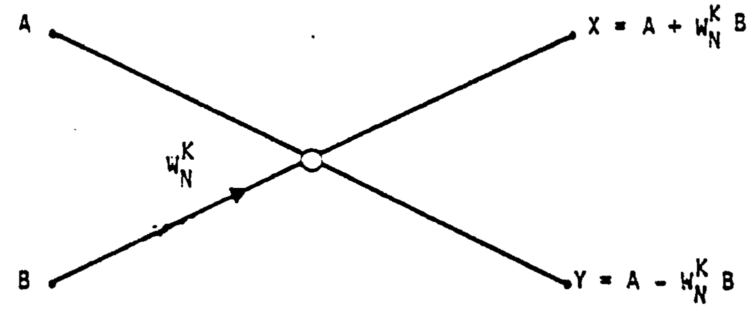


Fig. 3.3 2 point butterfly (DIT)

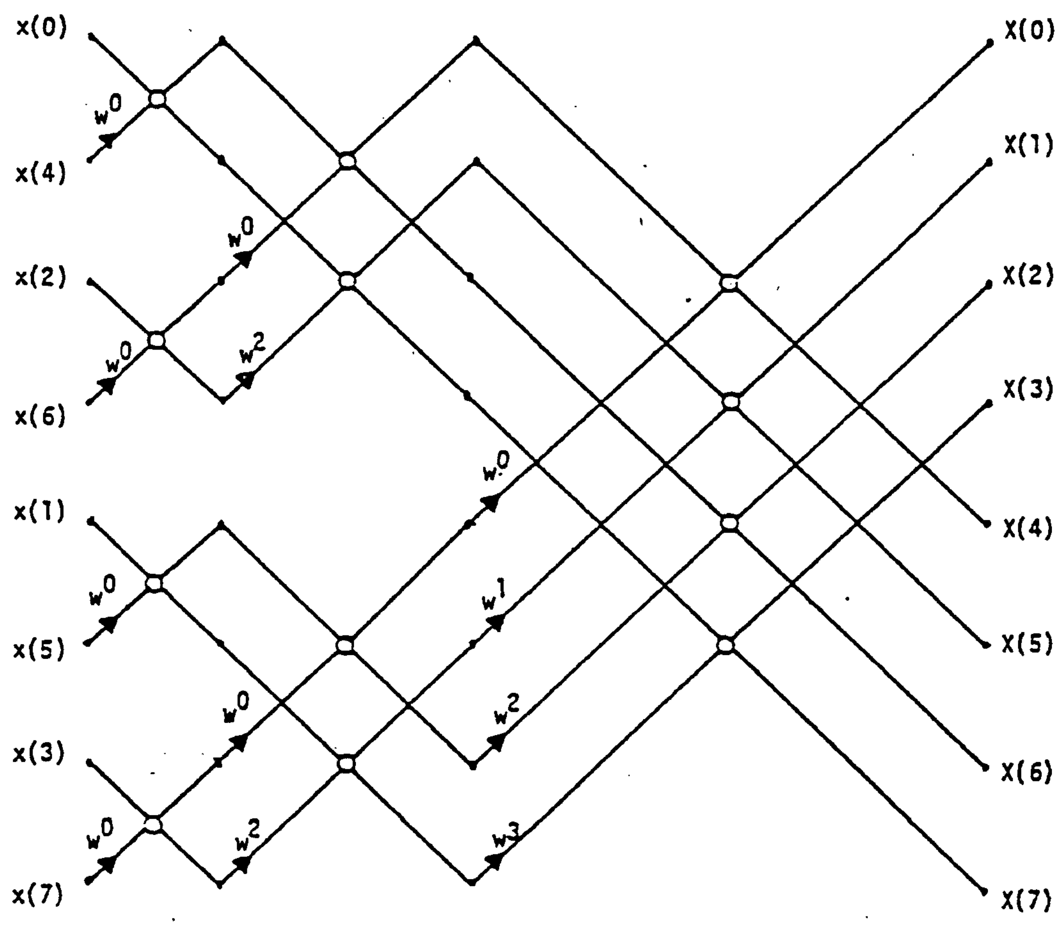


Fig. 3.4 Eight point butterfly (DIT)

Decompose X(k) into even and odd sample sequence

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

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} (x_1(n) + x_2(n))\, W_{N/2}^{nk} \tag{3.7}$$

and

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

$$= \sum_{n=0}^{\frac{N}{2}-1} \{(x_1(n) - x_2(n)\, W_N^{n}\}\, W_{N/2}^{nk} \tag{3.8}$$

(3.7) and (3.8) are equivalent to two $\frac{N}{2}$ points DFT's. The procedure is repeatedly applied to each of the even and odd samples output subsequences until finally two point DFT's are left to be evaluated. Fig. 3.5 shows the butterfly unit and Fig. 3.6 shows the flow graph for 8 point DFT using DIF algorithm.

3.4 NUMBER THEORETIC TRANSFORM (NTT)

Agarwal and Burrus [5] have showed that the existence of an N point transform having the cyclic convolution property depends on the existence of a generator alpha (α) that is a root of unity of order N, and the existence of N^{-1}. In the complex number field, the DFT is the transform which exhibit cyclic convolution property with α equal to exp $(-j \frac{2\pi}{N})$.

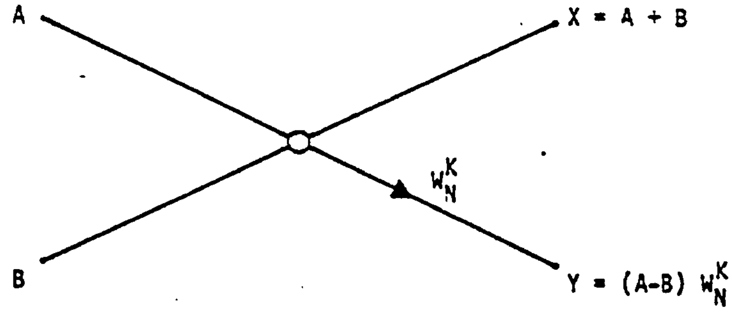


Fig. 3.5 2 point butterfly (DIF)

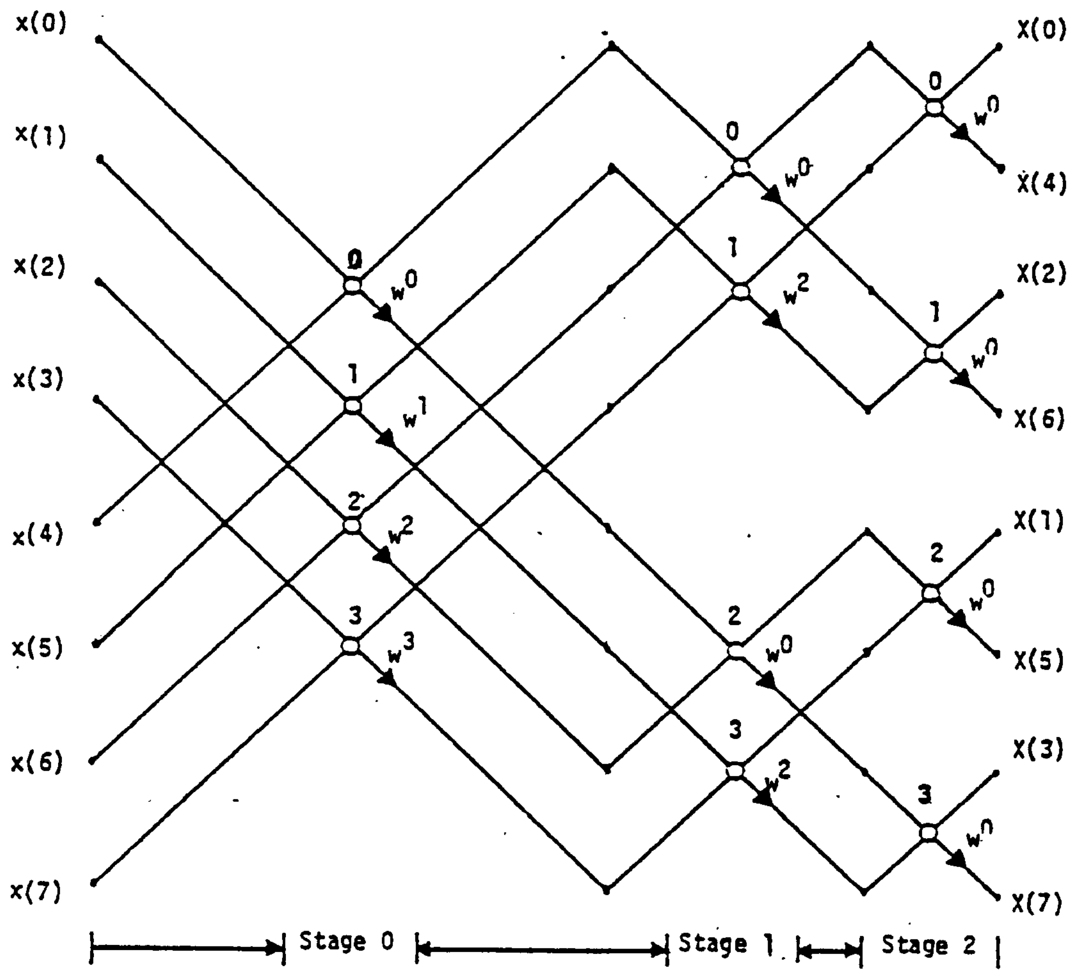


Fig. 3.6 Eight point butterfly (DIF)

It supports any length of the transform because of the variable periodicity of $\exp(-j \cdot 2\pi/N)$ but at the same time it involves multiplication by irrational coefficients (sines and cosines) making exact computation impossible on a digital machine. At each stage, the output has to be scaled down to avoid overflow thus requiring some kind of scaling operation and at the same time introducing extra computational errors.

Pollard [4] has shown that transforms defined in a finite ring or field exhibit the cyclic convolution property with a suitable choice of the ring or field and the appropriate alpha. These transforms are known as Number theoretic transforms (NTT) and defined as

$$X(k) = \left| \sum_{n=0}^{N-1} x(n)\, \alpha^{nk} \right|_M \tag{3.9}$$

and

$$x(n) = \left| N^{-1} \sum_{k=0}^{N-1} X(k)\, \alpha^{-nk} \right|_M \tag{3.10}$$

where N^{-1} belongs to the ring/field. Unlike the DFT, NTT's do not allow arbitrary transform lengths. The maximum attainable length N, depends upon the choice of the ring or field and alpha. Before discussing the choice of parameter, the invertibility and convolution property of NTT is established in the next section.

3.4.1 Invertibility and Convolution Property of NTT

If α is the root of unity of order N, which is one of the basic conditions for the existence of the NTT, then the following relation holds

$$|\alpha^{Nj}|_M - 1 = 0 \qquad j = \text{an integer} \tag{3.11}$$

which can be factored as:

$$(\alpha^{j} - 1) \sum_{p=0}^{N-1} \alpha^{pj} = 0 \tag{3.12}$$

Therefore

$$\sum_{p=0}^{N-1} \alpha^{pj} = N \qquad \text{if } j \equiv 0 \bmod N$$

$$\sum_{p=0}^{N-1} \alpha^{pj} = 0 \qquad \text{otherwise} \tag{3.13}$$

since for $j \neq 0$ $\alpha^{j} - 1 \neq 0$

Invertibility

Assuming all the operations are performed mod m, substituting (3.9) into (3.10) and using (3.13)

$$x(n) = N^{-1} \sum_{k=0}^{N-1} X(k)\, \alpha^{-nk} = N^{-1} \sum_{k=0}^{N-1} \sum_{u=0}^{N-1} x(u)\, \alpha^{uk} \cdot \alpha^{-nk}$$

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

and hence the invertibility of NTT is proved.

Convolution

Let $X(f) = \sum_{t=0}^{N-1} u(t)\, \alpha^{ft}$

$$H(f) = \sum_{v=0}^{N-1} h(v)\, \alpha^{ft}$$

$$Y(f) = X(f) \cdot H(f)$$

Then, by (3.10), the inverse transform of Y(f)

$$y(s) = N^{-1} \sum_{f=0}^{N-1} X(f) \cdot H(f) \cdot \alpha^{-fs}$$

$$= N^{-1} \sum_{f=0}^{N-1} \sum_{t=0}^{N-1} \sum_{v=0}^{N-1} x(t)h(v)\ \alpha^{f(v+t-s)}$$

$$= N^{-1} \sum_{t=0}^{N-1} x(t)h(s-t) \cdot N = \sum_{t=0}^{N-1} x(t)h(s-t)$$

Since the summation $\sum_{f} \sum_{v} \alpha^{f(v+t-s)}$ is modulo N, hence this is the cyclic convolution and the CCP is proved.

3.5 CHOICE OF THE PARAMETERS FOR THE NTT

Practical considerations dictate a selection of ring/field that supports a transform whose parameters lead to efficient implementation of modular arithmetic, either in hardware or software. Most of the reported work on the NTT has supposed that the hardware will be implemented using the binary number system. In the conventional binary arithmetic, residue reduction is particularly easy when the modulus can be represented as power of two. Also multiplication by α will be simpler if α is also a power of two. In that case multiplication by α reduces to bit shifting. These restrictions severly limit the maximum attainable transform length.

We are interested in the implementation of NTT using ROM arrays and therefore the moduli and generators can be selected purely on number theoretic basis to maximize the transform length. The following definitions and theorems will be helpful in determining the attainable transform length for different moduli.

Definition 1: The Euler's totient function $\Psi(M)$ is defined as the number of integer in Z_M that are relative prime to M, e.g., for M = 5 $\Psi(5) = 4$.

Definition 2: For M a prime number $\Psi(M) = M-1$.

Definition 3: If M can be represented as $M = p_1^{r_1} \cdot p_2^{r_2} \ldots p_n^{r_n}$ where p_i's are primes than $\Psi(M) = M(1 - \frac{1}{p_1})(1 - \frac{1}{p_2}) \ldots (1 - \frac{1}{p_n})$.

Theorem 1: Euler's theorem states that the maximum order of an element in Z_M is $\Psi(M)$.

The implications of Euler's theorem are that maximum order of α in the ring Z_M is $\Psi(M)$ that is $\alpha^{\Psi(M)} = 1$ or the maximum value of transform length in Z_M is $\Psi(M)$. Mathematically $N_{max} = \Psi(M)$ and the allowed transform lengths should divide $\Psi(M)$.

Consider the case when M is even, then it contains a factor of 2 and therefore the maximum transform length is one, which is practically useless. This implies that M can not be taken as a multiple of two.

Next take the case when M is odd and represented as $2^k - 1$. Let k be composite and represented as pQ, with p prime, then $2^p - 1$ divides $2^{pQ} - 1$ and the maximum transform length is $2^p - 1$. Therefore only prime values of k need to be considered. These numbers are known as Mersenne numbers. Radar [6] has proposed transforms defined in the ring of integers modulo Mersenne number. These transform are referred to as Mersenne Number Transform (MNT).

It has been shown that transform of length 2p exists and the corresponding α is -2. The disadvantage of this multiplication free MNT is that the transform length is not a power of 2 and not even highly composite and therefore fast FFT-type computational algorithm can not be used.

For $M = 2^k + 1$ and k odd, 3 divides 2^{k+1} and the maximum transform length is 2. Consider k even and let $k = s \cdot 2^t$ where s is odd. Then $2^{2^t} + 1$ divides $2^{s.2^t} + 1$ and the length of the possible transform will be governed by $2^{2^t} + 1$. Therefore, integers of the form $2^{2^t} + 1$ are of interest. These numbers are known as Fermat numbers. Agarwal and Burrus [7] proposed transforme defined in the ring of integers modulo Fermat number. These transforms are referred to as Fermat number transforms. Fermat numbers up to F_4 are primes. In [7], it has been shown that an FNT with $\alpha = \sqrt{2}$ allows $N = 2^{t+2}$.

However the main disadvantage of the MNT and FNT is the rigid relationship between the dynamic range and attainable transform length. For example, with a 32 bit word machine using $F_5 = 2^{32} + 1$, N = 128 for $\alpha = \sqrt{2}$. There is also a limited choice of possible word lengths.

Other authors have used different fields but still the transform length is severly limited. The solution to this problem is found by computing the transforms over extension fields.

3.5.1 Transforms Defined Over Galois Fields

Definition 4: For any prime m and any positive integer n, there exists a finite field with m^n elements. This unique field is commonly denoted by the symbol $GF(m^n)$ and is called a Galois field. Any finite field with m^n elements is a simple algebric extension of the field Z_m.

Let F be a field. Then any field K containing F is an extension of F. If λ is a root of some irreducible polynomial $f(x) \varepsilon F[x]$ such that $f(\lambda) = 0$, then the extension field arising from a field F by the adjunction of a root λ is called a simple algebric extension,denoted by $F(\lambda)$. Each element of $F(\lambda)$ can be uniquely represented as a polynomial.

$$a_0 + a_1 \lambda + \ldots . a_{n-1} \lambda^{n-1}, \; a_i \; \varepsilon \; F.$$

The field of complex numbers is an example of an extension of the field of real numbers, it is generated by adjoining a root $j = \sqrt{-1}$ of the irreducible polynomial $x^2 + 1$.

If $f(x)$ is an irreducible polynomial of degree n over Z_m, m prime, then the Galois field with m^n elements GF (m^n) is defined as the field of residue class of polynomial of Z_m [x] reduced modulo $(f(x))$.

Pollard [4] has shown that transforms of the form

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

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

defined over the Galois fields of m^n elements, where m is a prime, also exhibit ccp. The maximum attainable transform length is given by $N_{max}=m^n - 1$ with restriction that α is cyclic of order N in GF (m^n).

Thus the extension fields allow a greatly increased transform length for the same value of m and the problem of obtaining large transform length is resolved. In order that the implementation of efficient, two things must be considered after the choice of m and N:

(i) construct the Galois field GF (m^n) such that the multiplication and addition of field elements require the smallest possible number of operations;

(ii) search for the generator of an N element cyclic sub-group in GF (m^n), α, that has the simplest form possible so that the number of operations required for multiplications by powers of α are minimized.

3.5.2 Construction of Galois Field GF (m^n)

To construct a Galois field of m^n elements, first an irreducible polynomial is to be formed. The form of the irreducible polynomial dictates the complexity of the computation in the field since addition and multiplication is defined as the polynomial addition and multiplication, followed by polynomial reduction modulo $f(x)$. We restrict our interest to GF of 2nd degree as they still offer simple hardware implementation and provide transform lengths which are quite suitable for practical purposes. We take the two cases of irreducible polynomial and find out the complexity of the computation.

Case 1: Let $f(x) = x^2 + x + 1$ be an irreducible polynomial of degree 2 over GF (m). Then, the extension field in which the given polynomial has a root, denoted by w, may be described by

$$GF(m^2) = \{a + bw \mid a, b \in GF(m)\}$$

and $\qquad w^2 + w + 1 = 0$ in $GF(m^2)$.

Take the multiplication of two elements of the field

$$|(a + bw) \cdot (a' + b'w)|_m$$

$$= |(aa' + bb'w^2) + w(ab' + a'b)|_m$$

Dividing the result by $w^2 + w + 1$

$$= |(a'a - bb') + w(ab' + a'b - bb')|_m$$

Thus multiplication of field elements require 4 binary multiplications and three binary additions.

<u>Case 2:</u> Let $f(x) = x^2 - r \quad r \in GF(m)$. Then the extension field, in which the given polynomial has a root is described by

$$GF(m^2) = \{a + \lambda b \mid a, b \in GF(m)\}$$

and $\qquad \lambda^2 - r = 0.$

Multiplication of two elements is now performed as

$$(a + b\lambda) \cdot (a' + b'\lambda) = |(aa' + bb'\lambda^2) + \lambda(ab' + a'b)|_m$$
$$= |(aa' + rbb') + \lambda(ab' + a'b)|_m$$

Residue reduction mod $(\lambda^2 - r)$ is simple since $\lambda^2 = r$. Multiplication of field elements require 4 binary multiplication and 2 addition. Since ROM arrays will be used for the implementation of NTT, multiplication by constant, r does not require separate stage. Because of the simplicity of the $(x^2 - r)$ polynomial, it is used to construct the 2nd order Galois field.

After the structure for irreducible polynomial has been decided, the next problem is to find a suitable value of r such that $x^2 \equiv r \pmod{m}$ is not solvable in GF(m).

Baraniecka [8] has described a complete procedure for finding the values of r for different fields. Following is a brief discussion of the method presented in [8].

All the prime numbers can be divided into two groups.

<u>4n + 1</u> e.g., 1, 5, 13, 17, ...

<u>4n + 3</u> e.g., 3, 7, 11, 19, ...

The most trivial value of r is -1 but for the case of 4n + 1 type primes, $\sqrt{-1}$ can be considered as a member of GF(m) and hence Galois fields of 2nd degree can not be constructed using the polynomial, $x^2 + 1$. For example, if m = 5, $\sqrt{-1}$ is congruent modulo 5 to 2 and 3. For 4n + 3 type primes, $\sqrt{-1}$ can be used to construct Galois fields of 2nd degree and $GF(m^2)$ is isomerphic to the residue class of complex, so called Gaussian integers. The elements of the field are defined as $a + \sqrt{-1}\, b$, $a, b \in GF(m)$. To find an irreducible polynomial for primes of 4n + 1 type, we make use of the following theorem.

<u>Theorem 2:</u> If g is a generator for the multiplicative group GF(m) - {0}, then $x^2 - g$ is an irreducible polynomial in GF(m).

For example, 13 is a prime of 4n + 1 type. It's generator of the cyclic group is 2. It can be easily verified that

$$x^2 \equiv 2 \bmod (13) \text{ has no solution}$$

or $x^2 - 2$ is an irreducible polynomial in GF(m). Hence a Galois field of 2nd degree can be constructed using $r = 2$. The elements of $GF(m^2)$ will be defined as $a + \sqrt{2}\, b$, $a, b \in GF(m)$.

3.5.3 Searching for the Generator α in $GF(m^2)$

We first summarize what has been presented so far:

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

(ii) choose the prime which will give this transform length over a Galois field of 2nd degree

(iii)construct the 2nd order fields in which binary operations are simpler.

The next problem is now to find out the generator, α, which is of the order N in $GF(m^2)$. To search for the generator α for $4n + 3$ type, the following theorem is stated. The prime, $m_i = 4n + 3$, can be represented as $m_i = q \cdot 2^p - 1$ with q odd.

Theorem: Given a base field Z_m and an irreducible polynomial $x^2 - r$ over GF(m), the extension field $Z_m(\sqrt{r})$ has a cyclic subgroup of order $N = 2^B$. The maximum value of B is P + 1. The generator α has the form $\beta + \gamma \sqrt{r}$.

For $4n + 3$, a prime $m_i = r$ can be taken as -1 and hence the general form of α is $\beta + \gamma\sqrt{-1}$. Transforms over $GF(m^2)$ with $r = -1$ can be used to compute convolution on complex data or convolution on two blocks of real data.

Example: For $m = 7 = 1.2^3 - 1$ the maximum radix two transform over $GF(7^2)$ is $N = 2^{p+1} = 16$. α for this prime can be chosen to be

$2 + 3\sqrt{-1}$ and it can be verified that the generator has order 16. Other values of α are also possible and can be used for the transform. $4n + 1$ primes can be represented as $m = q \cdot 2^p + 1$ where $(q,2) = 1$. The largest possible radix 2 transform length in $GF(m^2)$ is $N = 2^{p+1}$. For primes of this form, the generator has a simple form $\alpha = \sqrt{r}$ where $x^2 - r$ is an irreducible bionomial in $GF(m^2)$. This property is obtained from the following theorem.

Theorem 3: Let $m = q \cdot 2^k + 1$, be an odd prime number. Then:

i) If g is generator for the multiplicative group $GF(m) - \{0\}$, then $x^2 - g$ is an irreducible polynomial in $GF(m)$.

ii) If g is as in (i), then $\sqrt{g}$ has multiplicative order $q \cdot 2^{k+1}$ in $GF(m^2)$ where elements are given as $a + b\sqrt{g}$ $a, b \in GF(m)$.

iii)We can find a generator $\sqrt{r}$, of a cyclic subgroup or order 2^{k+1} in $GF(m^2)$ where $r = g^{eq}$ with $(e,2) = 1$ and $x^2 = r$ an irreducible polynomial in $GF(m)$.

Example: Let the prime be $m = 97 = 3.2^5 + 1$. Maximum radix 2 transform length over $GF(97^2)$ is $N = 2^6 = 64$. From the tables of the primitive roots, it can be found out that for the prime 97, g = 5.

According to theorem 3, $\sqrt{5}$ will generate a cyclic subgroup of order 192, and the generator of the multiplicative order 64 is given by $\alpha = \sqrt{r} = (\sqrt{5})^{3e}$ where $(e,2) = 1$. Arbitrarily choosing $e = 1$, $\alpha = \sqrt{28}$, it can be verified that this α has order of 64 in $GF(97^2)$.

3.6 NTT USING RNS CONCEPTS

From the previous discussion, it can be seen that the NTT defined over 2nd order Galois fields, yields a practicable transform length and these 2nd order fields can be constructed using polynomials for which the binary operation in $GF(m^2)$ is simplest. Although the transform length achieved is large enough for practical purposes, dynamic range is still severely limited. This problem can be solved using RNS concepts. The NTT can be performed over different Galois fields and then the final result can be reconstructed using the chinese remainder theorem or a mixed radix conversion scheme [8]. Thus computing the transform over a finite ring which is isomorphic to a direct sum of several Galois field of 2nd degree, $R = GF(m_1^{\,2}) + \ldots\ldots + GF(m_n^{\,2})$ increases the dynamic range to $\prod_{i=1}^{n} m_i$. The conditions for the existence of the NTT over the finite ring can now be restated.

i) For each m_i, α_i must be a primitive Nth root of unity in $GF(m_i^{\,2})$

ii) $N \mid (m_i^{\,2} - 1)$ $i = 1,2,..,n$ or in other words $N \mid \gcd(m_i^{\,2} - 1)$, $i = 1,\ldots,n$

As a practical example, assume a transform length of 32 points is required. The prime moduli 17, 31 and 47 can be used and the dynamic range is then given by their product $17 \times 31 \times 47 \approx 2^{14.65}$ and therefore a word length of approximately 14 bits is achieved. These are not the only choice of primes. Other primes can also be used for the same transform length but which will give different dynamic ranges.

Table 3.1 shows the primes and the maximum transform length that can be achieved using these primes. It may be noted that for any transform length N and the generator α, the transform length is halved if α is raised to power two, for example, for prime 193

Primes m_i	Factorization of m_i-1	Factorization of m_i^2-1	Maximum Radix 2 Length in $GF(m_i)$	Maximum Radix 2 Length in $GF(m_i^2)$
3	2	2^3	2	8
5	2^2	3.2^3	4	8
7	3.2	3.2^4	2	16
11	5.2	5.2^3	2	8
13	3.2^2	$7.3.2^3$	4	8
17	2^4	$3^2.2^5$	16	32
19	$3^2.2$	$5.3^2.2^3$	2	8
23	11.2	$11.3.2^4$	2	16
29	7.2^2	$7.5.3.2^3$	4	8
31	5.3.2	$5.3.2^6$	2	64
37	$3^2.2^2$	$19.3^2.2^3$	4	8
41	5.2^3	$7.5.3.2^4$	8	16
43	7.3.2	$11.7.3.2^3$	2	8
47	23.2	$23.3.2^5$	2	32
53	13.2^2	$13.3^3.2^3$	4	8
59	29.2	$29.5.3.2^3$	2	8
61	$5.3.2^2$	$31.5.3.2^3$	4	8

Table 3.1 TABLES OF FIRST FEW PRIMES AND THE ASSOCIATED TRANSFORM LENGTH.

the maximum transform length is 128 and the corresponding α is $\sqrt{125}$. This same prime can be used for transform length 64 and the α would be 125, for N = 32, α = 185 and so on. Thus for the smaller transform length, large primes can be used to provide large dynamic range.

Fig. 3.7 shows a conceptual block diagram to implement an NTT using the RNS. At the first stage a distributor is required which can feed the data modulo respective primes to different units. Each prime requires a supporting memory structure and a computational unit. The advantage of using RNS is that the computation can be performed in parallel and the speed of operation does not depend upon the number of primes used and hardware is the only limitation on the number of primes to be used. After the computation, the final result of the transform can be reconstructed in a reconstruction stage, using the ch. rem. theorem or mixed radix conversion.

3.7 SUMMARY

In this chapter, the implementation of convolution using transform technique has been discussed. It was shown that certain transforms exhibit cyclic convolution property and can be used to implement circular or linear convolution. The general structure of these transforms is

$X(k) = \sum_{n=0}^{N-1} x(n)\,\alpha^{nk}$ where α is the Nth root of unity and N is the transform length. In a complex number field for $\alpha = e^{-j\cdot\frac{2\pi}{N}}$, the transform is known as the DFT and exhibits the cyclic convolution property. The main disadvantage of the DFT is the muliplication by

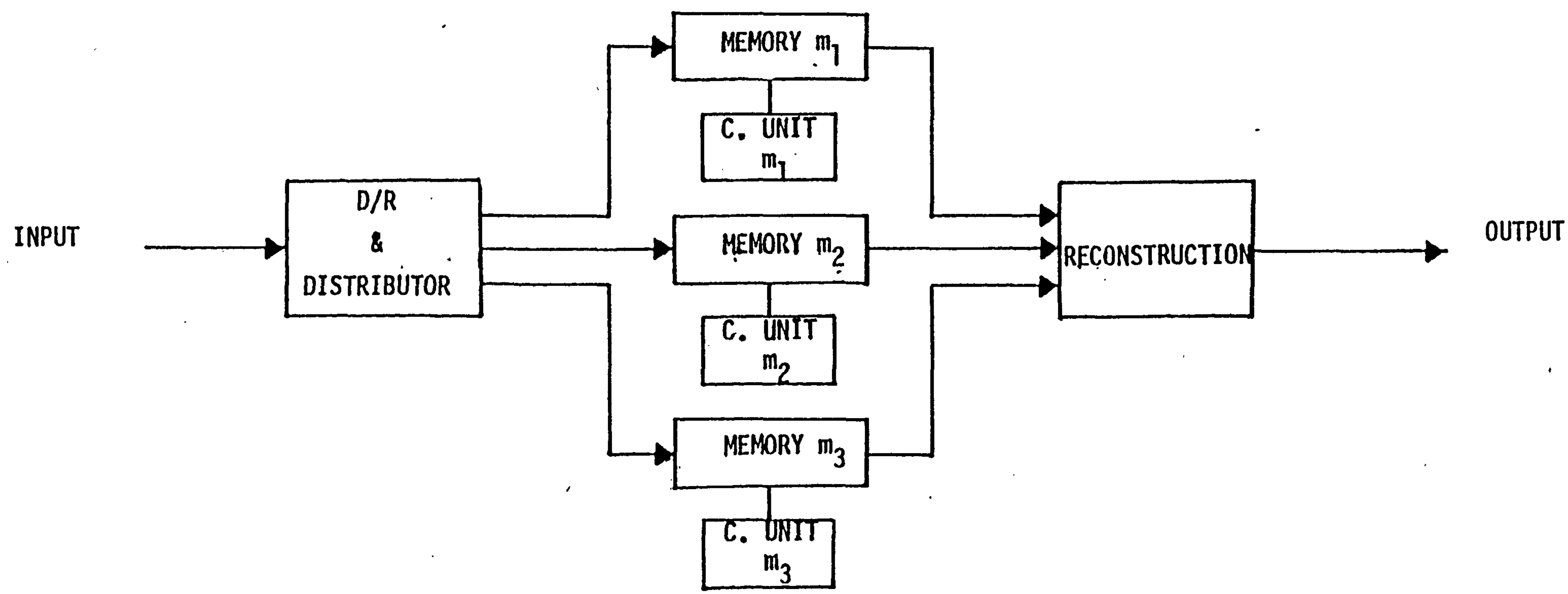


Fig. 3.7 Implemention of NTT using RNS for three moduli.

irrational coefficients, thus making it impossible to compute the transform exactly using binary arithmetic.

It was shown by different authors that the NTT defined over finite rings or fields also exhibit the ccp for suitable α. It was assumed that these transforms will be implemented using binary arithmetic and thus stress was given to the field for which residue reduction was simpler. α was chosen to have a simpler form preferably a power of 2 so that multiplication by α reduces to bit shifting. This severly restrictes the choice of ring/field and also α can not be chosen to yield the maximum transform length. In this chapter it has been assumed that the NTT will be implemented using ROM arrays and therefore the moduli and α can be chosen freely to obtain the maximum transform length. A ROM array implementation still did not allow a suitable large transform length in GF of 1st degree and therefore GF of 2nd degree were introduced. The implementation of NTT in GF of 2nd degree were discussed and also it was shown that using GF of 2nd degree increases the transform length to more than the square of the transform length in the 1st degree fields. The use of 2nd degree field, though increasing the transform length, does not solve the problem of dynamic range. For an increased dynamic range, large moduli were to be used, which are not efficient for hardware implementation. This problem is solved through the use of the RNS by computing transform in parallel,modulo several primes, $\{m_i\}$, so that the dynamic range is given by $M = \prod_{i=1}^{n} m_i$.

In summary, the following procedures may be followed for selecting the parameters of NTT.

(1) Choose the desired transform length N for the particular application.

(2) Find the dynamic range required for the particular application.

(3) Depending upon the dynamic range and transform length, choose the suitable prime. For $N > 64$ and for large dynamic range requirements, it is more efficient to go for the 2nd degree fields.

(4) Construct the 2nd order fields using a simple form of irreducible polynomial.

(5) Find out the generator α, which has the simplest form and have an order of N.

The complete discussion on choosing these parameters was presented in this Chapter. The above procedure is a tentative procedure and the final choice of the parameter is dictated by the efficient hardware realization and the cost of the system. In the following chapter, a detailed discussion on efficient hardware realization will be presented.

CHAPTER 4

IMPLEMENTATION OF AN NTT BUTTERFLY

4.1 INTRODUCTION

The NTT processor mainly consists of a supporting memory structure and a computational unit commonly known as the butterfly unit. The main aim of the work presented, is to realize the butterfly unit in hardware, compatible with the memory structure used with the NTT processor.

In this chapter the design of the NTT butterfly is developed. The associated memory structure to support the NTT butterfly is discussed as required but the actual hardware design of the memory structure is not undertaken. A multiplexed butterfly unit was designed for hardware implementation, using look up tables and the pipeline configuration, for real time applications. A detailed simulation of the basic required memory structure and the butterfly unit designed for hardware implementation was done. After the verification of the simulation results, the butterfly unit was implemented in hardware using look up tables stored in Eproms. The only control required to run the butterfly unit is a clock pulse and a circuitry was designed and built for generating control pulses.

4.2 NTT PROCESSOR

The NTT has the same structure as the DFT and therefore for highly composite transform length N, the fast algorithm for computing the DFT can also be used to compute the NTT. Analogous to the FFT, the fast algorithm to compute the NTT will be called FNTT. Usually a sequential type processor is used to compute the transform, which saves hardware at the cost of slowing down the speed of computation. A multiplexed radix r butterfly is used as a computational unit with some supporting memory structure. This butterfly is accessed $\frac{N}{r} \times \log_r N$ times where r is the radix of the FNTT algorithm and N is the transform length. A conceptual block diagram of the NTT processor is shown in Fig. 4.1. The supporting memory is used to store the input data and the intermediate results of the computation. A control unit is also required to control the data flow to and from the memory, to keep track of stage of computation and the position of the butterfly in that stage.

4.2.1 Memory Structure

A great deal of literature is available for the memory organization of a FFT processor and is equally applicable to the FNTT. Pease [9] brought out an idea to use slow memory efficiently by splitting main memory into several sub-memories. Corinthois [10] used the idea presented by Pease and came up with an OIOO (ordered input-ordered output) algorithm which makes use of sequential memory.

A radix 2 butterfly unit requires a minimum of hardware and we restrict our interest to Radix 2 transform. The transform matrix can be represented as the product of matrices givesn by equation 4.1.

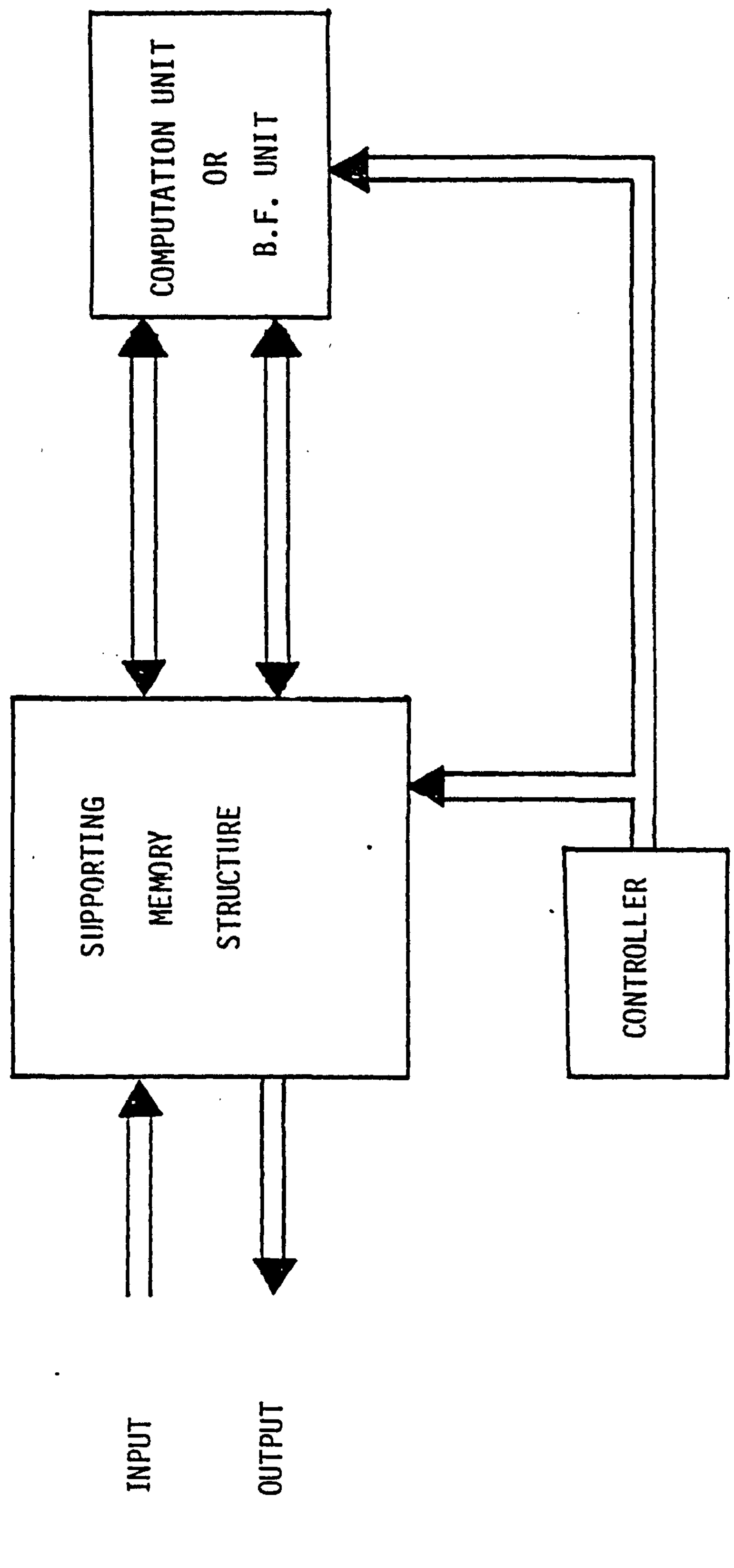


Fig. 4.1 Conceptual diagram of NTT processor

$$T_N = \prod_{m=n}^{1} p'_m u_m s \tag{4.1}$$

where $n = \log_2 N$ and $s = (I_{\frac{N}{2}} \times T_2)$ and x represents Kronecker product

and

$$T_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{4.2}$$

$$p'_i = I_{2^{i-1}} \times P_{\frac{N}{2^{i-1}}} \tag{4.3}$$

$$u_i = I_{2^{i-1}} \times D_{\frac{N}{2^{i-1}}} \tag{4.4}$$

$$p'_n = u_n = I_N \tag{4.5}$$

The operator, s, performs the two point transform on the input fed to the computational unit. The two point transform requires only addition and subtraction of the input data as is obvious from the operator s. The input data accessed from the memory are always $\frac{N}{2}$ points apart. The operator u performs multiplication by twiddle factors and p' is the permutation operator which shuffles the data to obtain the final output in ordered form.

This machine oriented algorithm requires two memory buffers, the input memory and the output memory, consisting of long shift registers and a computational unit. The input memory is divided into two sub-memories which store $\frac{N}{2}$ points. The one input from each sub-memory is fed to the computational unit and the output from the computational unit is stored in the output memory. After completion of each stage ($\frac{N}{2}$ butterfly computation), the data from the output memory is fed to the input memory and the shuffling on the data is performed as required by the operator p'_m in equation (4.1). A block diagram of the processor is shown in Fig. 4.2.

The main drawback to this kind of implementation is that each stage calls for a feedback phase in which data are serially moved from the output buffer to the input buffer in an order determined by the permutation operator. Corinthois [11] modified the above algorithm to eliminate the feedback process and the final form is given by the following equations.

$$T_N = \prod_{m=1}^{n} u_m \, s_m \tag{4.6}$$

where in general

$$s_{m-1} = s \, p_m \tag{4.7}$$

$$s_n = s \tag{4.8}$$

$$u_1 = I_N \tag{4.9}$$

where u and p have been defined earlier. In this algorithm, the operator s always calls for data that are at least $\frac{N}{4}$ words apart except at the first stage where they are $\frac{N}{2}$ words apart.

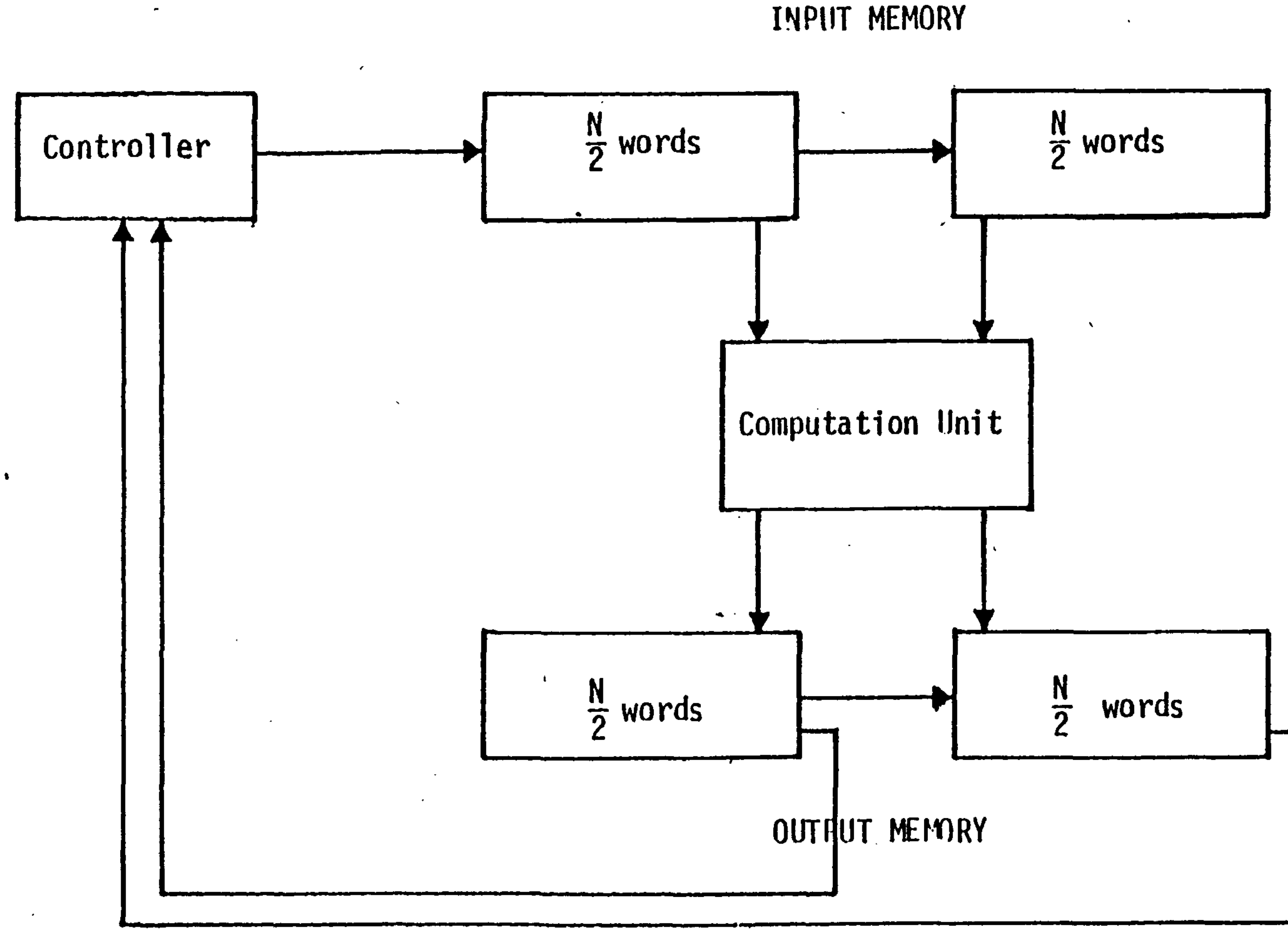


Fig. 4.2 Basic machine organization for OIOO.

Example 1: Consider the case when N=8. The matrix is given as:

$$T_8 = \prod_{m=1}^{3} u_m\, s_m = u_1 s_1\; u_2 s_2\; u_3 s_3 \tag{4.10}$$

since $u_1 = I_N$

$$T_8 = s_1 u_2\; s_2 u_3\; s_3 \tag{4.11}$$

and

$$s_3 = s = I_4 \times T_2 \tag{4.12}$$

The expansion of these matrices is shown in Fig. 4.3, and the flow graph implementing the transform is shown in Fig. 4.4. This algorithm does not require an extra feedback operation. The input and output memory consists of FIFOs and can be divided into 4 sub-memories which store $\frac{N}{4}$ words. The data flow can be handled according to the operator s_n at each stage.

A block diagram of the above processor is shown in Fig. 4.5. Assuming the input is already stored in MEMORY 1, the input data is fed to the computational unit and the output from the computational unit is then stored in MEMORY 2. After the first stage, the role of the memory is changed and MEM2 now becomes the input memory and MEM1 the output memory, and so on.

For real time applications, three memory buffers are required. While two buffers are being used for the computation, the third buffer can then be used to store the input sequence and also to supply the transformed sequence. A conceptual block diagram for a real time processor is shown in Fig. 4.6.

$$
\begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \end{bmatrix}
=
\overset{s_1}{\begin{bmatrix}
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & -1
\end{bmatrix}}
\overset{\mu_2}{\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \alpha^0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \alpha^0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \alpha^2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^2
\end{bmatrix}}
\overset{s_2}{\begin{bmatrix}
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\
0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & -1
\end{bmatrix}}
$$

$$
\overset{\mu_3}{\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \alpha^0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \alpha^1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \alpha^2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^3
\end{bmatrix}}
\overset{s_3}{\begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & -1
\end{bmatrix}}
\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix}
$$

Fig. 4.3 EXPANSION OF TRANSFORM MATRIX FOR 0100

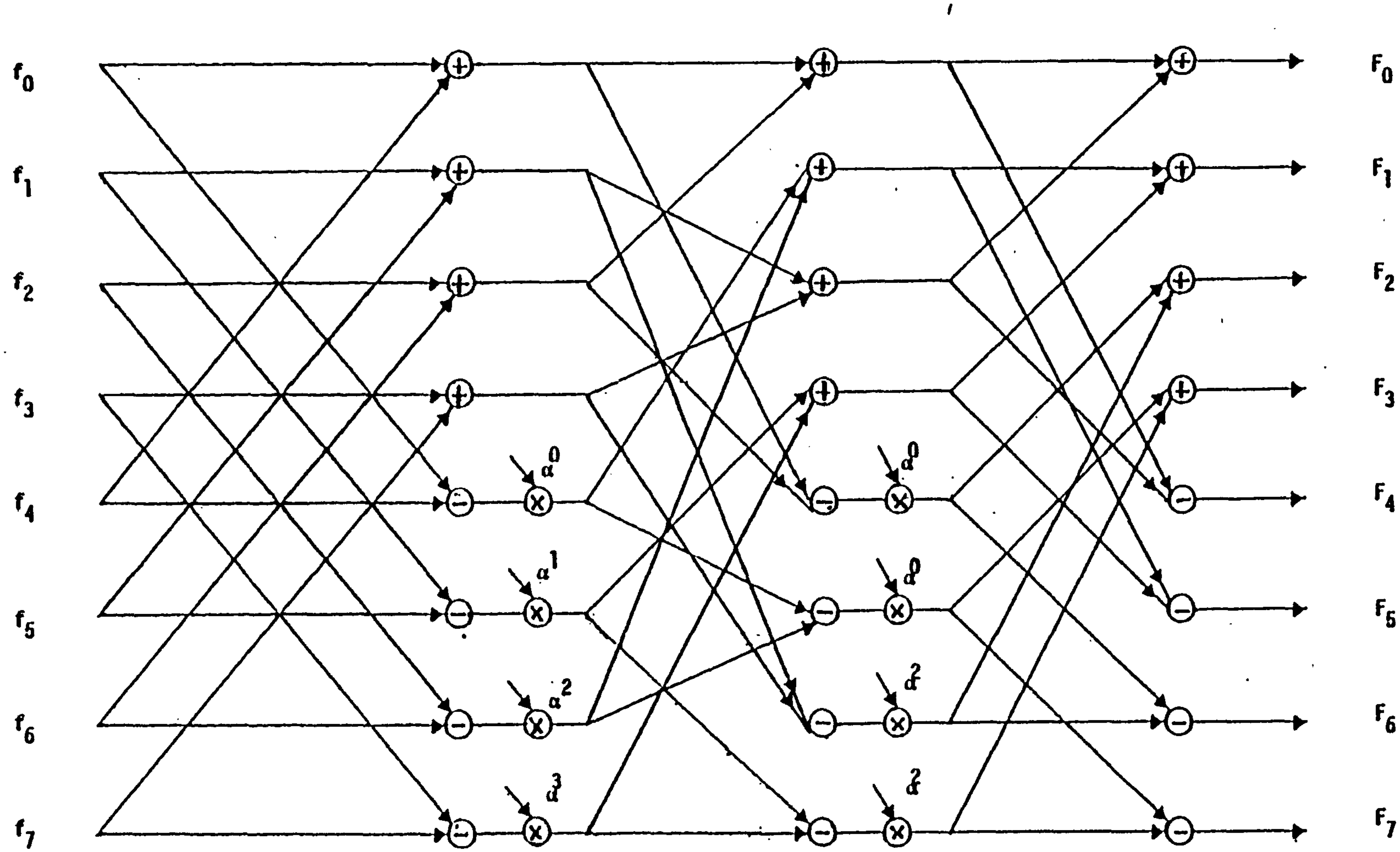


Fig 4.4 Flow graph for an eight point OIOO algorithm

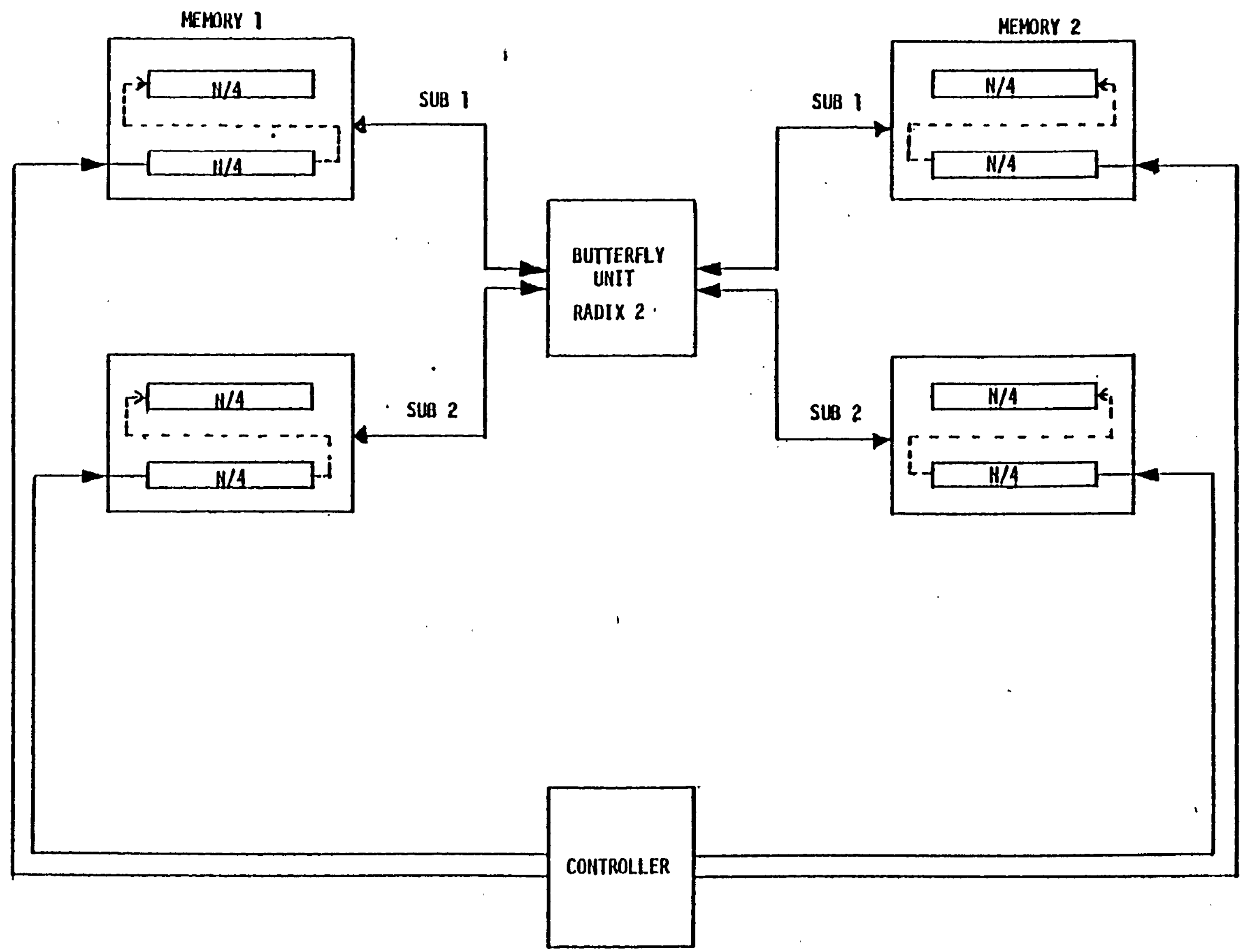


Fig 4.5 An NTT processor for OIOO algorithm

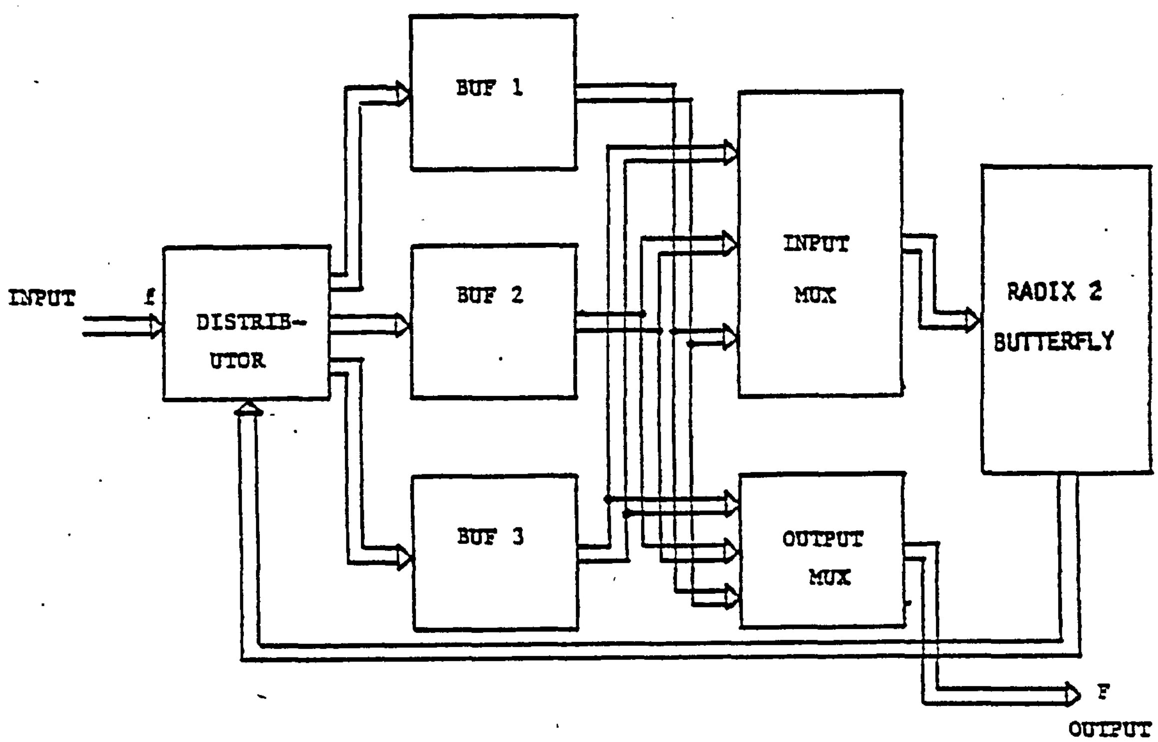


Fig. 4.6 An NTT processor for real time 0100

The addition-subtraction of the input points and the multiplication by the twiddle factors is performed in the computational unit. The computation is done in two stages, the addition-subtraction stage and the multiplication stage, the order determined by the algorithm used. For a high throughput, the structure of Fig. 4.6 requires that the input-out rate of the computational unit should be equal to the data rate of the memory structure. A pipeline structure seems a very good choice for the computational unit. It will be shown that the ROM oriented structure is extremely simple to pipeline and thus can be used with the above memory organization. The computational unit from now on in the thesis, will be referred to as a butterfly structure and will be restricted to radix 2, as mentioned earlier.

4.2.2 The Butterfly Unit

The input to this unit from the memory structure is two input complex points. The control unit supplies the information about the stage of computation and the position of the butterfly in that stage. The twiddle factors are generated in this unit and multiplied at the appropriate stage in the butterfly unit. By looking at the matrix expansion of the transform matrix (Fig. 4.3) we note that the FNTT algorithm obtained is of DIF type where the input points are first added-subtracted and then multiplied by the twiddle factors.

The selection of the field for NTT dictates the form of the cyclic generator and thus the twiddle factors. Therefore the field or the prime moduli should be chosen such that the generator is simple and also such that the resulting butterfly unit requires less hardware. The concept developed in the previous chapter will be applied for selecting the primes for efficient hardware realization of the butterfly unit.

4.2.3 Efficiency of Primes

Large transform length is achieved by the use of prime moduli. Prime moduli can be divided into two groups, 4n + 1 type and 4n + 3 type. For 4n + 3 type primes, the generator is of the form $\gamma + \beta\sqrt{-1}$ where γ & $\beta \varepsilon$ GF(m) and $x^2 + 1$ is an irreducible polynomial in the first order field. Fig. 4.7(a) shows the radix 2, DIF type butterfly and Fig. 4.7(b) shows the implementation of the butterfly using look up tables. The operation represented by $\odot$ are performed in look up tables. The input points are the elements of GF(m_i^2) and can be considered as complex points. In the first stage, addition-subtraction is performed. The subtracted part is then multiplied with the proper twiddle factors. All the binary operation performed are complex. Multiplication by twiddle factors requires 4 multiplications and one addition and subtraction. A total of three stages and 10 binary operations are required to obtain the output points.

For 4n + 1 type primes, α can have the simple form $\sqrt{r}$ where $r \varepsilon$ GF(m_1) and x^2-r is an irreducible polynomial in first order field. Fig. 4.8 shows the implementation of the butterfly unit for 4n + 1 type prime. Two different configurations are shown for the multiplications by powers of α. Even powers of α can be considered as purely real and therefore only real multiplications are required. The odd powers of α require a multiplexing stage after multiplication and also an additional multiplication by r which in look up table implementation does not require any extra stage.
Two stages and 6 binary operations are required to compute the output points. A comparison between two kind of primes is shown in

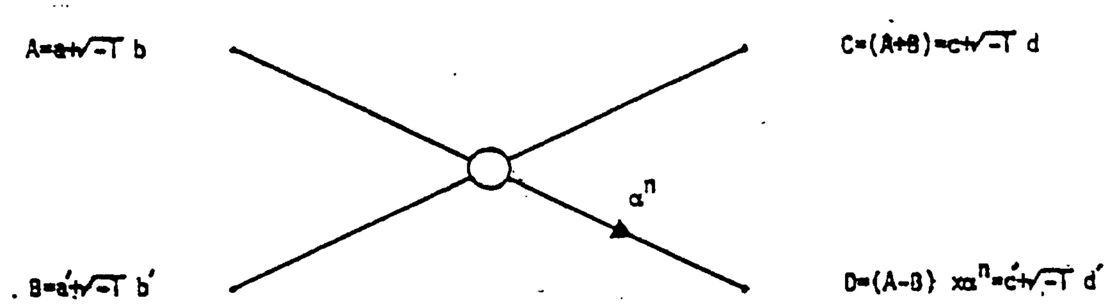


Fig. 4.7 a) Radix 2 butterfly for 4n+3 prime

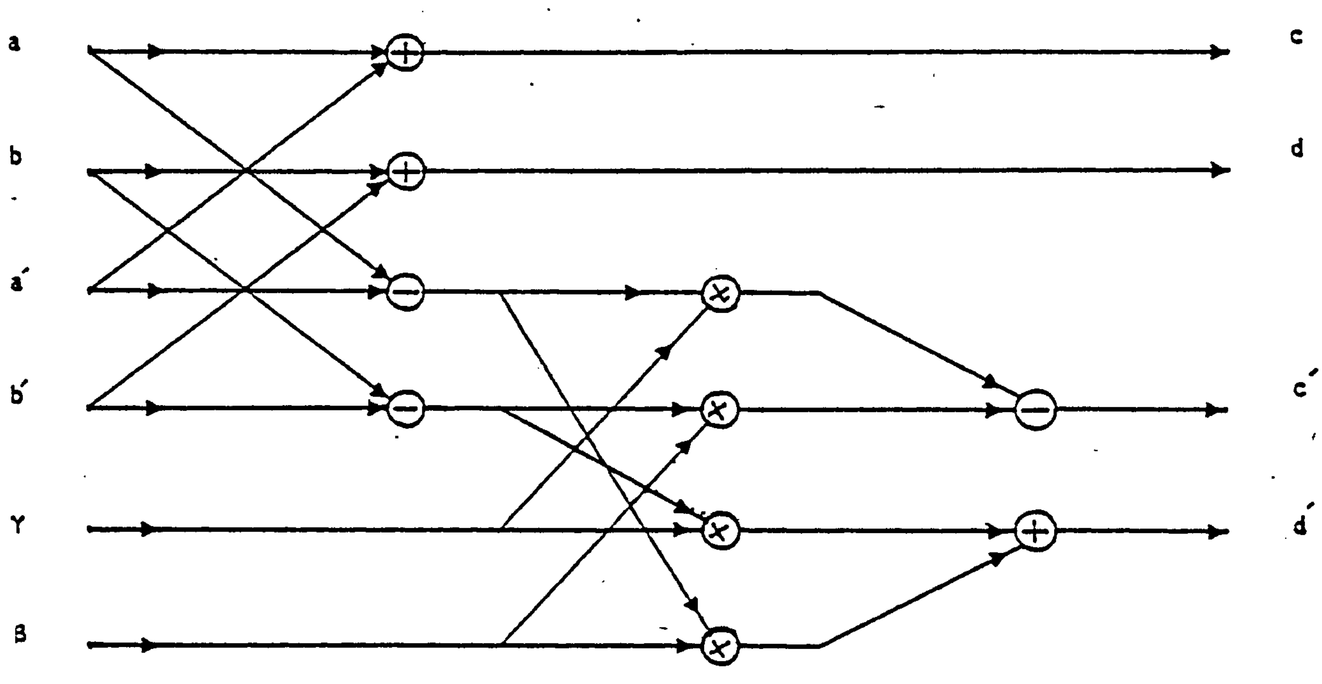


Fig. 4.7 b) Implementation of radix 2 butterfly unit for 4n+3 prime

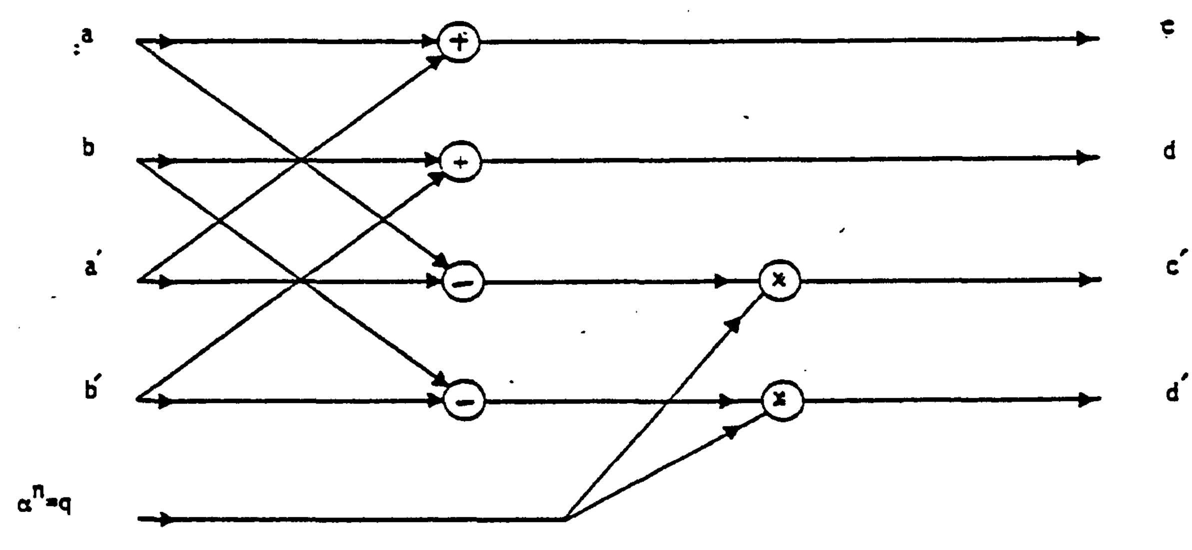


Fig. 4.8 a) Butterfly unit for 4n+1 prime (n=even)

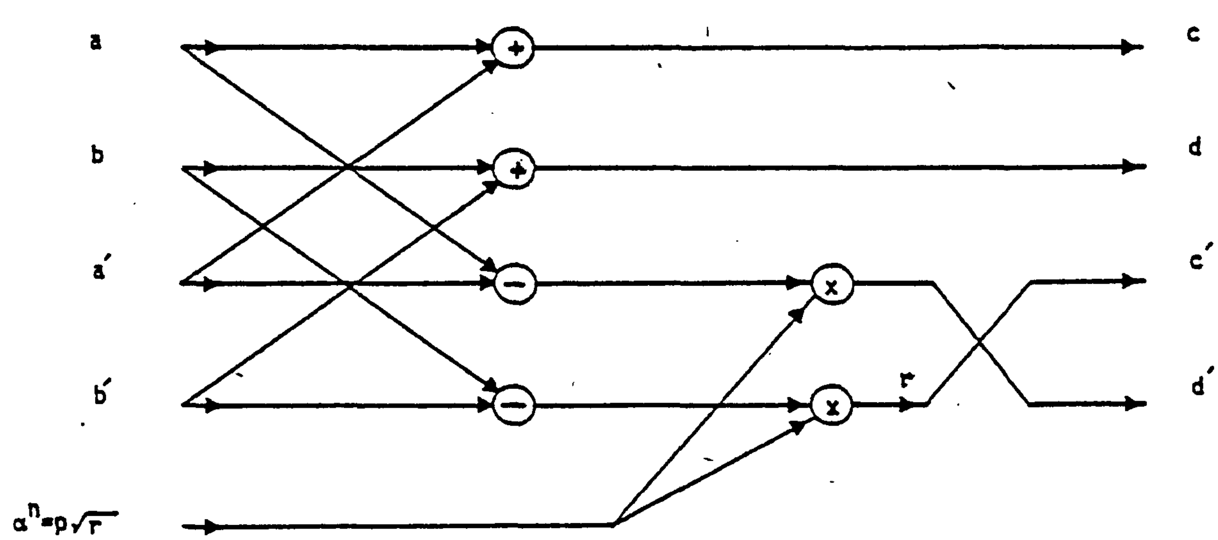


Fig 4.8 b) Butterfly unit for 4n+1 prime (n=odd)

the following table

primes	add.	subt.	mult.	stages
4n + 3	3	3	4	3
4n + 1	2	2	2	2

Table 4.1: Comparison between the primes.

From the table, it is obvious that 4n + 1 type primes are more efficient than 4n + 3 type primes. They not only require less number of stages, but also require less number of binary operations. Therefore, while choosing the primes for NTT, the preference should be given to 4n + 1 type primes. Tables 4.2 and 4.3 list the suitable primes and the transform length associated with them.

4.2.4 Selection of the Primes for Hardware Implementation

Discussed in the previous chapter, the NTT is computed over a ring which is a direct sum of several second order Galois fields for a large dynamic range. A transform length of 128 points is quite reasonable for practical application. The primes will be selected to provide this transform length and a reasonable dynamic range.

4n + 1 type primes can be represented as $m = q \cdot 2^p + 1$ where q is odd and the maximum transform length over the second order field is equal to 2^{p+1}. For a 128 point transform length, p is 6 and the first few selections are:

Transform Length $N = 2^{k+1}$	Representation of Primes for Transform Length N $m_i = q \cdot 2^k+1$		Prime m_i	Representation of m_i in Number of Bits
$32 = 2^{4+1}$	q = 1	1.2^4+1	17	4.087
	q = 7	7.2^4+1	113	6.820
	q = 15	15.2^4+1	241	7.913
	q = 21	21.2^4+1	337	8.397
	q = 25	25.2^4+1	401	8.647
	q = 27	27.2^4+1	433	8.758
$64 = 2^{5+1}$	q = 3	3.2^5+1	97	6.644
	q = 11	11.2^5+1	353	8.464
$128 = 2^{6+1}$	q = 3	3.2^6+1	193	7.592
	q = 7	7.2^6+1	449	8.811

Table 4.2 TABLES OF PRIMES $m_i = 4n + 1$ LESS THAN 9 BITS

Transform Length $N = 2^{p+1}$	Representation of Primes for Transform Length N $m_i = q \cdot 2^p - 1$		Prime m_i	Representation in Number of Bits
$32 = 2^{4+1}$	q = 3	3.2^4-1	47	5.555
	q = 5	5.2^4-1	79	6.304
	q = 15	15.2^4-1	239	7.901
	q = 17	17.2^4-1	271	8.082
	q = 23	23.2^4-1	367	8.520
	q = 27	27.2^4-1	431	8.752
	q = 29	29.2^4-1	463	8.855
$64 = 2^{5+1}$	q = 1	1.2^5-1	31	4.954
	q = 7	7.2^5-1	223	7.801
	q = 15	15.2^5-1	479	8.904
$128 = 2^{6+1}$	q = 3	3.2^6-1	191	7.577
$256 = 2^{7+1}$	q = 1	1.2^7-1	127	6.989
	q = 3	3.2^7-1	383	8.581

Table 4.3 TABLES OF PRIMES $m_i = 4n + 3$ LESS THAN 9 BITS

$$m = q \cdot 2^6 + 1$$

for $q = 1$ $m = 65$ which is not a prime

for $q = 3$ $m = 193$

for $q = 5$ $m = 321$ which is not a prime

for $q = 7$ $m = 449$

for $q = 9$ $m = 557$.

The dynamic range associated with the first three moduli is:

$$\prod_{i=1}^{3} m_i = 193 \times 449 \times 577 \simeq 2^{25.6}$$ which is quite

reasonable for most of the applications. We are interested in implementing addition-subtraction using sub-moduli and 1K x 8 commercially available ROMS. The 1K x 8 ROMS have 10 address lines and the two numbers which are to be added-subtracted should not have a combined address of more than 10 bits. The sub-moduli are chosen such that their product is equal to or greater than two times the main modulus and therefore the main modulus should not have more than 9 bits representation. Moduli 193 and 449 have nine bits representation and a combined dynamic range of approximately 16 bits. If a dynamic range of more than 16 bits is required, then we are forced to use moduli of $4n + 3$ type which are less efficient than $4n + 1$ type.

The moduli of $4n + 3$ type can be represented as $m = r \cdot 2^k - 1$ where r is odd and the maximum transform length in 2nd order field is equal to 2^{k+1}.

For N equal 128, k is 6 and the first few selections are as follows:

$$m = r \cdot 2^6 - 1$$

for $r = 1$	$m = 63$ which is not a prime
for $r = 3$	$m = 191$
for $r = 5$	$m = 319$ which is not a prime
for $r = 7$	$m = 447$ which is not a prime

For $r > 7$, moduli have more than 9 bits representation and are not useful for our purposes. Table 4.3 shows that modulus 127 can also be used for a transform length of 128 points. For the same transform length, 191 provides larger dynamic range than 127. The final selection of moduli, from hardware constraints, is then $m_1 = 191$, $m_2 = 193$ and $m_3 = 449$, and the dynamic range is 23.98 bits. This is equivalent to saying that the number theoretic transform is computed over a finite ring which is isomorphic to the direct sum of three Galois fields of second degree that is:

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

The generator for these primes are as follows:

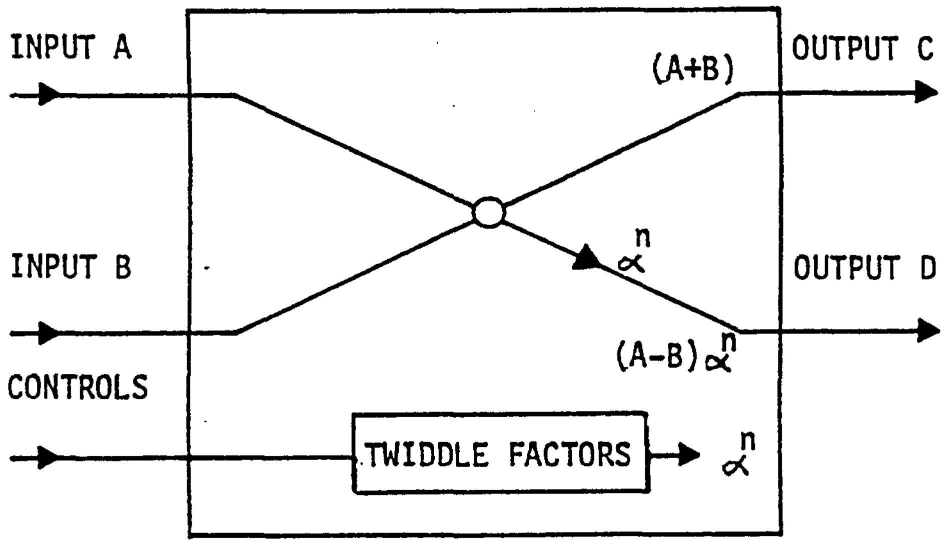
modulus $m_1 = 191$	$\alpha_1 = 66 + 6\sqrt{-1}$
modulus $m_2 = 193$	$\alpha_2 = \sqrt{125}$
modulus $m_3 = 449$	$\alpha_3 = \sqrt{391}$

4.3 ROM REALIZATION OF BUTTERFLY STRUCTURE

A conceptual block diagram of the butterfly unit is shown in Fig. 4.9. The two input points are supplied to the butterfly unit along with the stage of computation and the position of the butterfly. Another control is required to distinguish between the direct or inverse transform for generating the proper twiddle factors. For each input set of data, an output set is obtained with an initial lag of 5 or 7 stages depending upon the primes used. The computation inside the butterfly unit is performed using sub-moduli for efficient hardware realization.

4.3.1 ROM Realization for 4n + 1 Primes

Fig. 4.10 shows the implementation of the butterfly unit for a 4n + 1 type prime. Each rectangular block represents a ROM and a latch. For the DFT algorithm, the input points are first added and subtracted. The first stage therefore consists of residue tables, named as TRSM, sub-modulo 30 and 31. Eight tables are required to reduce the input data points modulo the sub-moduli. In the 2nd stage, sub-modulo addition is performed and at the 3rd stage the added part is reconstructed whereas the subtracted part is first reconstructed and then is converted into index form, again in sub-moduli. Reconstruction, index look up and sub-modulo reduction is performed in one table for each input and each sub-modulus. The twiddle factors in index form are also accessed at this stage. The fourth stage consists of addition of indices using sub-moduli. An extra multiplication table for pre-



CONTROLS:

DIRECT/INVERSE:	DIRECT OR INVERSE TRANSFORM
STAGE:	STAGE OF COMPUTATION
POST:	POSITION OF THE BUTTERFLY IN THE STAGE

Fig. 4.9 CONCEPTUAL DIAGRAM OF THE BUTTERFLY UNIT

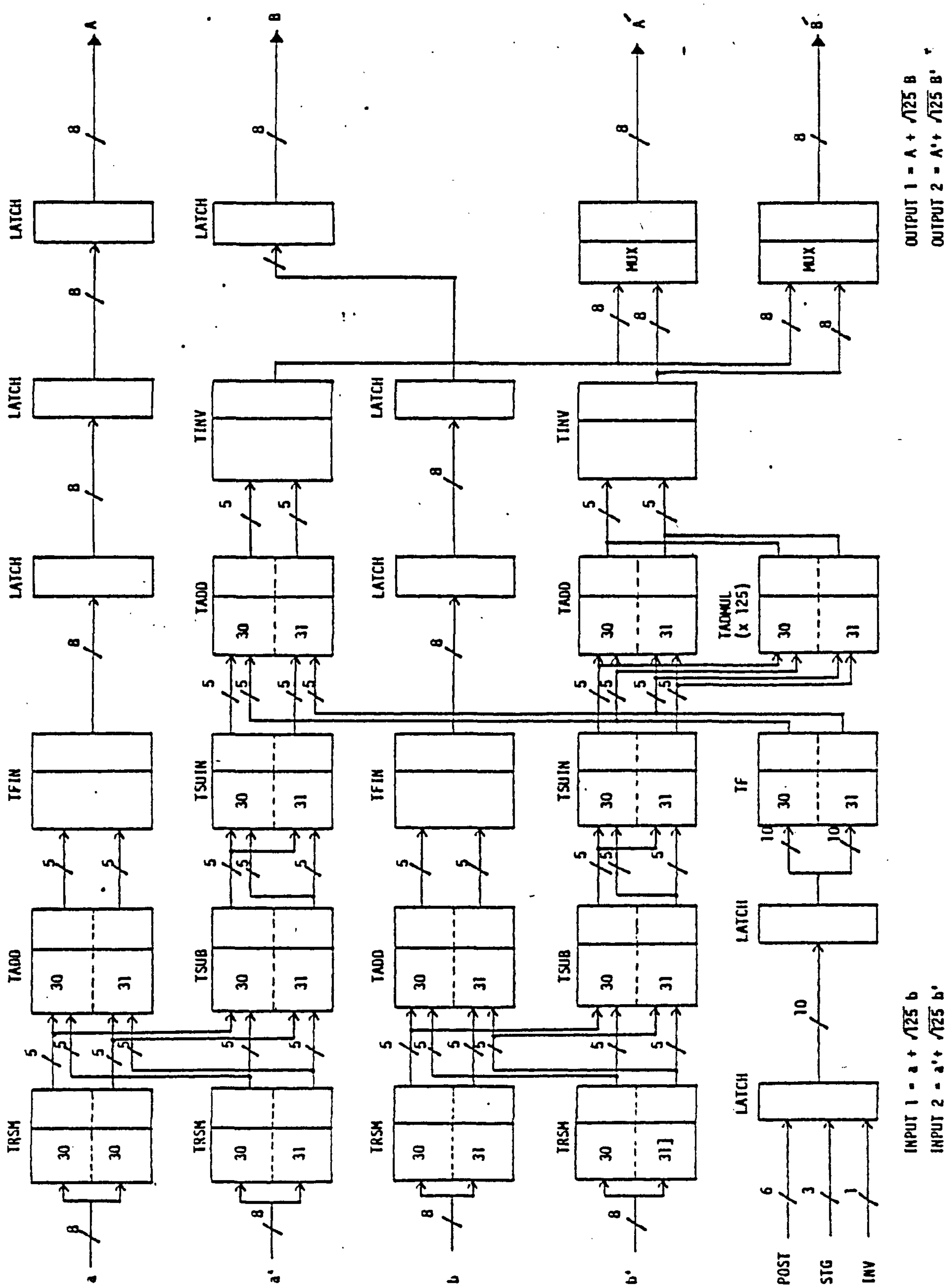


Fig. 4.10 DESIGN OF NTT BUTTERFLY FOR 4n + 1 PRIME (193)

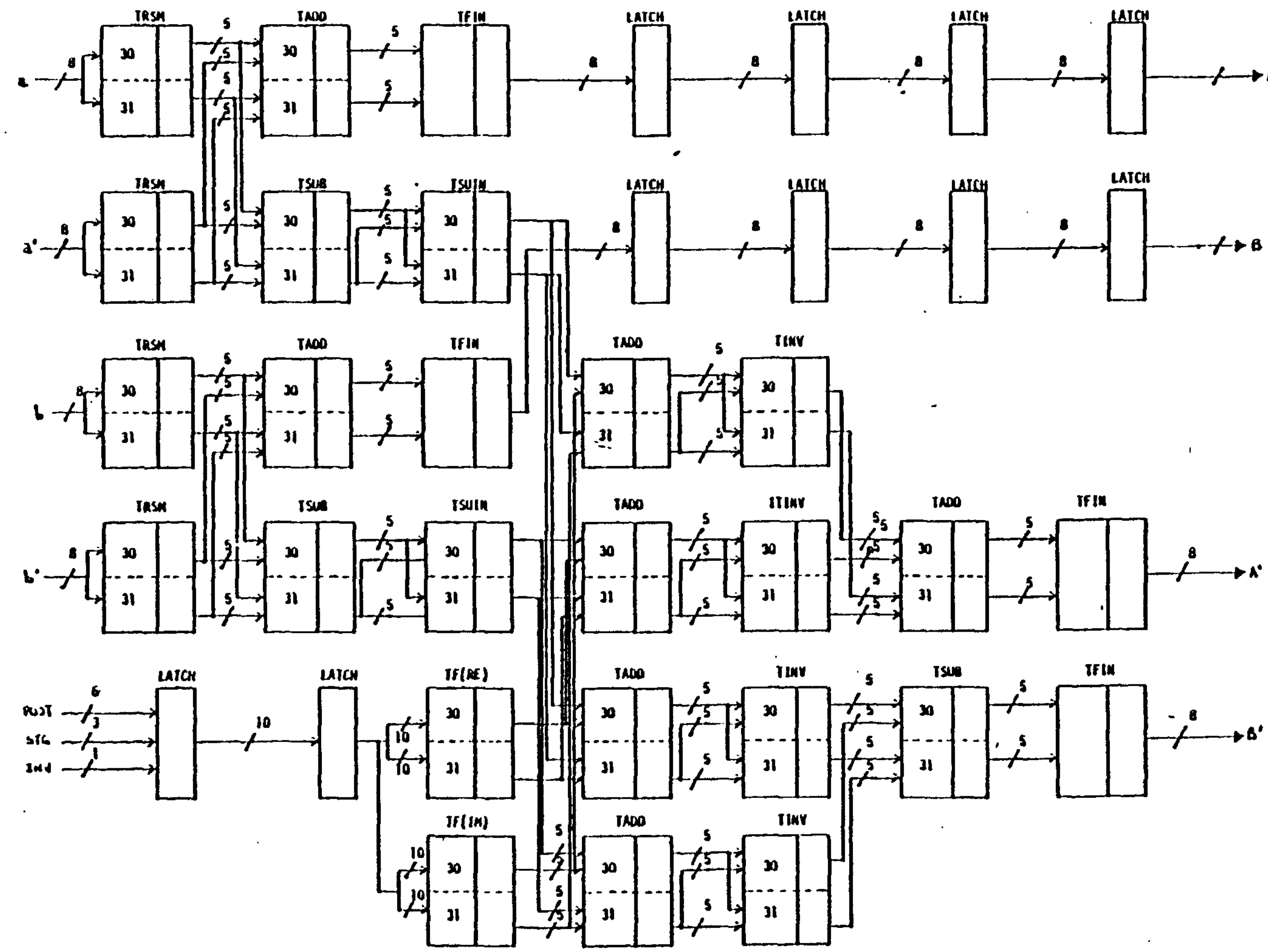
multiplication by r is also required at this stage and depending upon the power of alpha, the proper table is enabled. The fifth stage consists of accessing of the result of the multiplication from inverse look up tables and the multiplexing of the result according to even-odd powers of alpha. Looking at this structure, we find that after an initial delay of five stages, an output will be obtained and there is always a lag of five stages between input and output data.

4.3.2 ROM Realization for 4n + 3 Primes

Fig. 4.11 shows the implementation of the butterfly structure for 4n + 3 type primes. The first three stages of this structure are the same as that of 4n + 1 type. Multiplication by twiddle factors is complex for 4n + 3 type and therefore a complex multiplier is required. At the fourth stage, the addition of the indices is performed and then the fifth stage computes the real multiplications. An extra addition-subtraction is required to complete the complex multiplication which is done in the 6th and 7th stages. A total of seven stages are required to compute the two point butterfly and a lag of seven stages is presented between input and output. Table 4.4 shows the requirement for both type of primes.

primes	ROMS	Stages	MUX
4n + 3	48	7	-
4n + 1	32	5	2

Table 4.4 Requirements for both type of primes.



INPUT 1 = a + $\sqrt{-1}$ b
INPUT 2 = a' + $\sqrt{-1}$ b'

OUTPUT 1 = A + $\sqrt{-1}$ B
OUTPUT 2 = A' + $\sqrt{-1}$ B'

Fig. 4.11 DESIGN OF NTT BUTTERFLY FOR 4n + 3 PRIME (191)

From table 4.4, it is obvious that if both type of primes are used, then for 4n + 1 type primes, a delay of two stages should be introduced in the pipeline.

4.4 COMPUTER SIMULATION OF THE BUTTERFLY STRUCTURES

The butterfly structures for the three moduli were simulated on an IBM 370 using look up tables. The exact structures shown in Fig. 4.10 and Fig. 4.11 were simulated and the pipeline structure was preserved during simulation. The basic requirements for memory organization were used in the simulation part and the program for simulating memory structure was simplified. The shuffle operators were not used in the memory simulation part and the output obtained was in bit reversed form. A standard shuffle routine was used to change the bit reversed output into ordered output. This does not affect the butterfly structure in any way. The simulation programs were divided into three parts.

(i) MAIN PROGRAM: From Fig. 4.10 and 4.11, we note that output from each table is latched on each clock pulse. The latching is necessary to allow the (i+1)th stage to capture data before the address lines of the ith stage change. A pointer was initialized in the main program to clear all the registers before the application of the first data set. The subroutine table is then called to generate all the tables required for the butterfly unit. A double DO loop is used to keep track of each stage of the computation and the position of the butterfly. The input data points which are always $\frac{N}{2}$ points apart are fed to the NTT subroutine and the output is stored in the consecutive memory locations. After the completion of the transform, the data is then shuffled to obtain the ordered output.

The subroutine NTT is the simulation of the butterfly structure. The controls to this subroutine are passed in the calling argument. The NTT call is

CALL NTT (INV, INP1, INP2, STG, POST, OUT1, OUT2).

The multiplication by N^{-1} for inverse transform is also performed in the main program although for hardware implementation, multiplication by N^{-1} can be performed before starting the processing. The main program is the essential part for testing the working of the butterfly structure.

(ii) SUBROUTINE TABLE: This program generates all the required tables for each moduli. Modulo reduction was done using the instruction mode

IR = MOD (IR, MMOD)

where MMOD is the modulus and IR is the number to be reduced. The NTT is an integer number system and the implicit integer statement was used to declare all the variables as integers. The index and inverse index tables are quite easy to generate. The following six statements generates the complete index as well as inverse index table. PRIM is the primitive root and PER is the order of the primitive root. Starting value of VAL is one as zero does not have any index. IND is the index of the number and IIND is the inverse index

```
      DO 21     K = 1,PER
      VAL  =  VAL * PRIM
      VAL  =  MOD (VAL, MMOD)
      IND (VAL+1) = K
      IIND (K+1) = VAL
21    CONTINUE
```

The following steps were required to generate the powers of α

(a) initialize the value of α

(b) multiply the value with α. The multiplication performed is an extension field multiplication

(c) reduce the value to proper modulus

(d) store the value of α as the next value

(e) repeat step (b) till $| \alpha^N |_m = 1$

Noting that $| \alpha^{128} |_m = 1$, the powers of alpha for the inverse transform are obtained by adding 128 to negative powers, e.g., $\alpha^{-3} = \alpha^{128-3} = \alpha^{125}$. Other parts of the subroutine table are self explanatory. The complete listing of the program is given in the Appendix.

<u>(iii)SUBROUTINE NTT:</u> This program simulates the butterfly structure. This part assumes that the butterfly structure is arranged in pipeline configuration. Each call to this subroutine shifts the data to one stage. The subroutine call is

CALL NTT (INV, INP1, INP2, STG, POST, OUT1, OUT2),

where INV is for direct or inverse transform. INP1 and INP2 are the two complex input points. STG is the stage of the computation and POST is the position of the butterfly in that stage. OUT1 and OUT2 are the output points of the butterfly. All the registers are numbered and , before applying any input to the NTT, these registers are initialized by the control pointer named point, which clears all the registers when the subroutine is called for the first time. The twiddle factors for a particular butterfly are generated in this routine. The powers of α from 0 to 64 are stored in a table TF. The address for the twiddle factor is generated as follows:

(1) butterflies are numbered from 0 to 63 starting from the top in the flow graph, e.g. Fig. 3.6

(2) stages are numbered from 0 to 6

(3) the proper address is then generated by masking the number of bits equal to the stage number starting from the least significant bit, e.g., for stage 2 and butterfly 8, the power of α is given by

$$\text{power of } \alpha = \text{POST}/(2^{**}\text{STG}) * (2^{**}\text{STG})$$

$$= \frac{8}{2^2} \times 2 = 4 \text{ and the twiddle factor is } \alpha^4.$$

Multiplexing is also required for the moduli of a 4n + 1 type prime. The power of α is checked for even or odd and then the appropriate action is taken. The statements check are the status of multiplexer control.

The other parts of the program are self explanatory. The complete program can be found in Appendix A.

4.4.1 The Transform of Real and Complex Data for Both Primes

Before discussing the results of the simulation, the procedure for convolving real and complex data using the NTT is described. As mentioned in the previous chapter, 2nd order Galois field is isomorphic to the complex residue ring for 4n + 3 type prime. Therefore the complex data can be convolved using 4n + 3 primes. In the case of real data, two successive blocks of the data can be transformed simultaneously by feeding one block as the real part of the data and the other block as the imaginary part. This effectively increases the transform length in the case of real data.

For primes of 4n + 1 type, $\sqrt{-1}$ can be considered as a member of the field and therefore the maximum order of any element in the multiplicative group of the complex ring is $m_i - 1$, i.e., the length of the transform is the same as in the real residue field modulo m_i. One possible implementation of the transform of the complex data is to separately transform the real and imaginary parts in two Galois fields GF(m) for 4n + 1 type prime.

4.4.2 Upper Bound on the Convolution

To compute the convolution unambigously, the components of the circular convolution sum in a single Galois field, are required to have an upper bound m_i; i.e., signed numbers should remain in the interval

$-\frac{m_i-1}{2} \leq y \leq \frac{m_i-1}{2}$. The absolute upper bound on the input sequences is

$$\max|x| \cdot \max|h| \leq \frac{m_i-1}{2N} \tag{4.13}$$

where x(n) and h(n) are the input sequences. This bound on the dynamic range is pessimistic for many practical applications and if the sequence h(n) is known, it is enough to have

$$\max|x| \leq \frac{m_i-1}{2\sum_{t=0}^{N-1}|h(t)|} \tag{4.14}$$

If the input sequence consists of a set of positive numbers, the above can be restated as

$$\max|x| \leq \frac{m_i-1}{\sum_{t=0}^{N-1}|h(t)|} \tag{4.15}$$

The components of the complex circular convolution of sequences $x(t) = x_2(t) + j\,x_i(t)$ and $h(t) = h_r(t) + j\,h_i(t)$ are required to have an upper bound m_i. Hence the absolute upper bound on x and h is:

$$\max|x_r| \cdot \max|h_r| - \max|x_i| \cdot \max|h_i| \leq \frac{m_i-1}{2N} - \ldots \tag{4.16}$$

and $$\max|x_r| \cdot \max|h_i| + \max|x_i| \cdot \max|h_r| \leq \frac{m_i-1}{2N} - \tag{4.17}$$

when the convolution is performed over residue class rings (more than one modulus), all m_i are to be replaced by M in the above equations.

4.4.3 Simulation Results

Three main programs were written to test the pipelined butterfly structure for both kind of primes. The first program tests the invertibility of NTT. The 2nd program was written to test the convolution property of NTT using one block of real data. The 3rd program was to convolve two different sets of real data with a sequence with constant value in the defined interval. The details are as follows:

(1) Two separate sequences were taken as input. The real part consisted of a RAMP function, rising from 0 to 127. The imaginary part was also a ramp from 127 to 0. The 1st part of the program consists of initializing the tablesby calling subroutine TABLE. The input data is then initialized and a double DO loop then computes the transform. Input data is divided into two blocks of $\frac{N}{2}$ points. The input to butterfly consists of one point from each part. Thus, the input points are always $\frac{N}{2}$ points apart. After the transform is computed, it is permuted to produce an ordered output. INV control is then set to one and the transformed sequence is used as input for the inverse transform. After the inverse transform, each point is multiplied by $|\frac{1}{N}|_{m_i}$ to produce the original sequence. When implementing in hardware, multiplication by N^{-1} is implemented in look up tables and does not require any extra stage or delay. The above procedure was repeated for three choosen moduli and invertibility was proved. Fig. 4.12(a) shows the real and imaginary parts of the input sequence. Fig. 4.12(b) shows the transformed sequence in $GF(193^2)$ and Fig. 4.12(c) shows the

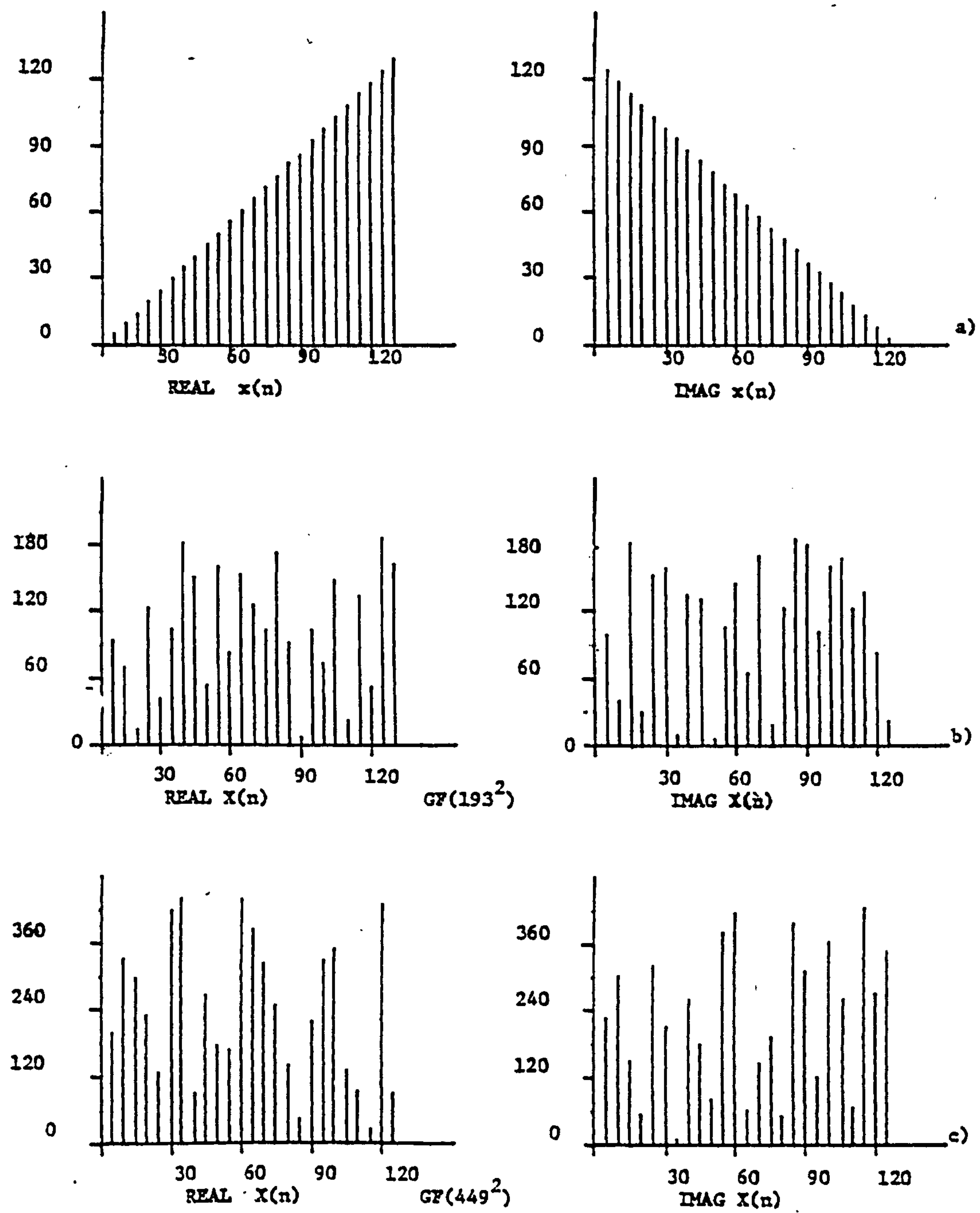


Fig. 4.12 INPUT AND TRANSFORM OF x(n)

transformed sequence in GF(449^2). Different transformed sequences are obtained in different fields for the same input sequence. After taking the inverse transforms in both the fields, same input sequence was obtained.

(2) This part was written to perform convolution of two sequences. Only one block of data was taken and was fed as the real part. The imaginary part was set to zero. To avoid ambiguity, the input sequences were chosen such that the result of the convolution is contained within the dynamic range. The 1st sequence was a rectangular pulse of height 1. The 2nd sequence was another rectangular pulse of height 2. These sequences were transformed, multiplied and then an inverse transform was performed to obtain convolution of the sequence. Zeros were appended to both the input sequences to compute linear convolution using the ccp of the NTT. Fig. 4.13(a) shows the real part of the two input sequence. The imaginary part of the sequences were taken as zero. Fig. 4.13(b) shows the transform of x(n) and Fig. 4.13(c) shows the transform of h(n) in GF(193^2). Note that imaginary parts are present in the transform domain although the original sequences had no imaginary parts. Fig. 4.13(d) shows the result of the convolution in GF(193^2).

(3) This program was the same as in part two except that the one of the input sequence was taken as a complex sequence. This sequence was convolved with another sequence whose imaginary part was set to zero. Fig. 4.14(a) shows the input sequence x(n) and Fig. 4.14 (c) shows the sequence h(n). Fig. 4.14(b) shows the transform of x(n) and Fig. 4.14(d) shows the transform of h(n) in GF(449^2). Fig. 4.14(e)

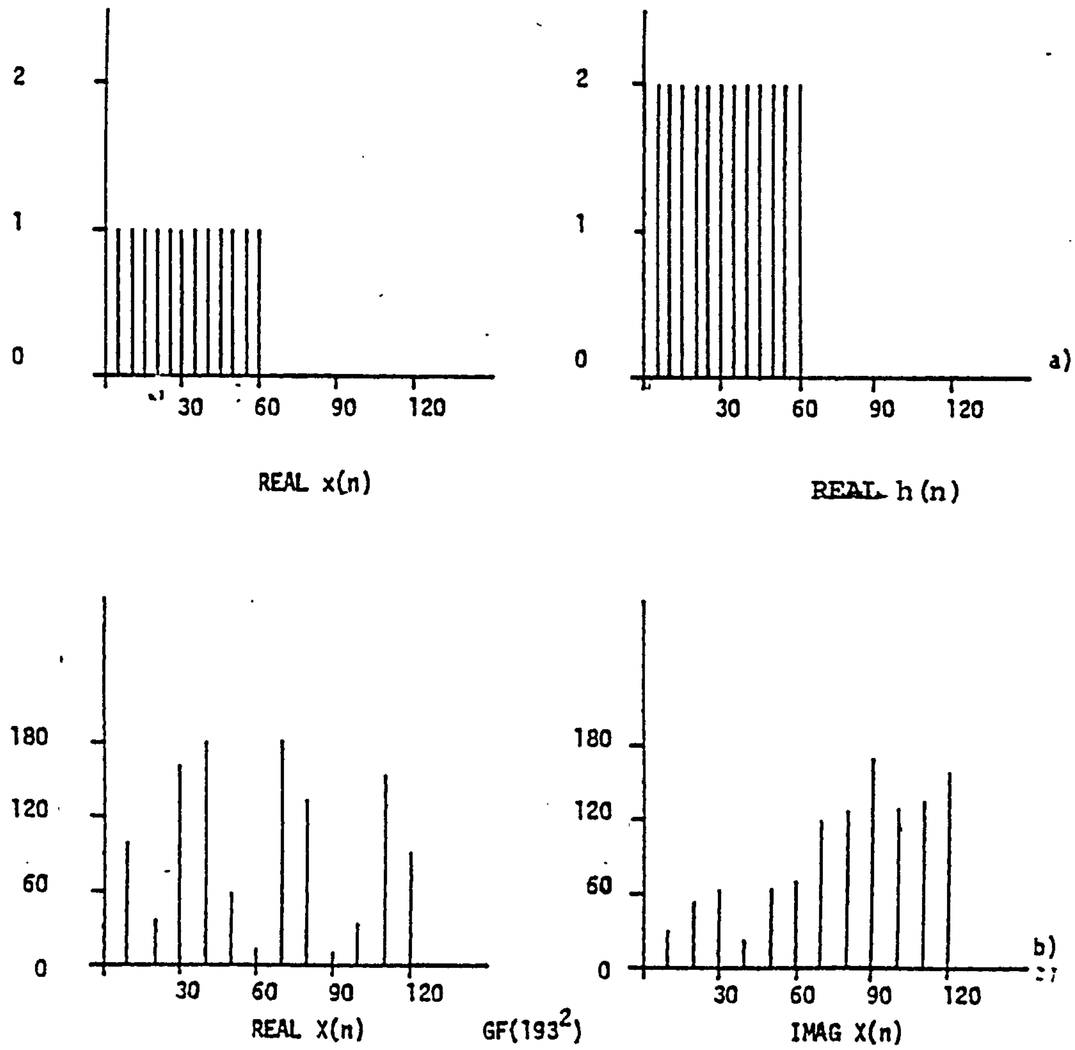


Fig. 4.13 CONVOLUTION OF REAL INPUT

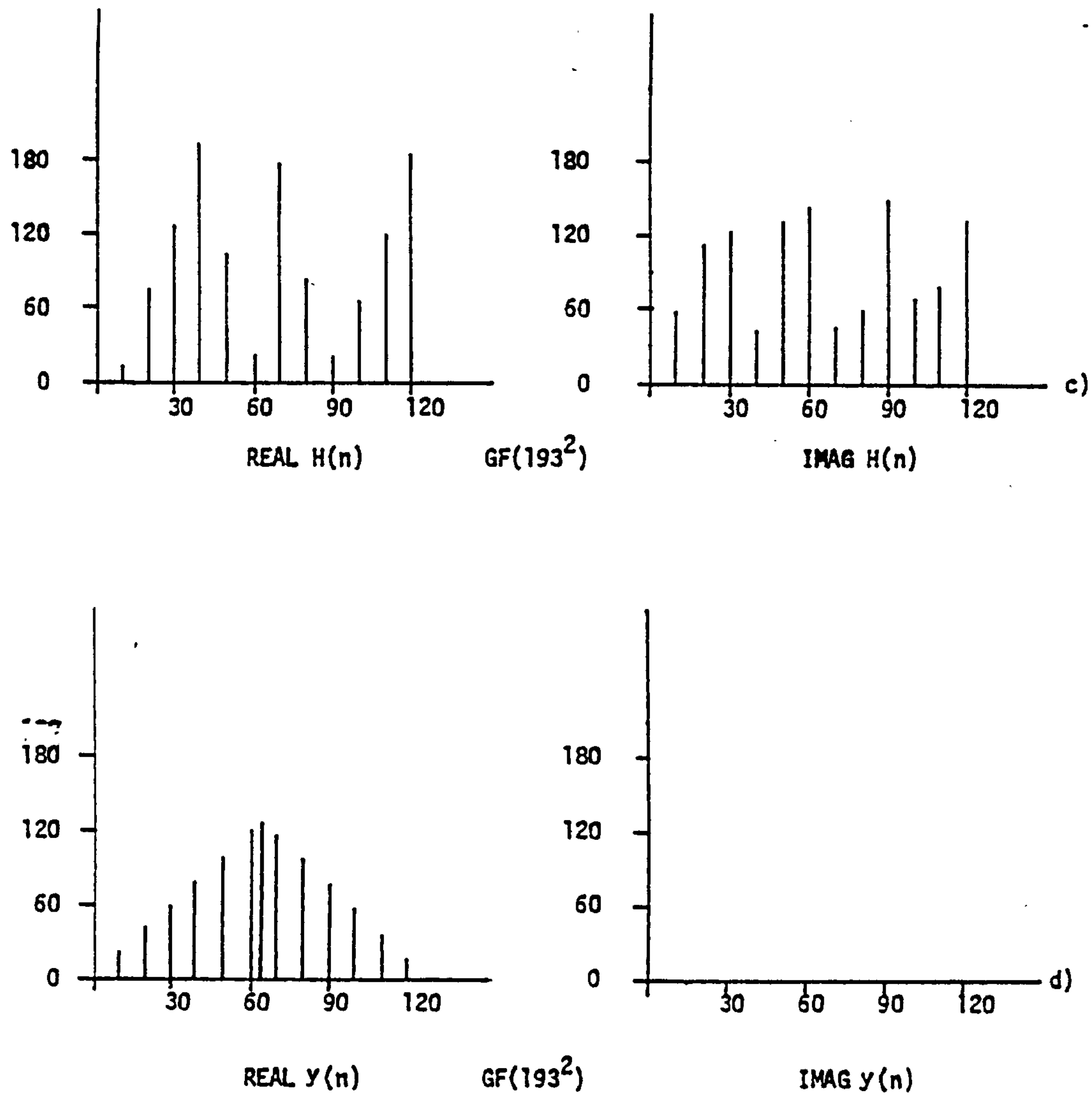


Fig. 4.13 CONVOLUTION OF REAL INPUT

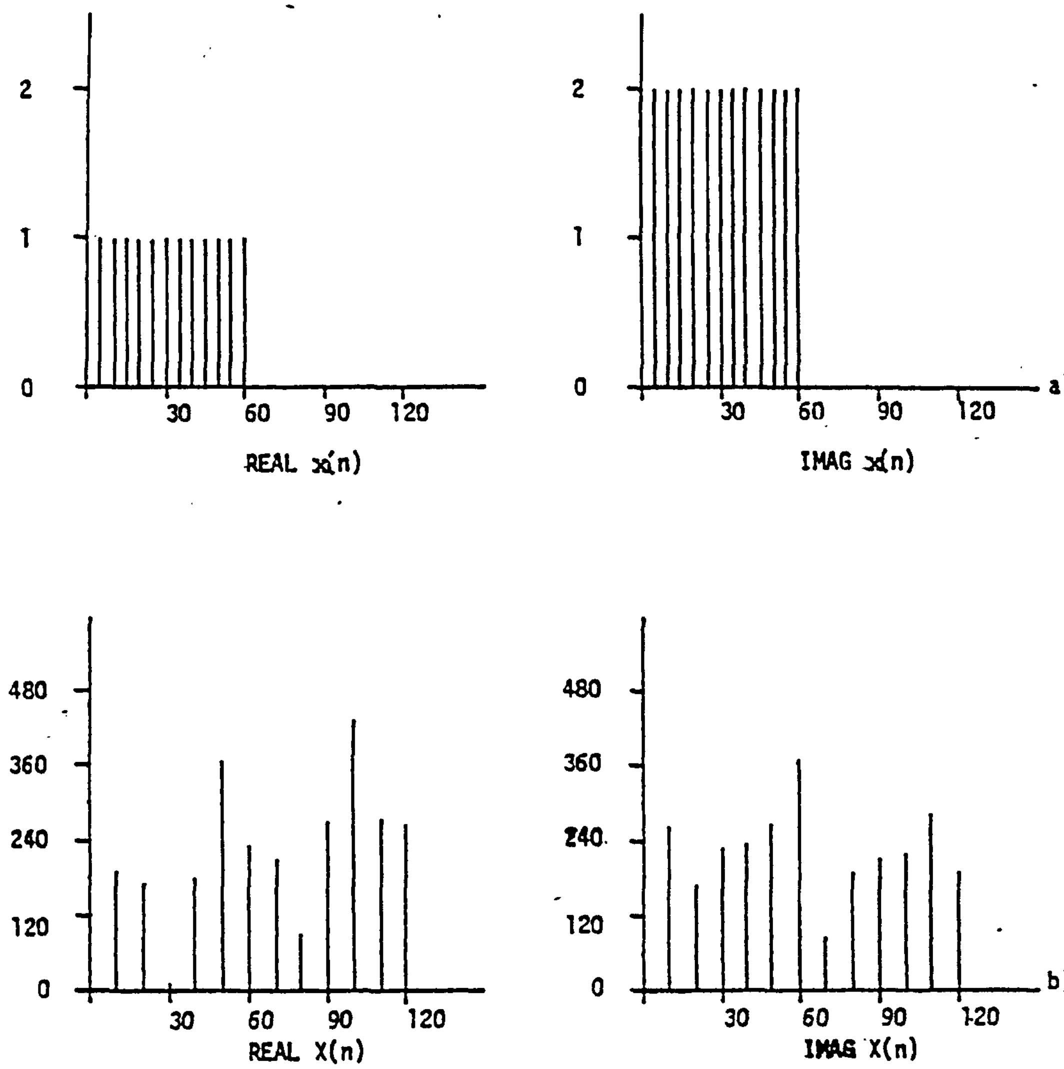


Fig. 4.14 CONVOLUTION OF COMPLEX INPUT IN GF(449^2)

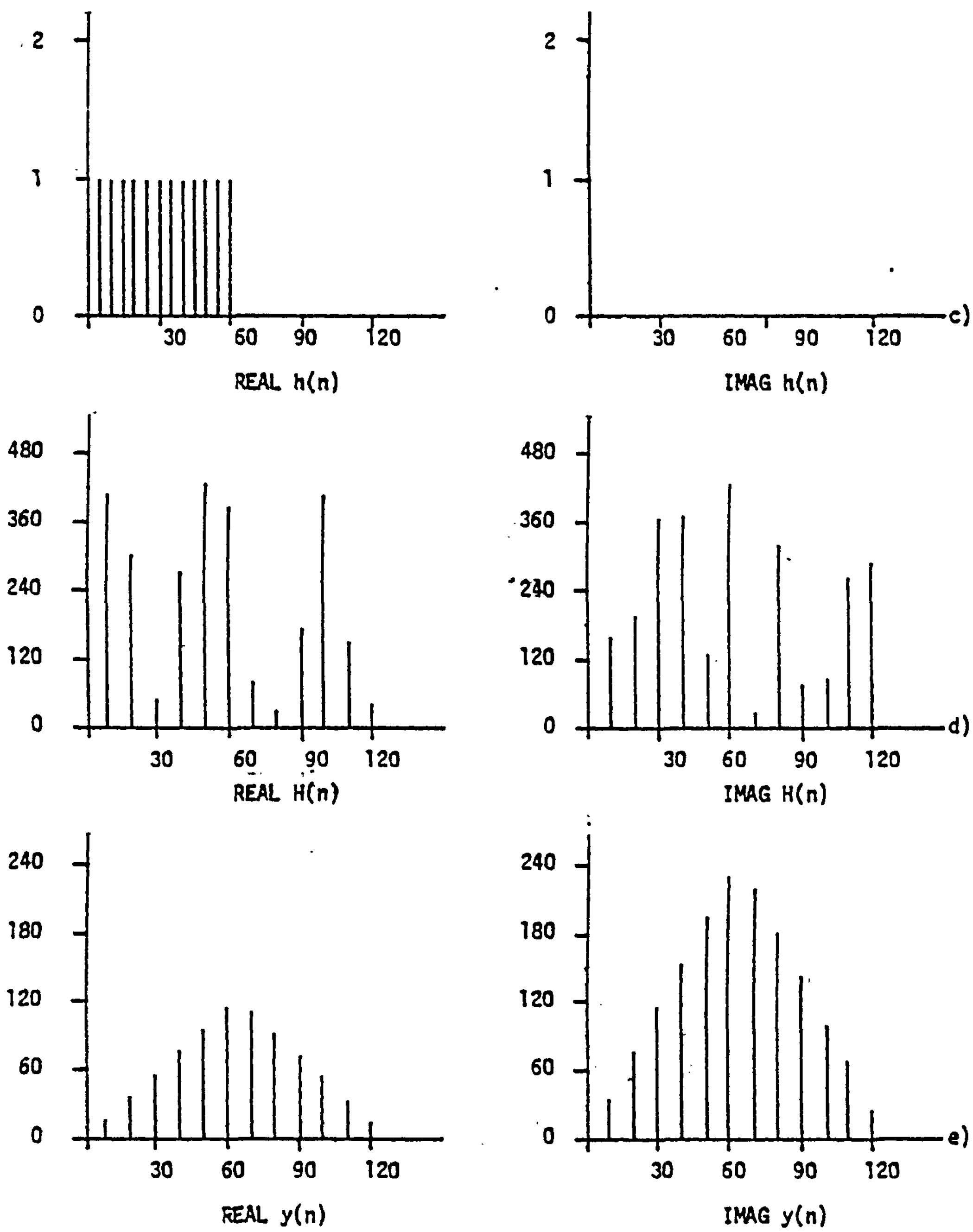


Fig. 4.14 CONVOLUTION OF COMPLEX INPUT IN GF(449^2)

shows the results of the convolution in GF(449^2). In this way two blocks of the real data can be simultaneously convolved with the other sequence and the effective transform length for the input sequence is doubled. The simulation programs can be found in Appendix A.

4.5 HARDWARE IMPLEMENTATION OF THE BUTTERFLY STRUCTURE

A complete butterfly structure for modulus 4n + 1 was implemented in hardware. The modulus 193 was choosen because it yields hardware of the simpler form. The hardware implementation is that of a proto type and the Eproms used are not the fastest available in the market. The access time of the Eproms used is 450 nsec and the registers used have a settling time of 30 nsec. The butterfly structure is a pipeline structure and the throughput rate depends on the access time of the ROMS and latch settling time. The data on the output of the ROM is latched before the new address is supplied. The clock pulses are therefore delayed for every stage starting from the output stage. Fig. 4.15 shows the clock pulses required for latching the data, from the Eproms, at each stage.

The width of the clock pulses is equal to the latch settling time say t_s nsec. Before the clock pulse can be applied to any stage, the address lines on Eproms should be stable for at least t_{acc} ns (address to output delay) and therefore the maximum rate at which the pipeline can run is equal to $2 \cdot t_s + t_{acc}$.

4.5.1 Description of ICs Used

(i) Eproms 2708 were used to store the look up tables for the butterfly structure for mod 193. The complete data for this Eprom can be found

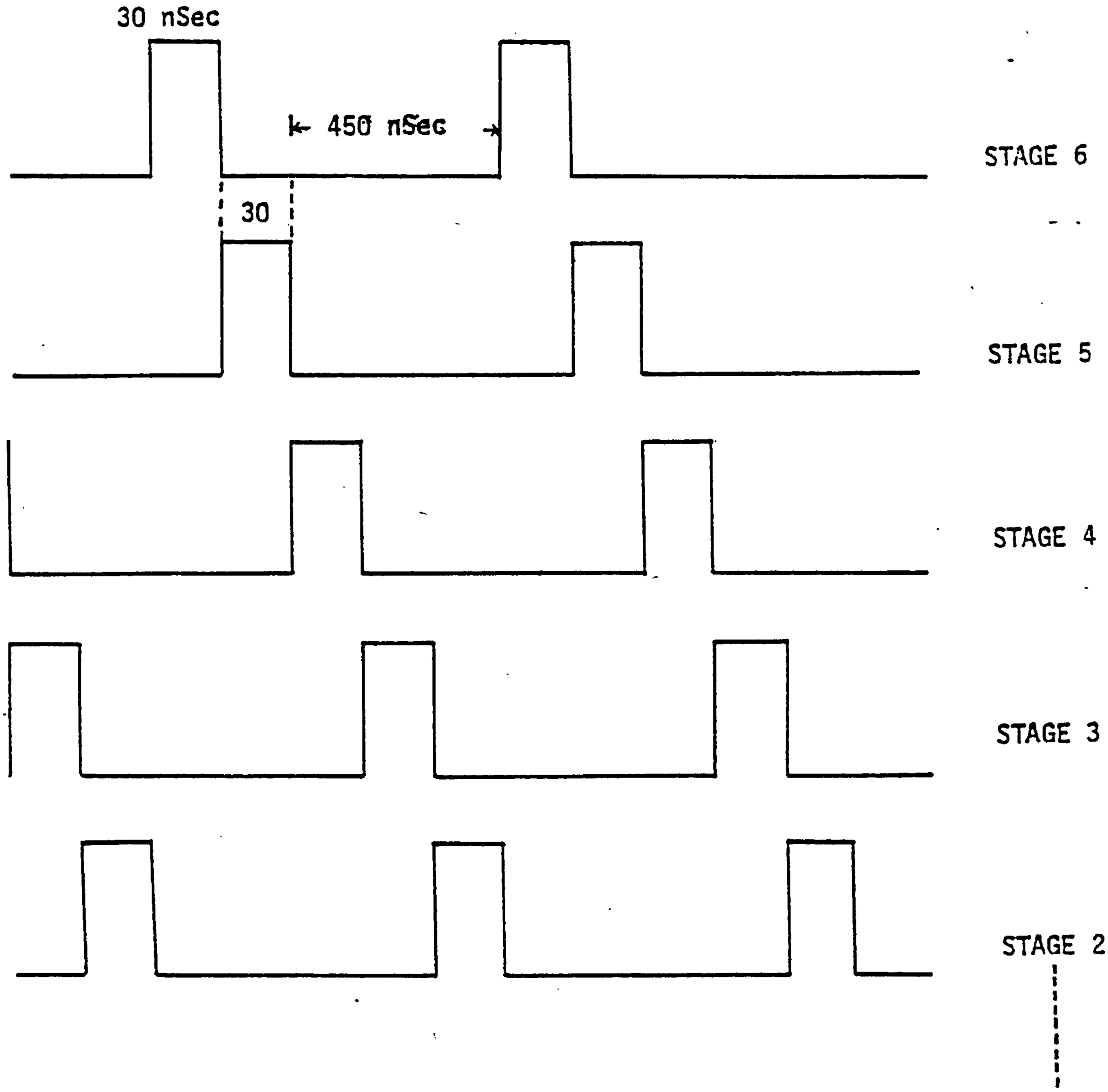


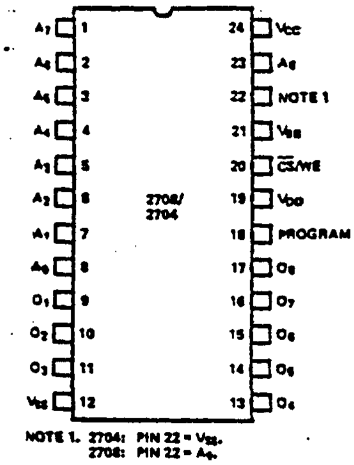
Fig. 4.15 Clock pulses for the Butterfly unit

in [12]. Fig. 4.16 shows the pin connections of the 2708.

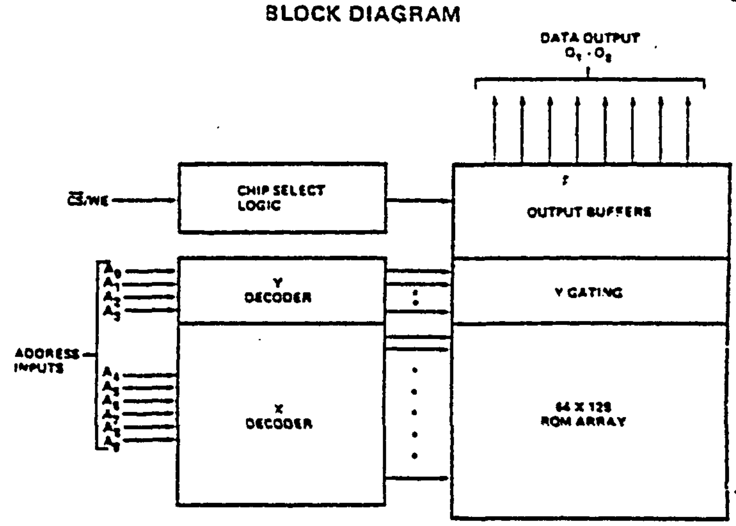
The Eprom requires three power supplies in the read mode, V_{CC}, V_{BB} and V_{DD} which are +5, -5 and 12 volt respectively. It is a 1k x 8 bits Eprom and has 10 address lines and 8 data lines. Higher address lines are grounded if they are not in use, e.g., table of residues where only 8 address lines are required for modulus 193. All the computation in the butterfly was done using the sub-modular approach, therefore, only five data lines were used. The other three data lines can be used as controls, e.g., for parity check. We have used the 6th data line as a control line for multiplexers. The Eproms can be programed on an Intel universal prom programmer. These Eproms have tristated outputs which are controlled by the voltage level on $\overline{\text{CS}}$/WE pin. Thus the output of more than one Eprom can be hooked together without any problem of a bus-conflict. The access time of the Eprom is 450 nsec.

(ii) 8 bit input output port, 8212 was used as the latch. This is a very powerful chip and can be used for multiple purposes. The pin configuration is shown in Fig. 4.17. To use it as a latch, the device selection logic ($\overline{\text{DS1}}$ · DS2) is set true and the mode pin is kept at high level. The strobe pin is used as input for clock pulses. When the strobe is high, the output follows the input and for strobe low, output does not change. The maximum latch settling time is 30 nsec. and therefore the clock pulse which is used to strobe the data has a pulse width of 30 nsec. The $\overline{\text{CLR}}$ pin is permanently kept high for the latch operation.

PIN CONFIGURATIONS



BLOCK DIAGRAM



PIN NAMES

A_0-A_9	ADDRESS INPUTS
O_1-O_8	DATA OUTPUTS
$\overline{CS}$/WE	CHIP SELECT/WRITE ENABLE INPUT

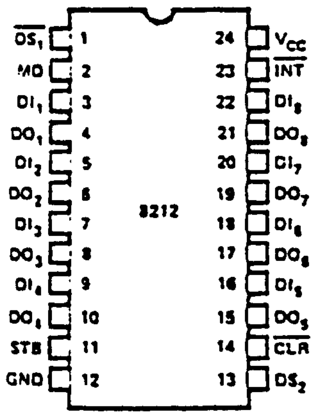
PIN CONNECTION DURING READ OR PROGRAM

MODE	PIN NUMBER 9-11, 13-17	12	18	19	20	21	24
READ	D_{OUT}	V_{SS}	V_{SS}	V_{DD}	V_{IL}	V_{BB}	V_{CC}
PROGRAM	D_{IN}	V_{SS}	Pulsed V_{IHP}	V_{DD}	V_{IHW}	V_{BB}	V_{CC}

Fig. 4.16 BLOCK DIAGRAM AND PIN CONFIGURATION OF

27o8, 1K x 8 EPROM

PIN CONFIGURATION



PIN NAMES

DI_1-DI_8	DATA IN
DO_1-DO_8	DATA OUT
$\overline{DS_1}$-DS_2	DEVICE SELECT
MD	MODE
STB	STROBE
$\overline{INT}$	INTERRUPT (ACTIVE LOW)
$\overline{CLR}$	CLEAR (ACTIVE LOW)

LOGIC DIAGRAM

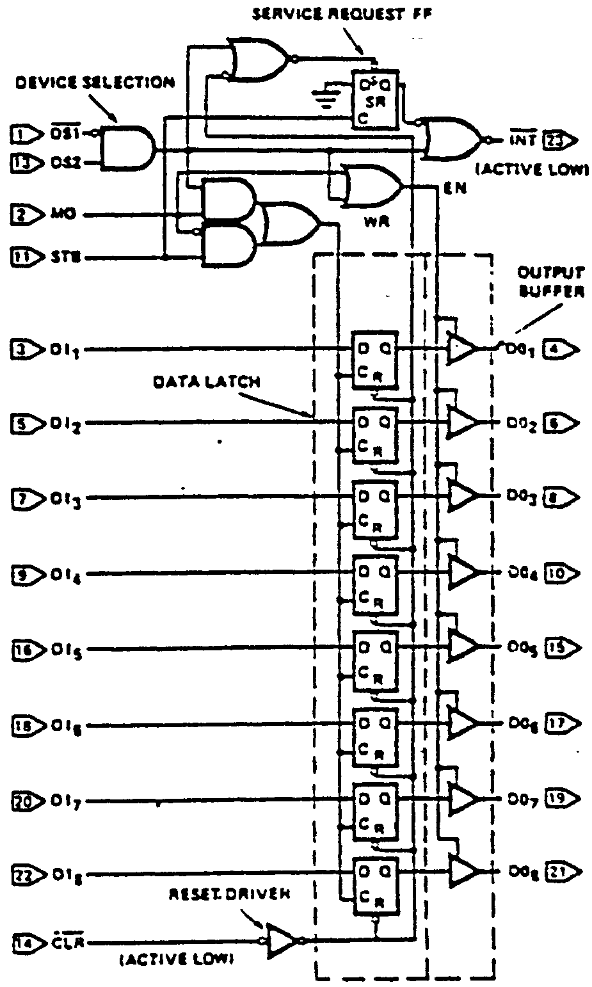


Fig. 4.17 LOGIC DIAGRAM AND PIN CONFIGURATION OF 8212, 8-BIT LATCH

The 8212 was also used as a multiplexer. When the device selection logic is zero, the output goes to a high impedence making multiplexing possible.

4.5.2 Generating and Storing The Tables

For the storing of the tables, a universal prom programmer, by Intel, was used and the tables were generated using assembly language to program an Intel 220 system. The Intel 220 system is a microprocessor based system and uses an 8085, 8 bit, microprocessor chip as the central processor unit.

All the programs written to generate tables can be found in the Appendix B. Modulo reduction is not as simple as in WATFIV and separate subroutines were written to reduce modulo 30, modulo 31 modulo 192, modulo 193 and modulo 930. Two more subroutines were written to compare the results to 738 for negative numbers and to reduce negative numbers modulo 193, namely COM738 and NEGCON. These were required to obtain the correct result after the subtraction of numbers using sub-moduli. The maximum result of addition of two numbers, modulo 193, is 384 and the maximum negative result is -192. When the final result is reconstructed using the Chinese Remainder Theorem, the negative number, say x, will be represented as $30 \cdot 31 - x$ or $930 - x$ and therefore the number range 738 to 929 is used for negative numbers. The division of the dynamic range is as follows:

$0 \leq x \leq 384$	positive numbers
$384 \leq x < 738$	prohibited combinations, they never occur as a result of an operation

$738 \leq x \leq 929$ negative numbers

After the reconstruction, if the number occurs in the negative range, it has to be represented modulo 193 and correction has to be done. Subroutine COM738 is called to find out the range in which number lies. If the number is greater than or equal to 738, then subroutine NEGCON is called to convert the negative number to mod 193. Consider the numbers 30 and 182. The result of subtraction is 30 - 182 = -152, which in sub-moduli will be represented as 778. To convert it to main moduli, subtract 930 from it and add 193 which is 778 - 930 + 193 = 41 and is the actual representation of -152 modulo 193. The following is a listing of the program which converts the negative number to modulo 193.

```
                PUBLIC    NEGCON
                CSEG
        NEGCON: PUSH      H
                LXI       H, 8400H; no. to be converted is in
                                    memory location 8400H
                MOV       A,M
                SUI       162
                ADI       193
                MOV       M,A
                POP       H
                RET
                END
```

930 can be represented as 00000011 10100010 in the binary number system. Subtracting 930 from any number, greater than or equal to 738, is equivalent to subtracting the lower byte of 930, which is 162, from the number and then adding 193. The one byte result is the correct conversion of the negative number. The reader can verify that the above program converts all negative numbers from 738 to 929 correctly.

The main programs for addition table, subtraction table, index table, inverse index table, twiddle factors table, and the reconstruction table were written separately and are given in the appendix. The generation of the twiddle factor table requires special attention. The memory organization which is used for this implementation simplifies the generation of the twiddle factors. The following procedure was used to generate the table.

(i) store the values of the powers of alpha from 0 to 63

(ii) number the butterfly from 0 to 63 in binary number system from the top where the 1st input point is supplied as input

(iii) number the stages from 0 to 6

(iv) mask the number of least significant bits equal to the number of stage, e.g., for stage 2, numbered as one, only one bit is masked.

butterfly no.	masked bit	power of α
000000	000000	0
000001	000001	0
000010	000010	2
000011	000011	2
000100	000100	4
000101	000101	4
000110	000110	6
000111	000111	6
⋮	⋮	⋮

Hence the correct twiddle factors are generated for each stage. There are 64 butterfly computations per stage and seven stages, therefore a storage of 64 x 7 words is required for the twiddle factors for a direct transform. A_0 to A_5 address lines on the Eproms were used for specifying the butterfly position, and A_6 to A_8 to specify stage of the computation of the transform. The A_9 address line is used for addressing the twiddle factors for inverse transform.

The addition table storage is quite simple. The first five address lines are for the addend and the next five the adder. The first five address lines on the subtraction table are for the subtractor and the next five for subtrahend.

The inverse look up table TINV and final look up table TFIN are stored such that input modulo 30 is applied on A_0 - A_4 and modulo 31 on A_5 - A_9.

In the index look up table, 31 is stored as the index of zero. In the index addition table, which is same as the standard addition table, 31 is stored in the locations addressed by 31. TINV tables contains zero in the location addressed by 31, so that the correct result of multiplication by zero is obtained.

4.5.3 A Typical Pipeline Interconnection

Fig. 4.18 shows a typical connection between Eproms and the latches. The address to the Eproms comes from the previous stage. Every look up table (Eprom) requires ten address lines, except the tables of residues which require only 8 lines. The other inputs to the Eproms are

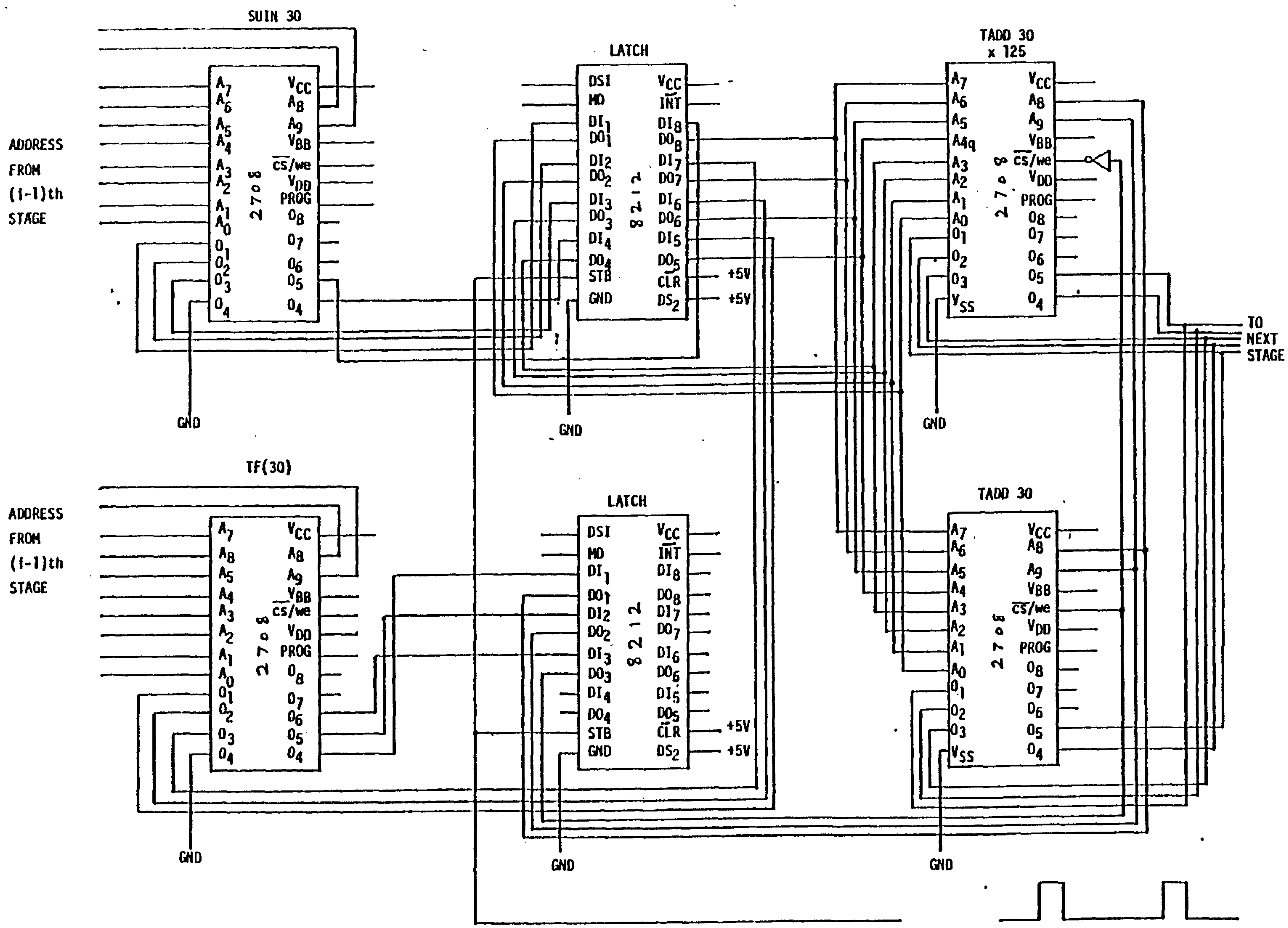


Fig. 4.18 A TYPICAL PIPELINE INTERCONNECTION

connected to the power supply at proper voltage levels as required. The output data from the tables is five bits since all the computation is done using sub-moduli. These data lines are connected to the input of the latch. The remaining three input lines to the latch are obtained from the other table. The 8212 is used as a latch and appropriate input levels are supplied to it. When the clock pulse is applied on the strobe input of 8212, the data from the Eproms is latched and is available on the output lines of 8212 after 30 nsec. Two separate tables for multiplication by twiddle factors are required for even and add powers of α. One of the multiplication pre-multiplication by r for odd powers of α. The sixth bit from the twiddle factor table is used to select/deselect the proper multiplication table. The $\overline{cs}$/we pin on the Eprom is used for selecton of the table. The tristate output of the Eproms enables the connection of the output of two tables together. An inventer is used to select-deselect the tables for even/odd powers of alpha.

Fig. (4.19) to Fig. (4.22) shows the block diagram of the butterfly structures, which was built on protoboards. These figures are included to help the debugging of the unit. Table 4.5 gives the necessary information about the control connections and the power supply connection for both, Eproms and the latches. Fig. 4.23 shows the photograph of the butterfly unit.

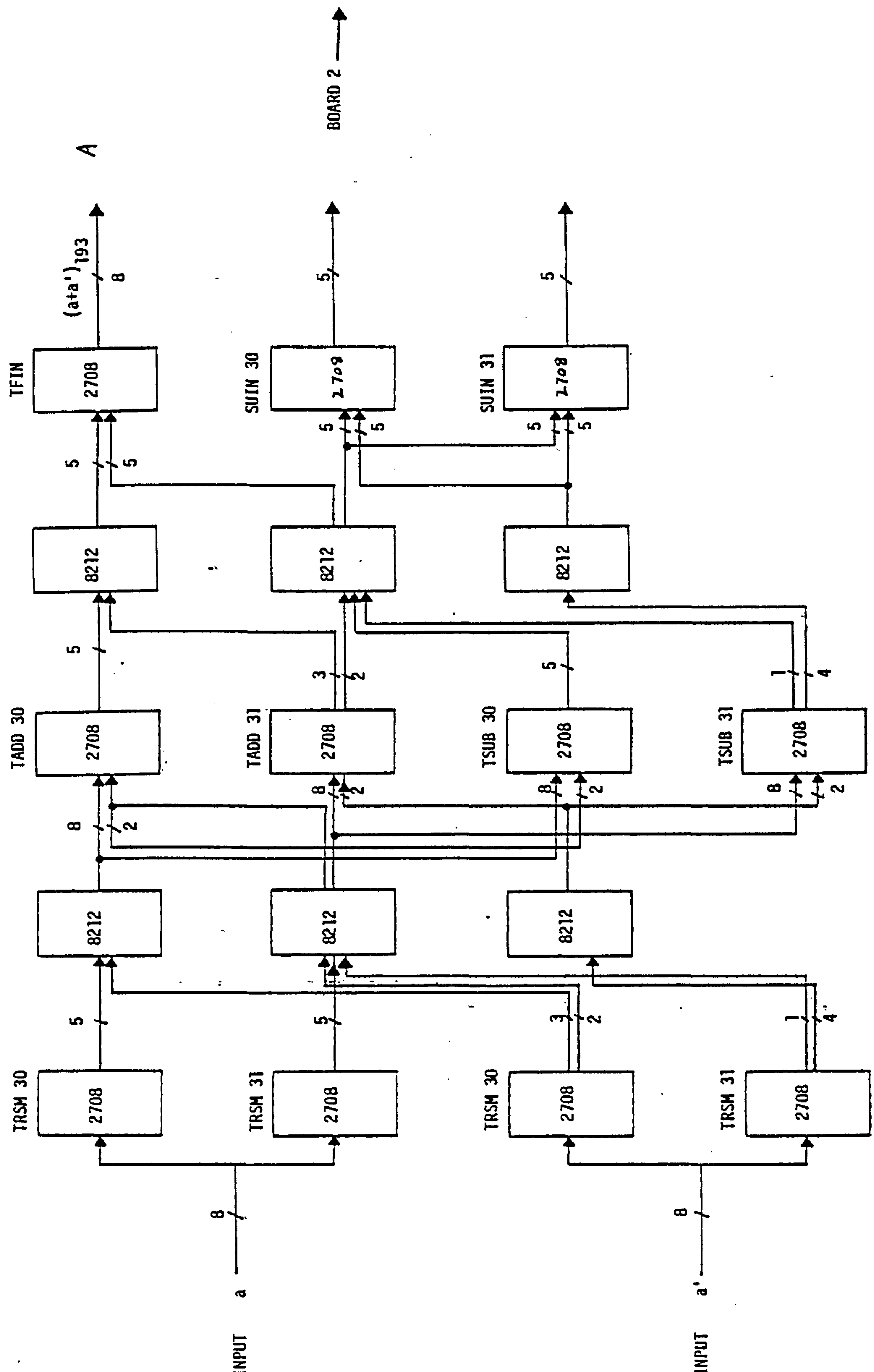


Fig. 4.19 BOARD 1

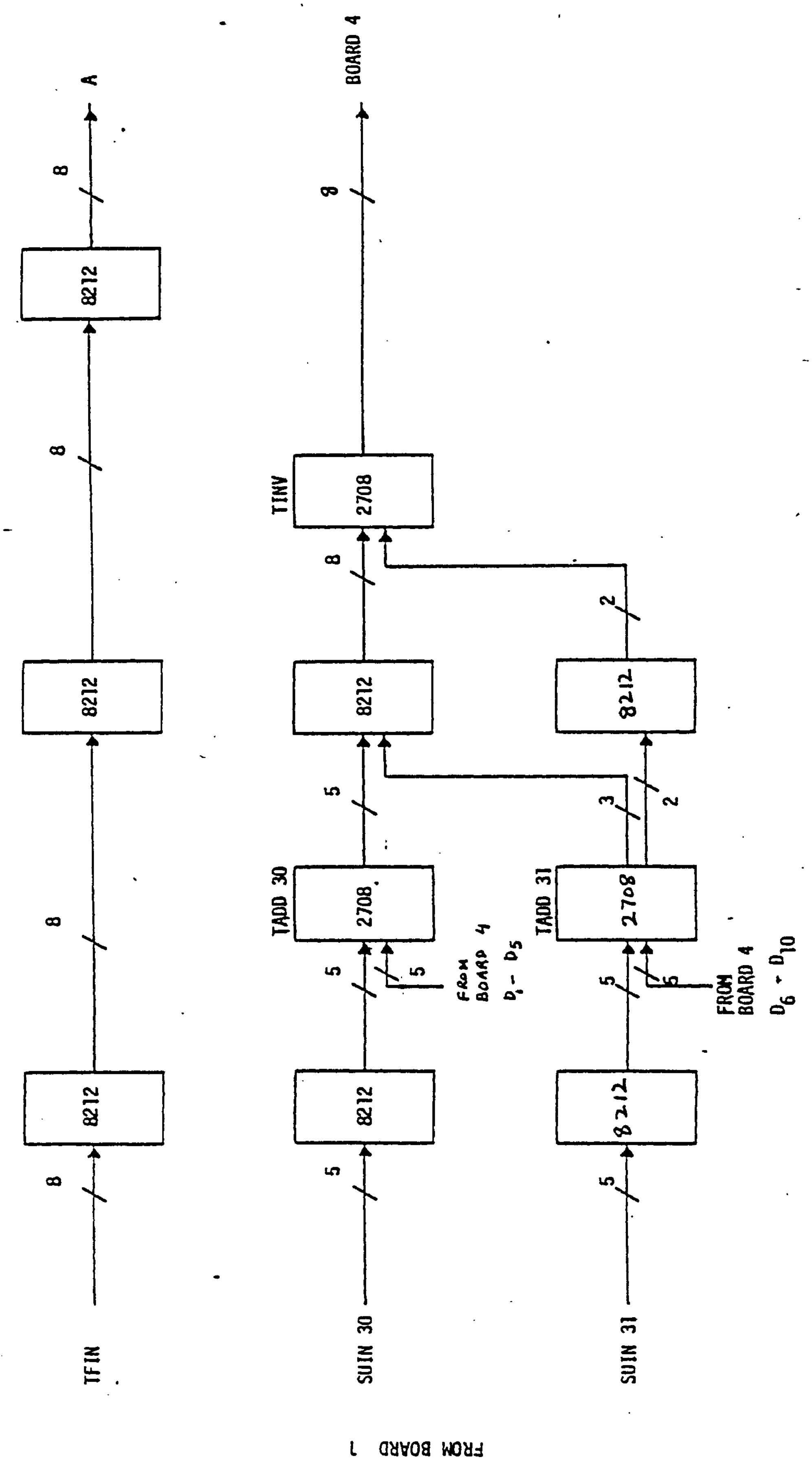


Fig. 4.20 BOARD 2

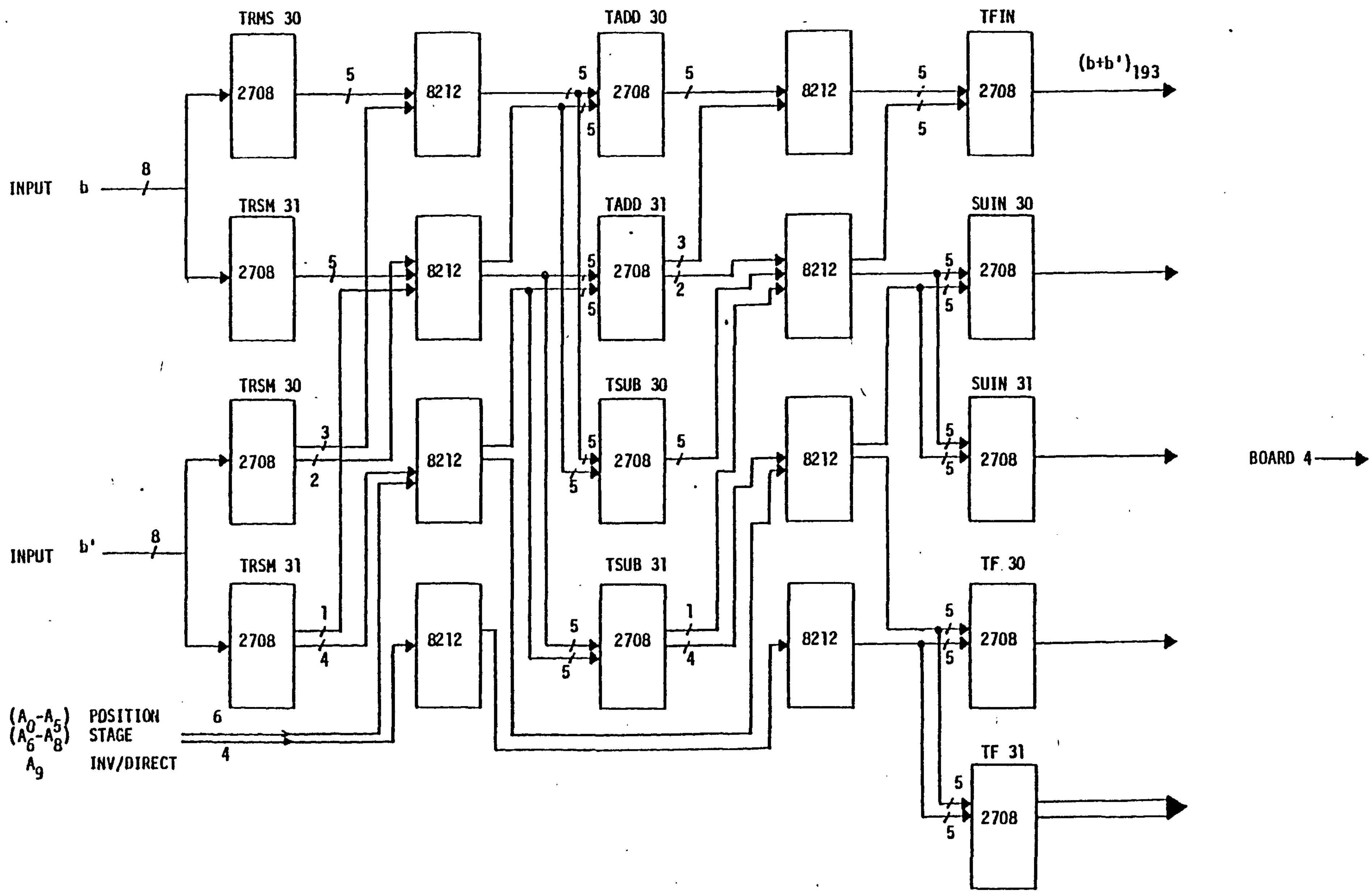


Fig. 4.21 BOARD 3

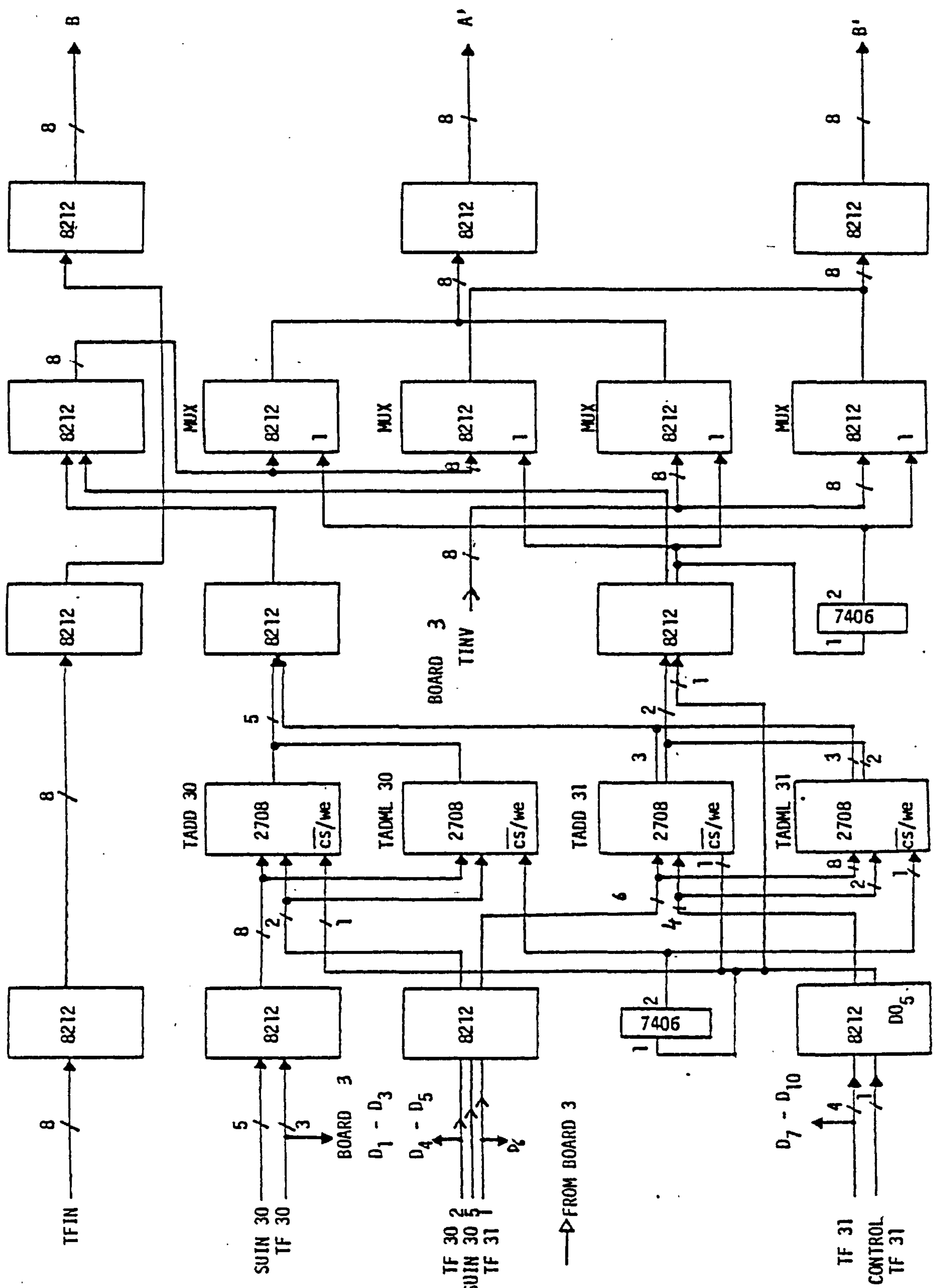


Fig. 4.22 BOARD 4

COLOUR OF WIRES	FUNCTION
BLACK	GND
RED	+ 5 V
ORANGE	- 5 V
BLUE	+ 12V
YELLOW	CONTROLS FOR EVEN/ODD POWER OF ALPHA
WHITE	CLOCK FOR THE LATCHES

Table 4.5 NECESSARY INFORMATION ON THE HARDWARE UNIT.

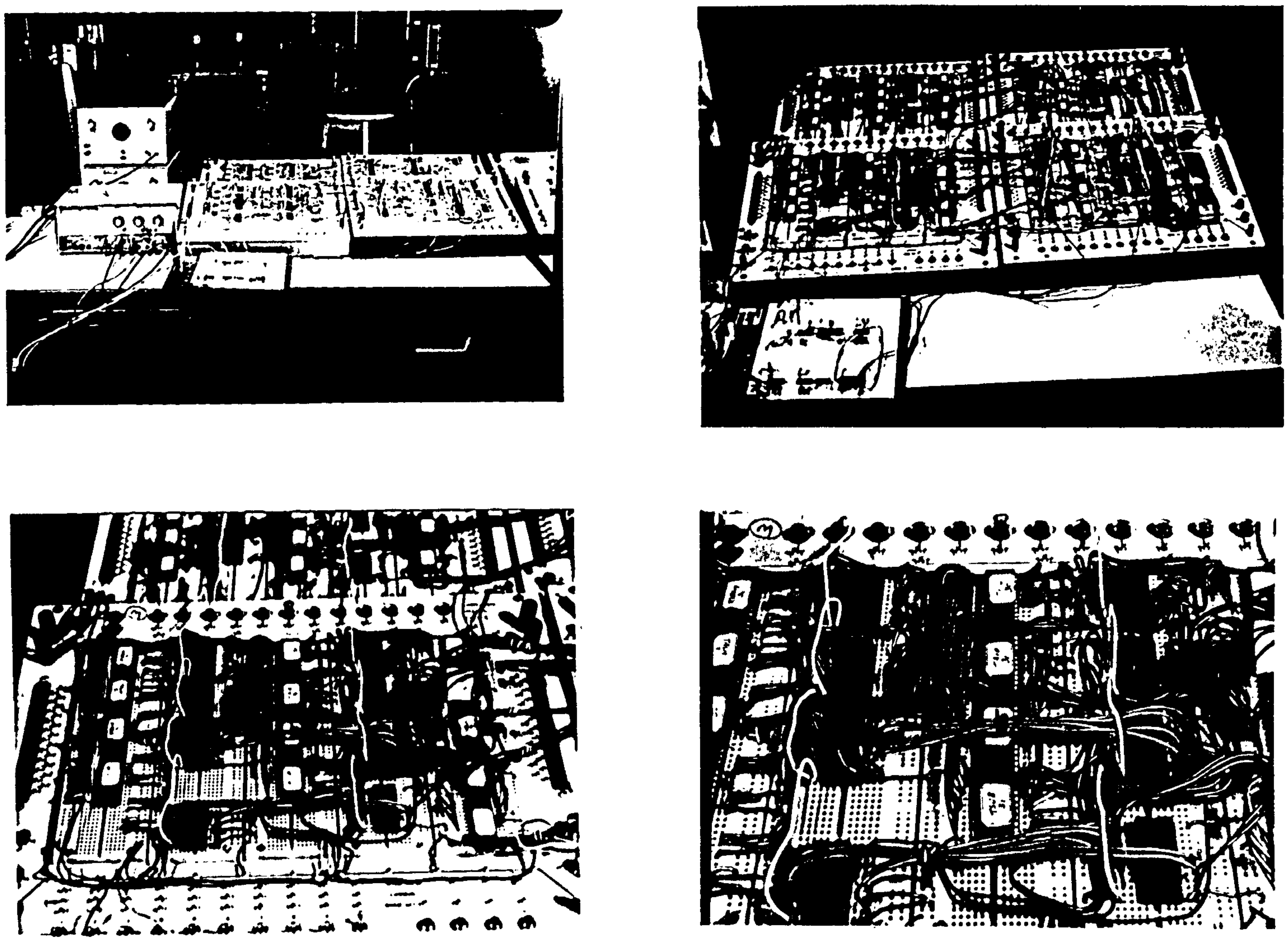


Fig. 4.23 DIFFERENT VIEWS OF THE HARDWARE IMPLEMENTED BUTTERFLY UNIT

4.6 CLOCK CIRCUITRY

A specific clock pulse is required to shift the data in the pipeline. The circuit diagram for the generation of the clock pulse is given in Fig. 4.24. The square wave from the function generator is made TTL compatible by using an NPN transistor. The output from the transistor stage is then fed to a 4 bit binary counter. The outputs of the counter are then fed to one of sixteen decoders. Only one output line of the decoder goes low at each count. This negative going pulse is then fed to an invertor, to obtain a positive going pulse. A buffer is used to supply enough current to operate the latches at each stage.

The alternate pulses were taken from the decoder for each stage. The butterfly unit has five stages of computation and only five pulses from the decoder are used. The frequency of the function generator can be varied up to 1.96 MZ without affecting the working of the pipeline.

4.7 EXPERIMENTAL VERIFICATION

The butterfly structure, was tested for real time application. An input data from the simulation results was used for testing the butterfly. The answer was verified from the simulation results. The input data, the butterfly position and the stage number are:

input point 1	$30 + 65\sqrt{125}$
input point 2	$41 + 103\sqrt{125}$
stage	2
butterfly	4

The value of α for the 2nd stage and the 4th butterfly is 125. The

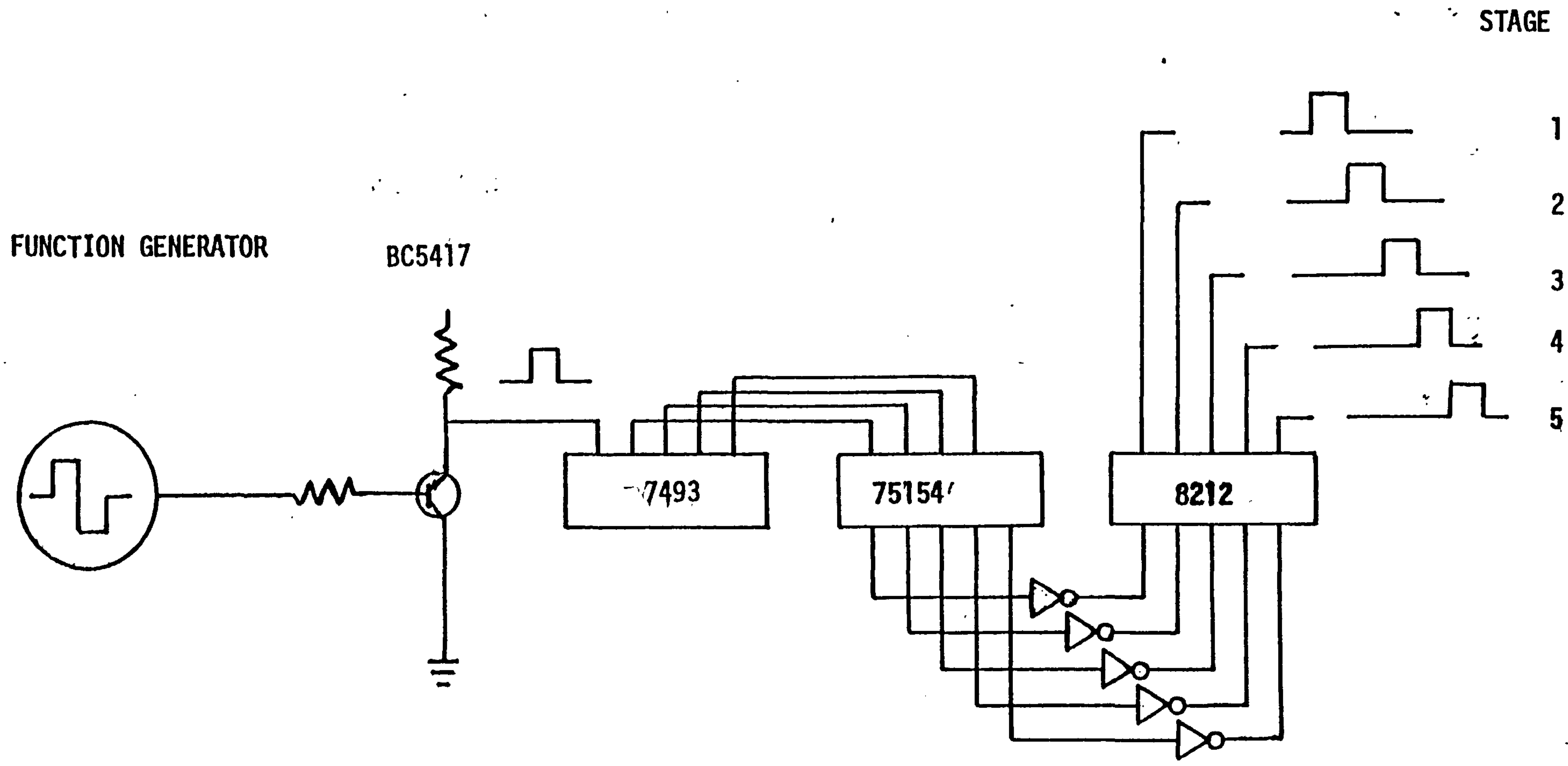


Fig. 4.24 CLOCK CIRCUIT FOR PIPELINE STRUCTURE

butterfly input and outputs are shown in Fig. 4.25(a). The working of the pipeline can not be tested for real time application if the input is fixed. One bit of the data was therefore constantly varied and the intermediate results were checked on a display. The rate of data input and the frequency of the clock pulses were varied to see the effect on the pipeline structure. It was noted that when the clock pulse rate was slower than the rate of change of input data, the output was not correct. The

bit of the data are shown in Fig. 4.25(b). The input and output are:

$$a + \sqrt{125}\, b = 30 + 65\sqrt{125} = 00011110 + 01000001\sqrt{125}$$

$$a' + \sqrt{125}\, b' = 41 + 103\sqrt{125} = 00101001 + 01100110\sqrt{125}$$

$$c + \sqrt{125}\, d = 71 + 168\sqrt{125} = 01000111 + 10101000\sqrt{125}$$

$$c' + \sqrt{125}\, d' = 169 + 75\sqrt{125} = 10101001 + 01001011\sqrt{125}$$

changing the least significant bit of a, gives the results as:

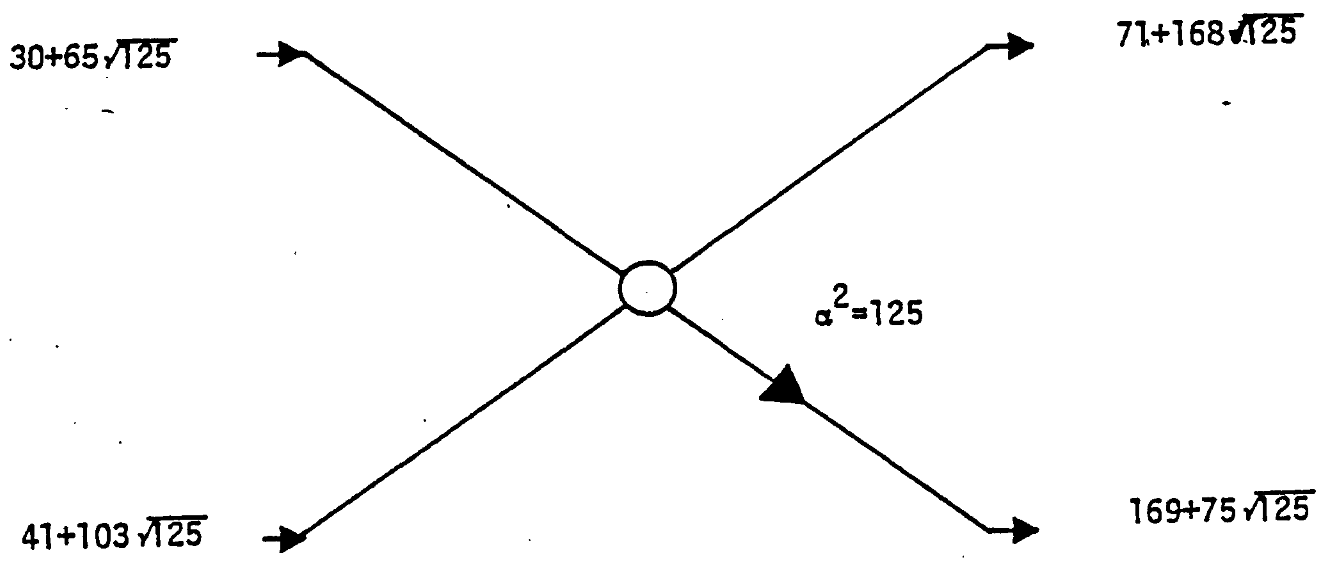
$$a + \sqrt{125}\, b = 31 + 65\sqrt{125} = 00011111 + 01000001\sqrt{125}$$

$$a' + \sqrt{125}\, b' = 41 + 103\sqrt{125} = 00101001 + 01100111\sqrt{125}$$

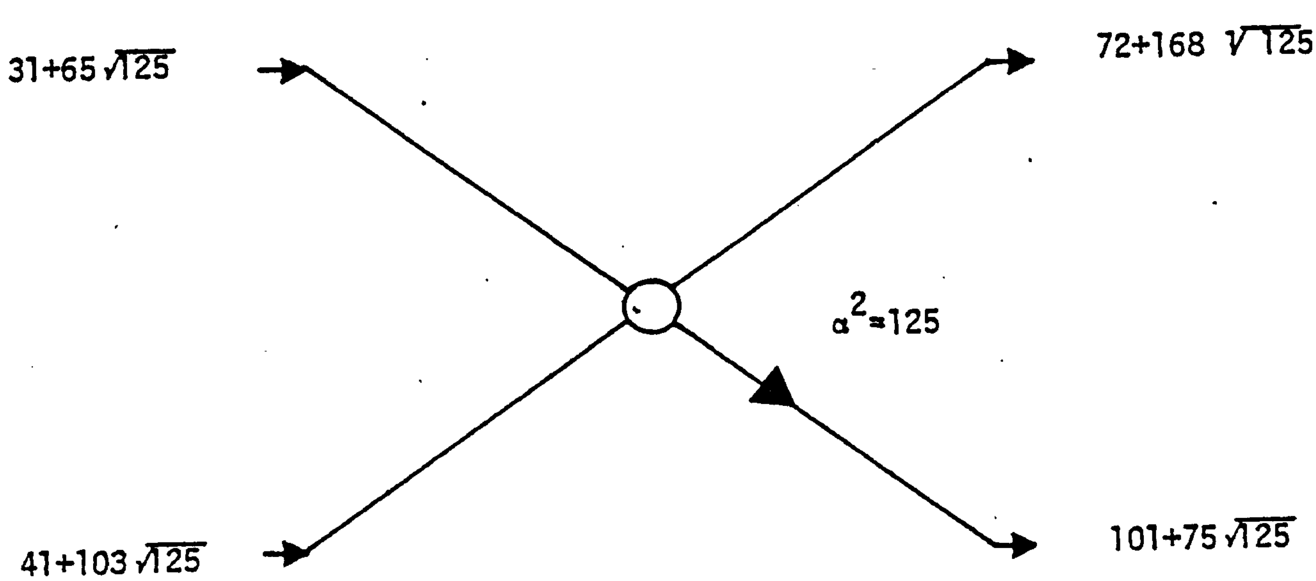
$$c + \sqrt{125}\, d = 72 + 168\sqrt{125} = 01001000 + 10101000\sqrt{125}$$

$$c' + \sqrt{125}\, d' = 101 + 75\sqrt{125} = 01100101 + 01001011\sqrt{125}$$

e.g., by changing the least significant bit of a from 0 to 1 changes the most significant bit of c' from 1 to zero and also the other bits of c and c' change. Thus any bit of c or c' can be checked to verify the working of the pipeline.



a)



b)

Fig. 4.25 INPUT-OUTPUT OF THE BUTTERFLY BEFORE-AFTER CHANGING ONE BIT

4.8 DISCUSSION ON THE HARDWARE REALIZATION OF THE B.F. UNIT

The butterfly unit for modulus 193 was realized using 2708 Eproms and 8212 latches. The addition-subtraction in the B.F. unit is performed using sub-moduli 30 and 31. The multiplication is performed using the index addition method and the addition of the indices is done using the sub-moduli method.

In comparison with the direct method of implementing addition-subtraction using look up tables, the sub-moduli approach offers a saving in the storage for tables. Another way of implementing addition-subtraction is the use of an adder-subtractor followed by a ROM for the correction look up. Fig. 4.26 shows the implementation of addition-subtraction using an adder-subtractor for modulus 193.

The two inputs, modulo 193, are fed to the adder-subtractor and the 9 bit result of addition is then fed to a ROM which contains the corrected result of addition modulo 193. The correct result is stored in the location addressed by the 9 bit result of addition. For example if a=191 and b=189, the result from the adder is 380 and represented as 101111100. The correct result of addition modulo 193 is 187 and therefore 187 can be stored in the location with the address 101111100.

The adder-subtractor which is commercially available, performs 4 bit addition-subtraction and use two's complement arithmetic. The clock to output time is 14 nsec for an Am25LS15 (Advanced Micro-Devices). Addition modulo 193 would require two packages and one ROM. Assuming that the input is in sub-moduli form and no residue tables are required,

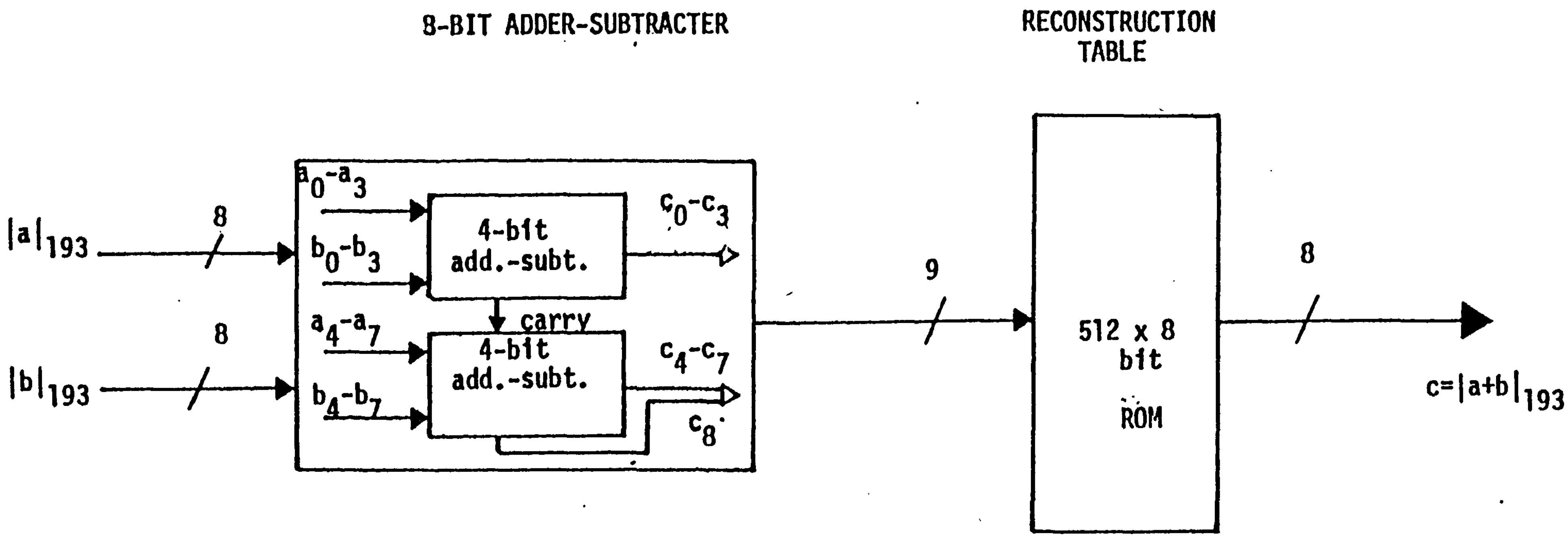


Fig. 4.26 ADDITION MODULO 193 USING ADDER-SUBTRACTER AND ROM

three ROMs are required to perform addition-subtraction using sub-moduli (Fig. 2.4). The package count is the same for adder-subtractor or sub-moduli implementation. If the input is not in sub-modular form, then sub-moduli approach requires 7 ROMs and three stages as compared to 3 packages and 2 stages for adder-subtractor approach. Thus, the choice of implementation depends on the form of the input.

Another criterion for the choice of adder-subtractor is the type of ROMs which are used for the implementation of the complete butterfly structure. The pipeline structure of the butterfly unit requires latches at the output of each computation stage and if Shottky Proms 63RA883 are used, no additional latches are required as these Proms contain latches at the output of the Proms. If the adder-subtractor are used, then for the pipeline structure, an additional 18 bit latch will be required.

The multiplication in the butterfly structure is performed by the addition of indices method. The addition of indices modulo 192 is performed using sub-moduli method. In the sub-moduli approach, the multiplication by zero can be easily corrected and no extra logic is required for detecting the multiplication by zero. However, if the adders are used to perform indices addition, extra logic is required for zero multiplication [13].

The complexity of the structure increases if different kinds of IC's are used. Because of the simplicity of the ROM based structure, the adder-subtractors were not used in the hardware realization and the prototype unit was built using 2708 Eproms. The ROM based structure is preferred because of the fact that it can immediately make use of the advances in the VLSI technology associated with memory fabrication. The

Eproms used can be replaced by fast memory for a particular application.

The 8212 registers were used to latch the data at the (i+1)th stage before the data changes at the ith stage. These registers are level sensitive and the output follows the input as long as the clock is high. A delayed clock pulse for every stage is required as shown in Fig. 4.15. The access time of the ROM is 450 nsec and the latch settling time is 30 nsec. The rate of clock pulses is equal to the access time of the ROM plus two times the latch settling time. From Fig. 4.15 it is seen that there is an overlap at the negative going pulse and the positive going pulse of successive stages, showed in the figure by dotted lines. This overlap created a problem in running the structure for real time data. The clock pulses were generated using a one of 16 decoder and alternate pulses were used to strobe the data so that enough time was available between transitions. This in effect, slowed down the clock rate and the theoretical maximum speed could not be achieved. These latches were used because of their availability.

The remedy to this problem is the use of latches which are edge trigged, e.g., Am 25LS07 (Advanced Micro-Devices). These latches are positive edge triggered and have a latch settling time of 17 nsec. The same clock pulse can be applied to all the stages. At the positive edge, the data will be latched at all the stages and the output of the ROMs will not change until 17 nsec. The clock rate is then the access time of the ROM plus the latch settling time which is now only 17 nsec, and thus the butterfly unit can run at a faster rate.

Assuming that the two output points from the pipelined butterfly unit are obtained after every τ nsec, where τ is equal to the access time of the ROM plus latch settling time, then the time to compute one stage of the NTT transform is equal to 64τ for a transform length of 128 points. The radix 2, 128 point transform length requires seven stages and the time to compute direct or inverse transform of an input sequence is equal to $7 \times 64 \times \tau$ nsec.

The Eproms used in the implementation of the butterfly unit have an access time of 450 nsec and if the AM25LS07 latches are used then τ is equal to 467 nsec and the maximum clock rate is then equal to 2.14 MHz. Thus this butterfly unit can be used with a memory structure which supplies data at 2.14 MHz rate.

4.9 SUMMARY

The design of an NTT processor was described in this Chapter. A study of the supporting memory structure was also undertaken.

The choice of primes for NTT for efficient hardware implementation was discussed and it was shown that 4n + 1 type primes not only require less hardware but also require less number of stages for the butterfly unit. A procedure was described to choose the primes for efficient hardware realization of the butterfly unit. A ROM structure for both kinds of primes for butterfly unit was suggested for pipeline configuration.

The simulation of both kinds of structures was done and the convolution property of NTT for the selected primes was verified. The details of the simulation were presented in this Chapter.

Finally, a butterfly structure for 4n + 1 type primes was built using ROM arrays and a complete discussion was presented. The pipeline structure was tested using time varying data. This butterfly unit, built in hardware, will be used to perform number theoretic transforms with a supporting memory structure.

CHAPTER 5

SUMMARY

The Number Theoretic Transform has a recent origin and is useful for the applications where exact computation is required. The NTT is defined over an finite ring or field and has the same structure as the DFT. It can thus be computed efficiently using fast algorithms for highly composite transform length. A machine that computes the number theoretic transform of a sequence is called an NTT processor. The basic parts of the NTT processor are the supporting memory structure and a computational unit commonly known as the butterfly unit. A saving in hardware of the NTT processor is achieved if a sequential type of processor is built. Such a processor requires some supporting memory and a multiplexed butterfly unit which is accessed $N/r \log_r N$ times where N is the transform length and r is the radix of the fast number theoretic transform. The binary operations of addition, subtraction and the multiplication on the input sequence are performed in the butterfly unit. The parameters of the NTT given by α, N and m, determine the complexity of the butterfly unit. The binary number system has usually been used to perform arithmetic operations in the butterfly unit and consequently restrictions were imposed on the parameters of the NTT to allow for an efficient realization of the computational requirements of the B.F. unit. These restrictions made it impossible to choose the parameters of the NTT on a purely number theoretic basis and thus the efficiency of the NTT was much less than the limiting theoretical efficiency.

The recent advances in VLSI technology associated with memory fabrication have aroused interest in the implementation of the butterfly unit using look up tables stored in high density ROMs. The look-up table approach relaxes the constraints on the parameters of the NTT and thus the theoretical efficiency of the NTT can be reached. If the binary number system is used, then the look-up table approach does not seem very promising because of the tremendous size of memory required to store the tables, e.g., the addition modulo 193 would alone require 64k of memory.

The use of the residue number system allows one to break a large dynamic range problem into a number of smaller dynamic range problems. The combined dynamic range of L moduli is given as $M = \prod_{i=1}^{L} m_i$. These m_i's can be chosen to be small enough for an efficient realization of arithmetic operations moduli m_i. The operations modulo m_i can be performed in parallel because of the interdigit independence property of the residue number system, e.g., modulo 193 addition can be implemented in 7k memory using m_i's as 30 and 31.

The use of the RNS allows one to implement the butterfly unit in look-up tables efficiently. The large dynamic range is achieved by implementing the B.F. in parallel in smaller moduli and then recombining the result using the chinese remainder theorem or a mixed radix conversion scheme.

The modular addition-subtraction requirement in the B.F. unit does not offer any problem and can be efficiently implemented either using the sub-moduli approach or using adders-subtractors. The complexity of performing multiplication by the twiddle factors in the B.F. unit depends upon the field or prime used, and the generator α. The primes are divided into two groups, the $4n + 1$ type and $4n + 3$ type. The $4n + 1$ type primes offer a simpler structure for the butterfly unit, and are preferred over $4n + 3$ type primes.

In this work, the objective was to design a butterfly unit for number theoretic transform capable of exploiting the recent advances in memory technology. A structure for a NTT processor has been developed which is useful for real-time applications. A pipelined butterfly structure was found to be most suitable for use with the supporting memory structure for the real time applications. The butterfly units for both kinds of primes were designed using a pipeline structure. The structure based on the look-up tables stored in ROM is simplest to pipeline, and requires only a clock pulse to shift the data in the pipeline, thus the control circuitry is extremely simple. The package count for the butterfly unit for $4n + 1$ and $4n + 3$ type primes is 32 and 48 respectively including the storage tables for twiddle factors. The number of stages for $4n + 1$ type primes is 5 and for $4n + 3$ is seven.

The butterfly units were simulated on an IBM-370 computer along with the basic required memory structure to establish the feasibility of the proposed NTT processor. After the verification of the simulation results, the butterfly unit for modulo 193 (a 4n + 1 type prime) was implemented in hardware using 2708 Eproms and 8212 latches. The addition subtraction operations were realized using the sub-moduli approach. This approach was used because of the availability of Eproms and also for the simplicity of Eprom based structures. The package count for the butterfly unit is 32 Eproms, 31 latches and 4 multiplexers. The 8212's were used as latches and it was found out that they are not suitable for a pipeline structure since they slow down the speed of operation. Edge trigged latches are recommended. The 8212 latches were used because of their availability.

CHAPTER 6

CONCLUSIONS

The design of a ROM oriented implementation of an NTT butterfly has been carried out in this work. The butterfly has been realized using Eproms and latches in an extremely simple pipeline structure. Level sensitive latches require slower clock rates to function effectively and hence edge triggered latches are preferable.

The addition-subtraction operations have been carried out using sub-moduli approach because of the simplicity of the resulting pipeline structure. The adder-subtractor approach requires less number of stages and a small package count but increases the complexity of the unit. The adder-approach for summing indices to implement multiplication is not that viable as it requires extra logic circuitry to detect zero multiplication and thus further increases the complexity of the butterfly unit.

A memory support structure has been simulated at the logic level in order to investigate the feasibility of the NTT processor described in the thesis. The proposed structure is such that the data transfer time associated with the memory is the same as the computational time of the butterfly unit. This further enhances the NTT processor's capability as a real-time signal processing facility. The memory structure can be realized using long shift registers for the dynamic storage of the data. The use of these shift registers eliminates the need for addressing the data.

Future work in this area will be to actually implement the memory support structure in hardware and to ultimately construct the complete NTT processor.

APPENDIX A

SIMULATION PROGRAMS

The simulation of the butterfly unit was done on an IBM-370-3031 computer. Listings of the programs are given here.

```
C          ***************************************************************
C          *    MAIN PROGRAM TO ESTABLISH THE INVERTIBILITY OF NTT        *
C          *                                                              *
C          ***************************************************************
      IMPLICIT INTEGER(A-H,O-Z)
      DIMENSION NUOP(128,2),TEM(128,2)
      DIMENSION INP1(2),INP2(2),OUT1(2),OUT2(2)
      COMMON TINV(2,32,32),TSUB(2,31,31),TFIN(30,31),TFI(2,64,2)
      COMMON TRSM(2,191),TADD(2,32,32),TSUIN(2,31,31),TF(2,64,2)
      COMMON POINT
C
C     GENERATE THE TABLES FOR NTT BUTTERFLY
C
C     MMOD IS THE MAIN MODULUS.NINV IS THE MULTIPLICATIVE INVERSE OF
      READ(5,201) MMOD,NINV
201   FORMAT(I3,5X,I3)
      CALL TABLES
C
C     SET THE POINTER FOR CLEARING LATCHES
C
      POINT=0
      INV=0
C
C     INITILIZE THE INPUT
C
      DO 50 I=1,128
      NUOP(I,1)=(I-1)
      NUOP(I,2)=(128-I)
   50 CONTINUE
C
C     START THE COMPUTATION FOR SEVEN STAGES.L IS FOR STAGE
C
   20 DO 1 L=1,7
      STG=L-1
      NJ=1
C
C     NN IS FOR THE POSITION OF BUTTERFLY IN THE STAGE
C
      DO 2 NN=1,64
      INP1(1)=NUOP(NN,1)
      INP1(2)=NUOP(NN,2)
      INP2(1)=NUOP(NN+64,1)
      INP2(2)=NUOP(NN+64,2)
      POST=NN-1
C
C     CALL THE NTT TO COMPUTE TWO POINT BUTTERFLY
C
      CALL NTT(INV,INP1,INP2,STG,POST,OUT1,OUT2)
C
C     DO NOT STORE OUTPUT FOR INITIAL 7 STAGE DELAY
C
      IF(NN.LT.8) GO TO 2
      TEM(NJ,1)=OUT1(1)
      TEM(NJ,2)=OUT1(2)
      TEM(NJ+1,1)=OUT2(1)
      TEM(NJ+1,2)=OUT2(2)
      NJ=NJ+2
    2 CONTINUE
```

```
C
C     OBTAIN THE LAST SEVEN POINTS FROM THE PIPELINE BUTTERFLY UNIT
C
      DO 4 JJ=1,7
      INP1(1)=INP1(2)=INP2(1)=INP2(2)=0
      CALL NTT(INV,INP1,INP2,STG,POST,OUT1,OUT2)
      TEM(NJ,1)=OUT1(1)
      TEM(NJ,2)=OUT1(2)
      TEM(NJ+1,1)=OUT2(1)
      TEM(NJ+1,2)=OUT2(2)
      NJ=NJ+2
    4 CONTINUE
      DO 5 K=1,128
      DO 5 KK=1,2
      NUOP(K,KK)=TEM(K,KK)
    5 CONTINUE
    1 CONTINUE
C
C     USE THE SHUFFLE PROGRAM TO OBTAIN THE ORDERED OUTPUT
C
      NU=128
      NV2=NU/2
      NM1=NU-1
      J=1
      DO 207 I=1,NM1
      IF(I.GE.J) GO TO 255
      RE=NUOP(J,1)
      NUOP(J,1)=NUOP(I,1)
      NUOP(I,1)=RE
      IM=NUOP(J,2)
      NUOP(J,2)=NUOP(I,2)
      NUOP(I,2)=IM
  255 K=NV2
  206 IF(K.GE.J) GO TO 207
      J=J-K
      K=K/2
      GO TO 206
  207 J=J+K
      PRINT 101,KOUNT,INV
  101 FORMAT('-',20X,'KOUNT=',I2,20X,'INV=',I2)
      DO 56 J=1,2
      PRINT,(NUOP(NN,J),NN=1,128)
   56 CONTINUE
      IF(INV.EQ.1) GO TO 103
      INV=1
      GO TO 20
  103 DO 110 I=1,128
      DO 110 J=1,2
      A=NUOP(I,J)*NINV
      NUOP(I,J)=MOD(A,191)
  110 CONTINUE
      PRINT 70
   70 FORMAT('-',20X,'THIS IS THE FINAL RESULT')
      DO 71 J=1,2
      PRINT,(NUOP(MM,J),MM=1,128)
   71 CONTINUE
      STOP
```

```
C          ****************************************************************
C          *   MAIN PROGRAM TO CONVOLVE TWO SEQUENCES. IMAGINARY          *
C          *   PARTS ARE ZERO                                             *
C          *                                                              *
C          ****************************************************************
      IMPLICIT INTEGER(A-H,O-Z)
      DIMENSION NUOP(128,2),TEM(128,2),TEM1(128,2)
      DIMENSION INP1(2),INP2(2),OUT1(2),OUT2(2)
      COMMON TINV(2,32,32),TSUB(2,31,31),TFIN(30,31),TFI(2,64,2)
      COMMON TRSM(2,191),TADD(2,32,32),TSUIN(2,31,31),TF(2,64,2)
      COMMON POINT
C
C     GENERATE THE TABLES FOR NTT BUTTERFLY
C
C     MMOD IS THE MAIN MODDULUS. NINV IS THE MULTIPLICATIVE INVERSE OF
      READ(5,201) MMOD,NINV
201   FORMAT(I3,5X,I3)
      CALL TABLES
C
C     SET THE POINTER FOR CLEARING LATCHES
C
      POINT=0
C
C     INITILIZE THE COUNTER TO PERFORM TRANSFORMS
C
      KOUNT=0
      INV=0
C
C     INITILIZE THE INPUT
C
      DO 50 I=1,64
      NUOP(I,1)=1
      NUOP(I,2)=0
   50 CONTINUE
      DO 51 I=65,128
      NUOP(I,1)=0
      NUOP(I,2)=0
   51 CONTINUE
C
C     START THE COMPUTATION FOR SEVEN STAGES. L IS FOR STAGE
C
   20 DO 1 L=1,7
      STG=L-1
      NJ=1
C
C     NN IS FOR THE POSITION OF BUTTERFLY IN THE STAGE
C
      DO 2 NN=1,64
      INP1(1)=NUOP(NN,1)
      INP1(2)=NUOP(NN,2)
      INP2(1)=NUOP(NN+64,1)
      INP2(2)=NUOP(NN+64,2)
      POST=NN-1
C
C     CALL THE NTT TO COMPUTE TWO POINT BUTTERFLY
C
      CALL NTT(INV,INP1,INP2,STG,POST,OUT1,OUT2)
```

```
C
C     DO NOT STORE OUTPUT FOR INITIAL 7 STAGE DELAY
C
      IF(NN.LT.8) GO TO 2
      TEM(NJ,1)=OUT1(1)
      TEM(NJ,2)=OUT1(2)
      TEM(NJ+1,1)=OUT2(1)
      TEM(NJ+1,2)=OUT2(2)
      NJ=NJ+2
    2 CONTINUE
C
C     OBTAIN THE LAST SEVEN POINTS FROM THE PIPELINE BUTTERFLY UNIT
C
      DO 4 JJ=1,7
      INP1(1)=INP1(2)=INP2(1)=INP2(2)=0
      CALL NTT(INV,INP1,INP2,STG,POST,OUT1,OUT2)
      TEM(NJ,1)=OUT1(1)
      TEM(NJ,2)=OUT1(2)
      TEM(NJ+1,1)=OUT2(1)
      TEM(NJ+1,2)=OUT2(2)
      NJ=NJ+2
    4 CONTINUE
      DO 5 K=1,128
      DO 5 KK=1,2
      NUOP(K,KK)=TEM(K,KK)
    5 CONTINUE
    1 CONTINUE
C
C     USE THE SHUFFLE PROGRAM TO OBTAIN THE ORDERED OUTPUT
C
      NU=128
      NV2=NU/2
      NM1=NU-1
      J=1
      DO 207 I=1,NM1
      IF(I.GE.J) GO TO 255
      RE=NUOP(J,1)
      NUOP(J,1)=NUOP(I,1)
      NUOP(I,1)=RE
      IM=NUOP(J,2)
      NUOP(J,2)=NUOP(I,2)
      NUOP(I,2)=IM
  255 K=NV2
  206 IF(K.GE.J) GO TO 207
      J=J-K
      K=K/2
      GO TO 206
  207 J=J+K
      PRINT 101,KOUNT,INV
  101 FORMAT('-',20X,'KOUNT=',I2,20X,'INV=',I2)
      DO 56 J=1,2
      PRINT,(NUOP(NN,J),NN=1,128)
   56 CONTINUE
      KOUNT=KOUNT+1
      IF(INV.EQ.1) GO TO 103
      IF(KOUNT.GT.1) GO TO 102
C
C     STORE THE FIRST TRANSFORMED SEQUENCE
```

```
      DO 57 I=1,128
      DO 57 J=1,2
      TEM1(I,J)=NUOP(I,J)
   57 CONTINUE
C
C     INITILIZE THE SECOND SEQUENCE
C
      DO 58 I=1,64
      NUOP(I,1)=2
      NUOP(I,2)=0
   58 CONTINUE
      DO 59 I=65,128
      DO 59 J=1,2
      NUOP(I,J)=0
   59 CONTINUE
C
C     TRANSFORM THE SECOND SEQUENCE
C
      GO TO 20
C
C     MULTIPLY THE SEQUENCES IN THE TRANSFORNS DOMAIN
C
  102 DO 60 I=1,128
      A=NUOP(I,1)
      B=NUOP(I,2)
      C=TEM1(I,1)
      D=TEM1(I,2)
      AC=A*C




      AC=MOD(AC,MMOD)
      BD=B*D
      BD=MOD(BD,MMOD)
      AD=A*D
      AD=MOD(AD,MMOD)
      BC=B*C
      BC=MOD(BC,MMOD)
      RE=AC-BD
      RE=MOD(RE,MMOD)
      IF(RE.LT.0) RE=RE+MMOD
      NUOP(I,1)=RE
      IMA=AD+BC
      NUOP(I,2)=MOD(IMA,MMOD)
   60 CONTINUE
C
C     TAKE THE INVERSE TRANSFORM
C
      INV=1
      GO TO 20
C
C     MULTIPLY WITH N INVERSE
C
  107 DO 110 I=1,128
      DO 110 J=1,2
      A=NUOP(I,J)*NINV
      NUOP(I,J)=MOD(A,MMOD)
  110 CONTINUE
      PRINT 70
   70 FORMAT('-',20X,'THIS IS THE FINAL RESULT')
      DO 71 J=1,2
      PRINT,(NUOP(MM,J),MM=1,128)
   71 CONTINUE
      ----
```

```
      ***********************************************************************
      *                                                                     *
      * MAIN PROGRAM TO CONVOLVE A COMPLEX SEQUENCE.                        *
      * SECOND SEQUENCE IS REAL.                                            *
      *                                                                     *
      ***********************************************************************
      IMPLICIT INTEGER(A-H,O-Z)
      DIMENSION NUOP(128,2),TEM(128,2),TEM1(128,2)
      DIMENSION INP1(2),INP2(2),OUT1(2),OUT2(2)
      COMMON TINV(2,32,32),TSUB(2,31,31),TFIN(30,31),TFI(2,64,2)
      COMMON TRSM(2,191),TADD(2,32,32),TSUIN(2,31,31),TF(2,64,2)
      COMMON POINT
C
C     GENERATE THE TABLES FOR NTT BUTTERFLY
C
C     MMOD IS THE MAIN MODDULUS.NINV IS THE MULTIPLICATIVE INVERSE OF N
      READ(5,201) MMOD,NINV
201   FORMAT(I3,5X,I3)
      CALL TABLES
C
C     SET THE POINTER FOR CLEARING LATCHES
C
      POINT=0
C
C     INITILIZE THE COUNTER TO PERFORM TRANSFORMS
C
      KOUNT=0
      INV=0
C
C     INITILIZE THE INPUT
C
      DO 50 I=1,64
      NUOP(I,1)=1
      NUOP(I,2)=0
   50 CONTINUE
      DO 51 I=65,128
      NUOP(I,1)=0
      NUOP(I,2)=0
   51 CONTINUE
C
C     START THE COMPUTATION FOR SEVEN STAGES.L IS FOR STAGE
C
   20 DO 1 L=1,7
      STG=L-1
      NJ=1
C
C     NN IS FOR THE POSITION OF BUTTERFLY IN THE STAGE
C
      DO 2 NN=1,64
      INP1(1)=NUOP(NN,1)
      INP1(2)=NUOP(NN,2)
      INP2(1)=NUOP(NN+64,1)
      INP2(2)=NUOP(NN+64,2)
      POST=NN-1
C
C     CALL THE NTT TO COMPUTE TWO POINT BUTTERFLY
C
      CALL NTT(INV,INP1,INP2,STG,POST,OUT1,OUT2)
```

```
C
C       DO NOT STORE OUTPUT FOR INITIAL 7 STAGE DELAY
C
        IF(NN.LT.8) GO TO 2
        TEM(NJ,1)=OUT1(1)
        TEM(NJ,2)=OUT1(2)
        TEM(NJ+1,1)=OUT2(1)
        TEM(NJ+1,2)=OUT2(2)
        NJ=NJ+2
      2 CONTINUE
C
C       OBTAIN THE LAST SEVEN POINTS FROM THE PIPELINE BUTTERFLY UNIT
C
        DO 4 JJ=1,7
        INP1(1)=INP1(2)=INP2(1)=INP2(2)=0
        CALL NTT(INV,INP1,INP2,STG,POST,OUT1,OUT2)
        TEM(NJ,1)=OUT1(1)
        TEM(NJ,2)=OUT1(2)
        TEM(NJ+1,1)=OUT2(1)
        TEM(NJ+1,2)=OUT2(2)
        NJ=NJ+2
      4 CONTINUE
        DO 5 K=1,128
        DO 5 KK=1,2
        NUOP(K,KK)=TEM(K,KK)
      5 CONTINUE
      1 CONTINUE
C
C       USE THE SHUFFLE PROGRAM TO OBTAIN THE ORDERED OUTPUT
C
        NU=128
        NV2=NU/2
        NM1=NU-1
        J=1
        DO 207 I=1,NM1
        IF(I.GE.J) GO TO 255
        RE=NUOP(J,1)
        NUOP(J,1)=NUOP(I,1)
        NUOP(I,1)=RE
        IM=NUOP(J,2)
        NUOP(J,2)=NUOP(I,2)
        NUOP(I,2)=IM
    255 K=NV2
    206 IF(K.GE.J) GO TO 207
        J=J-K
        K=K/2
        GO TO 206
    207 J=J+K
        PRINT 101,KOUNT,INV
    101 FORMAT('-',20X,'KOUNT=',I2,20X,'INV=',I2)
        DO 56 J=1,2
        PRINT,(NUOP(NN,J),NN=1,128)
     56 CONTINUE
        KOUNT=KOUNT+1
        IF(INV.EQ.1) GO TO 103
        IF(KOUNT.GT.1) GO TO 102
C
C       STORE THE FIRST TRANSFORMED SEQUENCE
C
```

```
      DO 57 I=1,128
      DO 57 J=1,2
      TEM1(I,J)=NUOP(I,J)
   57 CONTINUE
C
C     INITILIZE THE SECOND SEQUENCE
C
      DO 58 I=1,64
      NUOP(I,1)=2
      NUOP(I,2)=0
   58 CONTINUE
      DO 59 I=65,128
      DO 59 J=1,2
      NUOP(I,J)=0
   59 CONTINUE
C
C     TRANSFORM THE SECOND SEQUENCE
C
      GO TO 20
C
C     MULTIPLY THE SEQUENCES IN THE TRANSFORME DOMAIN
C
  102 DO 60 I=1,128
      A=NUOP(I,1)
      B=NUOP(I,2)
      C=TEM1(I,1)
      D=TEM1(I,2)
      AC=A*C



      AC=MOD(AC,MMOD)
      BD=B*D
      BD=MOD(BD,MMOD)
      AD=A*D
      AD=MOD(AD,MMOD)
      BC=B*C
      BC=MOD(BC,MMOD)
      RE=AC-BD
      RE=MOD(RE,MMOD)
      IF(RE.LT.0) RE=RE+MMOD
      NUOP(I,1)=RE
      IMA=AD+BC
      NUOP(I,2)=MOD(IMA,MMOD)
   60 CONTINUE
C
C     TAKE THE INVERSE TRANSFORM
C
      INV=1
      GO TO 20
C
C     MULTIPLY WITH N INVERSE
C
  103 DO 110 I=1,128
      DO 110 J=1,2
      A=NUOP(I,J)*NINV
      NUOP(I,J)=MOD(A,MMOD)
  110 CONTINUE
      PRINT 70
   70 FORMAT('-',20X,'THIS IS THE FINAL RESULT')
      DO 71 J=1,2
      PRINT,(NUOP(MM,J),MM=1,128)
   71 CONTINUE
      STOP
      END
```

```
C         ************************************************************
C         *                                                          *
C         *  SUBROUTINE TO GENERATE LOOK-UP TABLES FOR NTT          *
C         *   MODULUS=191                                            *
C         ************************************************************
      SUBROUTINE TABLES
      IMPLICIT INTEGER(A-H,O-Z)
      DIMENSION MODU(2),IND(191),IIND(191),ALPA(2,128,2)
      COMMON TINV(2,32,32),TSUB(2,31,31),TFIN(30,31),TFI(2,64,2)
      COMMON TRSM(2,191),TADD(2,32,32),TSUIN(2,31,31),TF(2,64,2)
      COMMON POINT
      MMOD=191
      PRIM=19
      MODU(1)=30;MODU(2)=31
      IND(1)=31;IND(2)=0;IIND(1)=1
      PER=MMOD-2
      PRO=MODU(1)*MODU(2)
      NMOD=MMOD-1
      VAL=1
C
C     INDEX TABLE
C
      DO 21 K=1,PER
      VAL=VAL*PRIM
      VAL=MOD(VAL,MMOD)
      IND(VAL+1)=K
      IIND(K+1)=VAL
   21 CONTINUE
C
C     SUB MODULI RESIDUE TABLE
C
      DO 1 I=1,2
      DO 2 N=1,MMOD
      A=N-1
      TRSM(I,N)=MOD(A,MODU(I))
    2 CONTINUE
      MM=MODU(I)
C
C     ADDITION TABLE
C
      DO 11 N=1,MM
      DO 11 NN=1,MM
      A=N+NN-2
      TADD(I,N,NN)=MOD(A,MODU(I))
   11 CONTINUE
C
C     SUBTRACTION TABLE
C
      DO 31 K=1,MM
      DO 32 J=1,MM
      A=K-J
      IF(A.LT.0) A=A+MM
      TSUB(I,K,J)=A
   32 CONTINUE
   31 CONTINUE
    1 CONTINUE
```

```
C
C     INVERSE LOOK UP TABLE
C
      DO 50 II=1,30
      DO 51 JJ=1,31
      X1=(II-1)*1
      X2=(JJ-1)*30
      X2=MOD(X2,31)
      X3=(X1*MODU(2)+X2*MODU(1))




      X4=MOD(X3,PRO)
      X5=MOD(X4,NMOD)
      X=IIND(X5+1)
      TINV(1,II,JJ)=MOD(X,MODU(1))
      TINV(2,II,JJ)=MOD(X,MODU(2))
C
C     TABLE FOR FINAL LOOKUP
C
      MODS=PRO-MMOD
      IF(X4.LE.MODS) GO TO 52
      X4=X4-PRO
      X4=X4+MMOD
      GO TO 53
   52 X4=MOD(X4,MMOD)
   53 TFIN(II,JJ)=X4
C
C     SUBTRACTION TABLE INDEXED
C
      B=IND(X4+1)
      TSUIN(1,II,JJ)=MOD(B,30)
      IF(X4.EQ.0) TSUIN(1,II,JJ)=31
      TSUIN(2,II,JJ)=MOD(B,31)
      IF(X4.EQ.0) TSUIN(2,II,JJ)=31
   51 CONTINUE
   50 CONTINUE
C
C     TABLE FOR POWERS OF ALPHA
C
      AA=66;BB=6;CC=66;DD=6
      A=IND(2)
      C=IND(67)
      D=IND(7)
      ALPA(1,1,1)=MOD(A,30)
      ALPA(2,1,1)=MOD(A,31)
      ALPA(1,1,2)=31
      ALPA(2,1,2)=31
      ALPA(1,2,1)=MOD(C,30)
      ALPA(2,2,1)=MOD(C,31)
      ALPA(1,2,2)=MOD(D,30)
      ALPA(2,2,2)=MOD(D,31)
      DO 61 NN=2,127
      RE=AA*CC-BB*DD
      RE=MOD(RE,MMOD)
      IF(RE.LT.0) RE=RE+MMOD
      IMAG=AA*DD+BB*CC
      IMAG=MOD(IMAG,MMOD)
      A=IND(RE+1)
      B=IND(IMAG+1)
      ALPA(1,NN+1,1)=MOD(A,30)
      IF(A.EQ.31) ALPA(1,NN+1,1)=31
      ALPA(2,NN+1,1)=MOD(A,31)
      IF(A.EQ.31) ALPA(2,NN+1,1)=31
```

```
      ALPA(1,NN+1,2)=MOD(B,30)
      IF(B.EQ.31) ALPA(1,NN+1,2)=31
      ALPA(2,NN+1,2)=MOD(B,31)
      IF(B.EQ.31) ALPA(2,NN+1,2)=31
      AA=RE
      BB=IMAG
   61 CONTINUE
C
C     TABLE FOR TWIDDLE FACTORS
C
      DO 62 II=1,64
      TF(1,II,1)=ALPA(1,II,1)
      TF(1,II,2)=ALPA(1,II,2)



      TF(2,II,1)=ALPA(2,II,1)
      TF(2,II,2)=ALPA(2,II,2)
C
C     TABLE FOR INVERSE NTT TWIDDLE FACTORS
C
      INK=II-1
      IF(INK.NE.0) INK=128-INK
      TFI(1,II,1)=ALPA(1,INK+1,1)
      TFI(1,II,2)=ALPA(1,INK+1,2)
      TFI(2,II,1)=ALPA(2,INK+1,1)
      TFI(2,II,2)=ALPA(2,INK+1,2)
      TT=INK+1
   62 CONTINUE
C
C     CORRECTION FOR ZERO MULTIPLICATION
C
      DO 251 I=1,2
      DO 252 J=1,32
      TADD(I,32,J)=31
      TINV(I,32,J)=0
  252 CONTINUE
      DO 253 J=1,32
      TADD(I,J,32)=31
      TINV(I,J,32)=0
  253 CONTINUE
  251 CONTINUE
      RETURN
      END
```

```
C          ************************************************************
C          *                                                          *
C          *  SUBROUTINE TO GENERATE LOOK-UP TABLES FOR NTT           *
C          *   MODULUS=193                                            *
C          ************************************************************
      SUBROUTINE TABLES
      IMPLICIT INTEGER(A-H,O-Z)
      COMMON TRSM(2,193),TADD(2,32,32),TADMUL(2,32,32),TSUB(2,31,31)
      COMMON TSUIN(2,31,31),TINV(32,32),TFIN(30,31),TF(2,64),TFI(2,64)
      COMMON POINT
      DIMENSION MODU(2),IND(193),IIND(193),ALPA(2,128)
      MMOD=193;PRIM=5;IND(1)=31;IND(2)=0;IIND(1)=1
      MODU(1)=30;MODU(2)=31
      PER=MMOD-2
      NMOD=MMOD-1
      PRO=MODU(1)*MODU(2)
      MODS=PRO-MMOD
      VAL=1
C
C     INDEX TABLE
C
      DO 21 K=1,PER
      VAL=VAL*PRIM
      VAL=MOD(VAL,MMOD)
      IND(VAL+1)=K
      IIND(K+1)=VAL
   21 CONTINUE
C
C     TABLE FOR POWER OF ALPA
C
      AA=0;BB=1;CC=0;DD=1
      ALPA(1,1)=0;ALPA(2,1)=0;ALPA(1,2)=0;ALPA(2,2)=0
      DO 31 NN=2,127
      RE=AA*CC+125*BB*DD
      IMAG=AA*DD+CC*BB
      RE=MOD(RE,MMOD)
      IMAG=MOD(IMAG,MMOD)
      IF(RE.NE.0) TEM=RE
      IF(IMAG.NE.0) TEM=IMAG
      A=IND(TEM+1)
      ALPA(1,NN+1)=MOD(A,30)
      ALPA(2,NN+1)=MOD(A,31)
      AA=RE
      BB=IMAG
   31 CONTINUE
C
C     TABLE FOR TWIDDLE FACTORS
```

```
      DO 34 II=1,64
      DO 35 I=1,2
      TF(I,II)=ALPA(I,II)
C
C     TABLE FOR INVERSE TWIDDLE FACTORS
C
      INK=II-1
      IF(INK.NE.0) INK=128-INK
      TFI(I,II)=ALPA(I,INK+1)
   35 CONTINUE
   34 CONTINUE
      DO 1 I=1,2
      MSUB=MODU(I)
C
C     SUB-MODULI RESIDUE TABLE
C
      DO 2 N=1,MMOD




      A=N-1
      TRSM(I,N)=MOD(A,MSUB)
    2 CONTINUE
C
C     ADDITION TABLE
C
      DO 11 N=1,MSUB
      DO 11 NN=1,MSUB
      A=N+NN-2
      TADD(I,N,NN)=MOD(A,MSUB)
C
C     TABLE WITH MULTIPLIER
C
      B=N+NN-2+3
      TADMUL(I,N,NN)=MOD(B,MSUB)
   11 CONTINUE
C
C     SUBTRACTION TABLE
C
      DO 15 K=1,MSUB
      DO 15 KK=1,MSUB
      A=K-KK
      IF(A.LT.0) A=A+MSUB
      TSUB(I,K,KK)=A
   15 CONTINUE
    1 CONTINUE
      DO 40 II=1,30
      DO 41 JJ=1,31
      X1=(II-1)*1
      X2=(JJ-1)*30
      X2=MOD(X2,31)
      X3=(X1*MODU(2)+X2*MODU(1))
      X4=MOD(X3,PRO)
```

```
C
C     TABLE FOR INVERSE LOOK UP
C
      X5=MOD(X4,NMOD)
      X=IIND(X5+1)
      TINV(II,JJ)=X
C
C     TABLE FOR FINAL LOOK UP
C
      IF(X4.LE.MODS) GO TO 52
      X4=X4-PRO
      X4=X4+MMOD
      GO TO 53
   52 X4=MOD(X4,MMOD)
   53 TFIN(II,JJ)=X4
C
C     SUBTRACTION TABLE INDEXED
C
      B=IND(X4+1)
      TSUIN(1,II,JJ)=MOD(B,30)
      IF(X4.EQ.0)  TSUIN(1,II,JJ)=31
      TSUIN(2,II,JJ)=MOD(B,31)
      IF(X4.EQ.0) TSUIN(2,II,JJ)=31
   41 CONTINUE
   40 CONTINUE
C
C     CORRECTION FOR ZERO MULTIPLICATION
C
      DO 251 I=1,2
      DO 252 J=1,32
      TADD(I,32,J)=31
      TADMUL(I,32,J)=31


      TINV(32,J)=0
  252 CONTINUE
      DO 253 J=1,32
      TADD(I,J,32)=31
      TADMUL(I,J,32)=31
      TINV(J,32)=0
  253 CONTINUE
  251 CONTINUE
      RETURN
      END
```

```
C         **************************************************************
C         *                                                            *
C         *  SUBROUTINE TO GENERATE LOOK-UP TABLES FOR NTT             *
C         *    MODULUS=449                                             *
C         **************************************************************
      SUBROUTINE TABLES
      IMPLICIT INTEGER(A-H,O-Z)
      COMMON TASM(2,449),TADD(2,32,32),TADMUL(2,32,32),TSUB(2,31,31)
      COMMON TSUIN(2,31,31),TINV(32,32),TFIN(30,31),TF(2,64),TFI(2,64)
      COMMON POINT
      DIMENSION MODU(2),IND(449),IIND(449),ALPA(2,128)
      MMOD=449;PRIM=3;IND(1)=31;IND(2)=0;IIND(1)=1
      MODU(1)=30;MODU(2)=31
      PER=MMOD-2
      NMOD=MMOD-1
      PRO=MODU(1)*MODU(2)
      MODS=PRO-MMOD
      VAL=1
C
C     INDEX TABLE
C
      DO 21 K=1,PER
      VAL=VAL*PRIM
      VAL=MOD(VAL,MMOD)
      IND(VAL+1)=K
      IIND(K+1)=VAL
   21 CONTINUE
C
C     TABLE FOR POWER OF ALPA
C
      AA=0;BB=1;CC=0;DD=1
      ALPA(1,1)=0;ALPA(2,1)=0;ALPA(1,2)=0;ALPA(2,2)=0
      DO 31 NN=2,127
      RE=AA*CC+391*BB*DD
      IMAG=AA*DD+CC*BB
      RE=MOD(RE,MMOD)
      IMAG=MOD(IMAG,MMOD)
      IF(RE.NE.0) TEM=RE
      IF(IMAG.NE.0) TEM=IMAG
      A=IND(TEM+1)
      ALPA(1,NN+1)=MOD(A,30)
      ALPA(2,NN+1)=MOD(A,31)
      AA=RE
      BB=IMAG
   31 CONTINUE
C
C     TABLE FOR TWIDDLE FACTORS
C
      DO 34 II=1,64
      DO 35 I=1,2
      TF(I,II)=ALPA(I,II)
C
C     TABLE FOR INVERSE TWIDDLE FACTORS
C
      INK=II-1
      IF(INK.NE.0) INK=128-INK
      TFI(I,II)=ALPA(I,INK+1)
   35 CONTINUE
   34 CONTINUE
      DO 1 I=1,2
      MSUB=MODU(I)
```

```
C
C     SUB-MODULI RESIDUE TABLE
C
      DO 2 N=1,MMOD




      A=N-1
      TRSM(I,N)=MOD(A,MSUB)
    2 CONTINUE
C
C     ADDITION TABLE
C
      DO 11 N=1,MSUB
      DO 11 NN=1,MSUB
      A=N+NN-2
      TADD(I,N,NN)=MOD(A,MSUB)
C
C     TABLE WITH MULTIPLIER
C
      B=N+NN-2+7
      TADMUL(I,N,NN)=MOD(B,MSUB)
   11 CONTINUE
C
C     SUBTRACTION TABLE
C
      DO 15 K=1,MSUB
      DO 15 KK=1,MSUB
      A=K-KK
      IF(A.LT.0) A=A+MSUB
      TSUB(I,K,KK)=A
   15 CONTINUE
    1 CONTINUE
      DO 40 II=1,30
      DO 41 JJ=1,31
      X1=(II-1)*1
      X2=(JJ-1)*30
      X2=MOD(X2,31)
      X3=(X1*MODU(2)+X2*MODU(1))
      X4=MOD(X3,PRO)
C
C     TABLE FOR INVERSE LOOK UP
C
      X5=MOD(X4,NMOD)
      X=IIND(X5+1)
      TINV(II,JJ)=X
C
C     TABLE FOR FINAL LOOK UP
C
      X6=MOD(X4,MMOD)
      TFIN(II,JJ)=X6
C
C     SUBTRACTION TABLE INDEXED
C
      IF(X4.LE.MODS) GO TO 55
      X4=X4-PRO
      X4=X4+MMOD
      GO TO 56
   55 X4=MOD(X4,MMOD)
   56 B=IND(X4+1)
      TSUIN(1,II,JJ)=MOD(B,30)
      IF(X4.EQ.0)   TSUIN(1,II,JJ)=31
      TSUIN(2,II,JJ)=MOD(B,31)
      IF(X4.EQ.0) TSUIN(2,II,JJ)=31
```

```
   41 CONTINUE
   40 CONTINUE
C
C     CORRECTION FOR ZERO MULTIPLICATION
C
      DO 251 I=1,2
      DO 252 J=1,32
      TADD(I,32,J)=31
      TADMUL(I,32,J)=31
      TINV(32,J)=0
  252 CONTINUE
      DO 253 J=1,32
      TADD(I,J,32)=31
      TADMUL(I,J,32)=31
      TINV(J,32)=0
  253 CONTINUE
  251 CONTINUE
      RETURN
      END
```

```
C         ***********************************************************************
C         *                                                                     *
C         *      PROGRAM TO SIMULATE THE HARDWARE DESIGN NTT STRUCTURE          *
C         *      MODULUS=191                                                    *
C         ***********************************************************************
      SUBROUTINE NTT(INV,INP1,INP2,STG,POST,OUT1,OUT2)
      IMPLICIT INTEGER(A-H,O-Z)
      COMMON TINV(2,32,32),TSUB(2,31,31),TFIN(30,31),TFI(2,64,2)
      COMMON TRSM(2,191),TADD(2,32,32),TSUIN(2,31,31),TF(2,64,2)
      DIMENSION INP1(2),INP2(2),OUT1(2),OUT2(2)
      COMMON POINT
      IF(POINT.NE.0) GO TO 200
      R1=R2=R3=R4=R5=R6=R7=R8=R9=R10=R11=R12=R13=R14=R15=R16=R17=R18=0
      R19=R20=R21=R22=R23=R24=R25=R26=R27=R28=R29=R30=R31=R32=R33=R34=0
      R35=R36=R37=R38=R39=R40=R41=R42=R43=R44=R45=R46=R47=R48=R49=R50=0
      R51=R52=R53=R54=R55=R56=R57=R58=R59=R60=R61=R62=R63=R64=R65=R66=0
C
C     LATCH THE OUTPUT
C
  200 OUT1(1)=R63
      OUT1(2)=R64
      OUT2(1)=R65
      OUT2(2)=R66
C
C      THE SEVENTH STAGE
C
      R63=TFIN(R55+1,R56+1)
      R64=TFIN(R57+1,R58+1)
      R65=TFIN(R59+1,R60+1)
      R66=TFIN(R61+1,R62+1)
C
C      THE SIXTH STAGE
C
      R55=R43
      R56=R44
      R57=R45
      R58=R46
      R59=TSUB(1,R47+1,R53+1)
      R60=TSUB(2,R48+1,R54+1)
      R61=TADD(1,R49+1,R51+1)
      R62=TADD(2,R50+1,R52+1)
C
C      THE FIFTH STAGE
C
      R43=R31
      R44=R32
      R45=R33
      R46=R34
      R47=TINV(1,R35+1,R36+1)
      R48=TINV(2,R35+1,R36+1)
      R49=TINV(1,R37+1,R38+1)
      R50=TINV(2,R37+1,R38+1)
      R51=TINV(1,R39+1,R40+1)
      R52=TINV(2,R39+1,R40+1)
      R53=TINV(1,R41+1,R42+1)
      R54=TINV(2,R41+1,R42+1)
C
C     THE FOURTH STAGE
C
```

```
      R31=R19
      R32=R20
      R33=R21
      R34=R22
      R35=TADD(1,R23+1,R27+1)
      R36=TADD(2,R24+1,R28+1)



      R37=TADD(1,R25+1,R27+1)
      R38=TADD(2,R26+1,R28+1)
      R39=TADD(1,R23+1,R29+1)
      R40=TADD(2,R24+1,R30+1)
      R41=TADD(1,R25+1,R29+1)
      R42=TADD(2,R26+1,R30+1)
C
C     THE THIRD STAGE
C
      R19=R10
      R20=R11
      R21=R12
      R22=R13
      R23=TSUIN(1,R14+1,R15+1)
      R24=TSUIN(2,R14+1,R15+1)
      R25=TSUIN(1,R16+1,R17+1)
      R26=TSUIN(2,R16+1,R17+1)
      IF(INV.NE.0) GO TO 300
      R27=TF(1,R18+1,1)
      R28=TF(2,R18+1,1)
      R29=TF(1,R18+1,2)
      R30=TF(2,R18+1,2)
      GO TO 400
  300 R27=TFI(1,R18+1,1)
      R28=TFI(2,R18+1,1)
      R29=TFI(1,R18+1,2)
      R30=TFI(2,R18+1,2)
C
C     THE SECOND STAGE
C
  400 R10=TADD(1,R1+1,R5+1)
      R11=TADD(2,R2+1,R6+1)
      R12=TADD(1,R3+1,R7+1)
      R13=TADD(2,R4+1,R8+1)
      R14=TSUB(1,R1+1,R5+1)
      R15=TSUB(2,R2+1,R6+1)
      R16=TSUB(1,R3+1,R7+1)
      R17=TSUB(2,R4+1,R8+1)
      R18=R9
C
C     THE FIRST STAGE
C
      T1=INP1(1)+1
      T2=INP1(2)+1
      T3=INP2(1)+1
      T4=INP2(2)+1
      R1=TRSM(1,T1)
      R2=TRSM(2,T1)
      R3=TRSM(1,T2)
      R4=TRSM(2,T2)
      R5=TRSM(1,T3)
```

```
      R7=TRSM(1,T4)
      R8=TRSM(2,T4)
C
C     GENERATE THE POWER OF ALPHA
C
      R9=POST/(2**STG)*(2**STG)
      POINT=POINT+1
      RETURN
      END
```

```
C        ***********************************************************************
C        *                                                                     *
C        *    PROGRAM TO SIMULATE THE HARDWARE DESIGN NTT STRUCTURE            *
C        *    MODULUS=193                                                      *
C        *                                                                     *
C        ***********************************************************************
      SUBROUTINE NTT(INV, INP1, INP2, STG, POST, OUT1, OUT2)
      IMPLICIT INTEGER(A-H, O-Z)
      COMMON TRSM(2,193), TADD(2,32,32), TADMUL(2,32,32), TSUB(2,31,31)
      COMMON TSUIN(2,31,31), TINV(32,32), TFIN(30,31), TF(2,64), TFI(2,64)
      COMMON POINT
      DIMENSION INP1(2), INP2(2), OUT1(2), OUT2(2)
      IF(POINT.NE.0) GO TO 200
      R1=R2=R3=R4=R5=R6=R7=R8=R9=R10=R11=R12=R13=R14=R15=R16=R17=R18=0
      R19=R20=R21=R22=R23=R24=R25=R26=R27=R28=R29=R30=R31=R32=R33=R34=0
      R35=R36=R37=R38=R39=R40=0
C
C     CHEKS ARE THE CONTROLS FOR EVEN/ODD POWER OF ALPHA
C
      CHEK1=CHEK2=CHEK3=CHEK4=CHEK5=0
C
C     LATCH THE OUTPUT
C
  200 OUT1(1)=R37
      OUT1(2)=R38
      OUT2(1)=R39
      OUT2(2)=R40
C
C     THE FIFTH STAGE
C
      R37=R33
      R38=R34
      IF(CHEK5.NE.0) GO TO 300
      R39=R35
      R40=R36
      GO TO 301
  300 R39=R36
      R40=R35
  301 R33=R27
      R34=R28
      R36=TINV(R31+1,R32+1)
      CHEK5=CHEK4
C
C     THE FOURTH STAGE
C
      R27=R19
      R35=TINV(R29+1,R30+1)
      R28=R20
      R29=TADD(1,R21+1,R25+1)
      R30=TADD(2,R22+1,R26+1)
      IF(CHEK3.NE.0) GO TO 302
      R31=TADD(1,R23+1,R25+1)
      R32=TADD(2,R24+1,R26+1)
      GO TO 303
  302 R31=TADMUL(1,R23+1,R25+1)
```

```
C
C      THE THIRD STAGE
C
  303  CHEK4=CHEK3
       R19=TFIN(R10+1,R11+1)
       R20=TFIN(R12+1,R13+1)
       R21=TSUIN(1,R14+1,R15+1)
       R22=TSUIN(2,R14+1,R15+1)
       R23=TSUIN(1,R16+1,R17+1)
       R24=TSUIN(2,R16+1,R17+1)
       IF(INV.NE.0) GO TO 400
       R25=TF(1,R18+1)
       R26=TF(2,R18+1)
       GO TO 401
  400  R25=TFI(1,R18+1)
       R26=TFI(2,R18+1)
C
C      THE SECOND STAGE
C
  401  CHEK3=CHEK2
       R10=TADD(1,R1+1,R5+1)
       R11=TADD(2,R2+1,R6+1)
       R12=TADD(1,R3+1,R7+1)
       R13=TADD(2,R4+1,R8+1)
       R14=TSUB(1,R1+1,R5+1)
       R15=TSUB(2,R2+1,R6+1)
       R16=TSUB(1,R3+1,R7+1)
       R17=TSUB(2,R4+1,R8+1)
       R18=R9
       CHEK2=CHEK1
C
C      THE FIRST STAGE
C
       A=INP1(1)
       B=INP1(2)
       C=INP2(1)
       D=INP2(2)
       R1=TRSM(1,A+1)
       R2=TRSM(2,A+1)
       R3=TRSM(1,B+1)
       R4=TRSM(2,B+1)
       R5=TRSM(1,C+1)
       R6=TRSM(2,C+1)
       R7=TRSM(1,D+1)
       R8=TRSM(2,D+1)
C
C      GENERATE THE POWER OF ALPHA
C
       R9=POST/(2**STG)*(2**STG)
       CHEK1=MOD(R9,2)
       POINT=POINT+1
       RETURN
       END
```

```
C         ************************************************************************
C         *                                                                      *
C         *      PROGRAM TO SIMULATE THE HARDWARE DESIGN NTT STRUCTURE          *
C         *      MODULUS=449                                                     *
C         ************************************************************************
      SUBROUTINE NTT(INV, INP1, INP2, STG, POST, OUT1, OUT2)
      IMPLICIT INTEGER(A-H, O-Z)
      COMMON TRSM(2, 449), TADD(2, 32, 32), TADMUL(2, 32, 32), TSUB(2, 31, 31)
      COMMON TSUIN(2, 31, 31), TINV(32, 32), TFIN(30, 31), TF(2, 64), TFI(2, 64)
      COMMON POINT
      DIMENSION INP1(2), INP2(2), OUT1(2), OUT2(2)
      IF(POINT.NE.0) GO TO 200
      R1=R2=R3=R4=R5=R6=R7=R8=R9=R10=R11=R12=R13=R14=R15=R16=R17=R18=0
      R19=R20=R21=R22=R23=R24=R25=R26=R27=R28=R29=R30=R31=R32=R33=R34=0
      R35=R36=R37=R38=R39=R40=0
C
C     CHEKS ARE THE CONTROLS FOR EVEN/ODD POWER OF ALPHA
C
      CHEK1=CHEK2=CHEK3=CHEK4=CHEK5=0
C
C    LATCH THE OUTPUT
C
  200 OUT1(1)=R37
      OUT1(2)=R38
      OUT2(1)=R39
      OUT2(2)=R40
C
C     THE FIFTH STAGE
C
      R37=R33
      R38=R34
      IF(CHEK5.NE.0) GO TO 300
      R39=R35
      R40=R36
      GO TO 301
  300 R39=R36
      R40=R35
  301 R33=R27
      R34=R28
      R35=TINV(R29+1, R30+1)
      R36=TINV(R31+1, R32+1)
      CHEK5=CHEK4
C
C     THE FOURTH STAGE
C
      R27=R19
      R28=R20
      R29=TADD(1, R21+1, R25+1)
      R30=TADD(2, R22+1, R26+1)
      IF(CHEK3.NE.0) GO TO 302
      R31=TADD(1, R23+1, R25+1)
      R32=TADD(2, R24+1, R26+1)
      GO TO 303
  302 R31=TADMUL(1, R23+1, R25+1)
      R32=TADMUL(2, R24+1, R26+1)
```

```
C
C     THE THIRD STAGE
C
  303 CHEK4=CHEK3
      R19=TFIN(R10+1,R11+1)
      R20=TFIN(R12+1,R13+1)
      R21=TSUIN(1,R14+1,R15+1)
      R22=TSUIN(2,R14+1,R15+1)
      R23=TSUIN(1,R16+1,R17+1)
      R24=TSUIN(2,R16+1,R17+1)




      IF(INV.NE.0) GO TO 400
      R25=TF(1,R18+1)
      R26=TF(2,R18+1)
      GO TO 401
  400 R25=TFI(1,R18+1)
      R26=TFI(2,R18+1)
C
C     THE SECOND STAGE
C
  401 CHEK3=CHEK2
      R10=TADD(1,R1+1,R5+1)
      R11=TADD(2,R2+1,R6+1)
      R12=TADD(1,R3+1,R7+1)
      R13=TADD(2,R4+1,R8+1)
      R14=TSUB(1,R1+1,R5+1)
      R15=TSUB(2,R2+1,R6+1)
      R16=TSUB(1,R3+1,R7+1)
      R17=TSUB(2,R4+1,R8+1)
      R18=R9
      CHEK2=CHEK1
C
C     THE FIRST STAGE
C
      A=INP1(1)
      B=INP1(2)
      C=INP2(1)
      D=INP2(2)
      R1=TRSM(1,A+1)
      R2=TRSM(2,A+1)
      R3=TRSM(1,B+1)
      R4=TRSM(2,B+1)
      R5=TRSM(1,C+1)
      R6=TRSM(2,C+1)
      R7=TRSM(1,D+1)
      R8=TRSM(2,D+1)
C
C     GENERATE THE POWER OF ALPHA
C
      R9=POST/(2**STG)*(2**STG)
      CHEK1=MOD(R9,2)
      POINT=POINT+1
      RETURN
      END
```

APPENDIX B

PROGRAMS TO GENERATE TABLES FOR

EPROMS ON INTEL 220

SOURCE STATEMENT

```
;SUBROUTINE TO REDUCE A 16 BIT NO. MODULO 30
;NO. TO BE REDUCED IS IN MEM. LOCATION 8400-8401
H
;RESULT IN 8400H
        PUBLIC  RED30
        CSEG
RED30:  PUSH    D
        PUSH    H
        LXI     H,8400H
        MOV     E,M
        INX     H
        MOV     D,M
CHEK:   MOV     A,D
        ORA     A
        JNZ     SUBT1
        MOV     A,E
        CPI     30
        JC      STOR
LOOP:   SUI     30
        CPI     30
        JNC     LOOP
STOR:   LXI     H,8400H
        MOV     M,A
        POP     H
        POP     D
        RET
SUBT1:  MOV     A,E
        SUI     30
        MOV     E,A
        JNC     CHEK
SUBT2:  MOV     A,D
        SUI     01
        MOV     D,A
        JMP     CHEK
        END
```

```
        SOURCE STATEMENT

;SUBROUTINE TO REDUCE A 10 BIT NO. MODULO 31
;NO. TO BE REDUCED IS IN MEM. LOCATION 8400-8401
H.
;RESULT IN 8400H.
        PUBLIC RED31
        CSEG
RED31:  PUSH    D
        PUSH    H
        LXI     H,8400H
        MOV     E,M
        INX     H
        MOV     D,M
CHEK:   MOV     A,D
        ORA     A
        JNZ     SUBT1
        MOV     A,E
        CPI     31
        JC      STOR
LOOP:   SUI     31
        CPI     31
        JNC     LOOP
STOR:   LXI     H,8400H
        MOV     M,A
        POP     H
        POP     D
        RET
SUBT1:  MOV     A,E
        SUI     31
        MOV     E,A
        JNC     CHEK
SUBT2:  MOV     A,D
        SBI     01
        MOV     D,A
        JMP     CHEK
        END
```

```
LOC  OBJ            LINE     SOURCE STATEMENT

                             ;SUBROUTINE TO REDUCE A 10 BIT NO. MODULO 192
                             ;NO. TO BE REDUCED IS IN MEM. LOCATION 8400-8401H
                             ;RESULT IN MEM. LOCATION 8400H
                                     PUBLIC  RED192
                                     CSEG
0000 D5                      RED192: PUSH    D
0001 E5                              PUSH    H
0002 210084                          LXI     H,8400H
0005 5E                              MOV     E,M
0006 23                              INX     H
0007 56                              MOV     D,M
0008 7A                      LOOP:   MOV     A,D
0009 B7                              ORA     A
000A C21300    C                     JNZ     DSUB
000D 7B                              MOV     A,E
000E FEC0                            CPI     192
0010 DA1E00    C                     JC      STOR
0013 2140FF                  DSUB:   LXI     H,-192
0016 19                              DAD     D
0017 EB                              XCHG
0018 C30800    C                     JMP     LOOP
001B 210084                  STOR:   LXI     H,8400H
001E 77                              MOV     M,A
001F E1                              POP     H
0020 D1                              POP     D
0021 C9                              RET
                                     END
```

```
LOC  OBJ            LINE     SOURCE STATEMENT

                             ;SUBROUTINE TO REDUCE A 10 BIT NO. MODULO 193
                             ;NO. TO BE REDUCED IS IN MEM. LOCATION 8400-8401H
                             ;RESULT IN MEM. LOCATION 8400H
                                     PUBLIC  RED193
                                     CSEG
0000 D5                      RED193: PUSH    D
0001 E5                              PUSH    H
0002 210084                          LXI     H,8400H
0005 5E                              MOV     E,M
0006 23                              INX     H
0007 56                              MOV     D,M
0008 7A                      LOOP:   MOV     A,D
0009 B7                              ORA     A
000A C21300    C                     JNZ     DSUB
000D 7B                              MOV     A,E
000E FEC1                            CPI     193
0010 DA1E00    C                     JC      STOR
0013 213FFF                  DSUB:   LXI     H,-193
0016 19                              DAD     D
0017 EB                              XCHG
0018 C30800    C                     JMP     LOOP
001B 210084                  STOR:   LXI     H,8400H
001E 77                              MOV     M,A
001F E1                              POP     H
0020 D1                              POP     D
0021 C9                              RET
                                     END
```

```
LOC   OBJ         LINE        SOURCE STATEMENT

                  ;SUBROUTINE TOREDUCE A 11 BIT NO.MODULO 930
                  ;THE MAX. NO. IS 1860
                  ;NO.REDUCED TO BE FOUND IN 8400-8401H
                  ;RESULT IN THE SAME LOCATION
                            PUBLIC    RED930
                            CSEG
0000 D5           RED930:   PUSH      D
0001 E5                     PUSH      H
0002 210084                 LXI       H,8400H
0005 5E                     MOV       E,M
0006 23                     INX       H
0007 56                     MOV       D,M
0008 7A           STAR:     MOV       A,D
0009 FE03                   CPI       03        ;CHECK HIGHER BYTE.
000B DA1C00   C             JC        STOR      ;NO    ACTION
000E C21700   C             JNZ       SUBT ;IF HIGHER BYTE GREATER.SUB
T
0011 7B                     MOV       A,E
0012 FEA2                   CPI       162       ;SUM OF LOWER BYTE OF 93
0
0014 DA1C00   C             JC        STOR
0017 215EFC       SUBT:     LXI       H,-930
001A 19                     DAD       D
001B EB                     XCHG
001C 210084       STOR:     LXI       H,8400H
001F 73                     MOV       M,E
0020 23                     INX       H
0021 72                     MOV       M,D
0022 E1                     POP       H
0023 D1                     POP       D
0024 C9                     RET
                            END
```

```
LOC  OBJ          SOURCE STATEMENT

                  ;SUBROUTINE TO MULTIPLY TWO 8 BIT NO.
                  ;NO.TO BE MULTIPLIED IN REG. D & E
                  ;RESULT IN MEM. LOCATION 8400-8401H
                          PUBLIC  MULTIP
                          CSEG
0000 C5           MULTIP: PUSH    B
0001 E5                   PUSH    H
0002 010000               LXI     B,0 ;TEMP. RESULT IN REG. PAIR B
0005 2E08                 MVI     L,8
0007 7A           NXTBIT: MOV     A,D
0008 1F                   RAR
0009 57                   MOV     D,A
000A D21000   C           JNC     NOADD
000D 78                   MOV     A,B
000E 83                   ADD     E
000F 47                   MOV     B,A
0010 78           NOADD:  MOV     A,B
0011 1F                   RAR
0012 47                   MOV     B,A
0013 79                   MOV     A,C
0014 1F                   RAR
0015 4F                   MOV     C,A
0016 2D                   DCR     L
0017 C20700   C           JNZ     NXTBIT
001A 210084               LXI     H,8400H
001D 71                   MOV     M,C
001E 23                   INX     H
001F 70                   MOV     M,B
0020 E1                   POP     H
0021 C1                   POP     B
0022 C9                   RET
                          END
```

```
LOC  OBJ        SOURCE STATEMENT

                ;SUBROUTINE TO ADD TWO 16 BIT NOS.
                ;NOS.STORED IN 8410-8413H
                        PUBLIC  ADSR
                        CSEG
0000 C5         ADSR:   PUSH    B
0001 D5                 PUSH    D
0002 E5                 PUSH    H
0003 211084             LXI     H,8410H
0006 4E                 MOV     C,M
0007 23                 INX     H
0008 46                 MOV     B,M
0009 23                 INX     H
000A 5E                 MOV     E,M
000B 23                 INX     H
000C 56                 MOV     D,M
000D 79                 MOV     A,C
000E 83                 ADD     E
000F 4F                 MOV     C,A
0010 78                 MOV     A,B
0011 8A                 ADC     D
0012 47                 MOV     B,A
0013 210084             LXI     H,8400H
0016 71                 MOV     M,C
0017 23                 INX     H
0018 70                 MOV     M,B
0019 E1                 POP     H
001A D1                 POP     D
001B C1                 POP     B
001C C9                 RET
                        END
```

```
LOC   OBJ          LINE    SOURCE STATEMENT

                           ;SUBROUTINE TO COMPARE TWO 16 BIT NO. WITH 738
                           ;NOS. STORED IN MEM. LOCATION 8400-8401H

                                   PUBLIC  COM738
                                   CSEG
0000  D5                   COM738: PUSH    D
0001  E5                           PUSH    H
0002  210084                       LXI     H,8400H
0005  5E                           MOV     E,M
0006  23                           INX     H
0007  56                           MOV     D,M
0008  7A                           MOV     A,D
0009  E602                         ANI     02
000B  CA1F00    C                  JZ      LESS
000E  7A                           MOV     A,D
000F  E601                         ANI     01
0011  C21A00    C                  JNZ     GREAT
0014  7B                           MOV     A,E
0015  FEE2                         CPI     226
0017  DA1F00    C                  JC      LESS
001A  3E01                 GREAT:  MVI     A,1
001C  C32100    C                  JMP     L1
001F  3E00                 LESS:   MVI     A,0
0021  210284               L1:     LXI     H,8402H;STATUS IN MEM. 8402H
0024  77                           MOV     M,A
0025  E1                           POP     H
0026  D1                           POP     D
0027  C9                           RET
                                   END
```

```
LOC   OBJ          LINE    SOURCE STATEMENT

                           ;SUBROUTINE TO CONVERT A NEGATIVE NO.MODULO 193
                           ;THE NO. TO BE CONVERTED IS IN MEM. LOC. 8400-84
                           01H
                                   PUBLIC  NEGCON
                                   CSEG
0000  D5                   NEGCON: PUSH    D
0001  E5                           PUSH    H
0002  210084                       LXI     H,8400H
0005  7E                           MOV     A,M
0006  D6A2                         SUI     162;THE SUM LOWER BITS OF 930
0008  C6C1                         ADI     193
000A  77                           MOV     M,A
000B  E1                           POP     H
000C  D1                           POP     D
000D  C9                           RET
                                   END
```

```
                    ;PROGRAM TO GENERATE THE SUB-MODULO 31 RESIDUE TABLE
                    ;TRSM31
        CSEG
START:  LXI     H,7400H
        MVI     B,193   ;START THE COUNTER
        MVI     C,0
LOOP:   MOV     A,C
        CPI     31
        JC      STORE
SUBT:   SUI     31
STORE:  MOV     M,A
        MOV     C,A
        INX     H
        DCR     B
        JNZ     LOOP
        JMP     0F855H
        END     START
```

```
                    ;PROGRAM TO GENERATE THE SUB-MODULO 30 RESIDUE TABLE
                    ;TRSM30
        CSEG
START:  LXI     H,7400H
        MVI     B,193   ;START THE COUNTER
        MVI     C,0
LOOP:   MOV     A,C
        CPI     30
        JC      STORE
SUBT:   SUI     30
STORE:  MOV     M,A
        MOV     C,A
        INX     H
        DCR     B
        JNZ     LOOP
        JMP     0F855H
        END     START
```

```
ASM80 TADD30.SRC PAGEWIDTH(32)

ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0          MODULE   PAGE    1

  LOC  OBJ           LINE        SOURCE STATEMENT

                                          ;ADDITION TABLE FOR MODULO 30
                                   CSEG
 0000 210074              START:   LXI    H,7400H
 0003 0E00                         MVI    C,0
 0005 161E                         MVI    D,30      ;COUNTER FOR M2
 0007 0600                L1:      MVI    B,0
 0009 1E1E                         MVI    E,30      ;COUNTER FOR M1
 000B AF                  LOOP:    XRA    A
 000C 78                           MOV    A,B
 000D 81                           ADD    C
 000E FE1E                         CPI    30
 0010 DA1500    C                  JC     STOR
 0013 D61E                         SUI    30
 0015 77                  STOR:    MOV    M,A
 0016 23                           INX    H
 0017 04                           INR    B
 0018 1D                           DCR    E
 0019 C20B00    C                  JNZ    LOOP
 001C 23                           INX    H
 001D 23                           INX    H
 001E 0C                           INR    C
 001F 15                           DCR    D
 0020 C20700    C                  JNZ    L1
 0023 C355F8                       JMP    0F855H
 0000           C                  END    START
```

```
ASM80 TADD31.SRC PAGEWIDTH(32)


ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0          MODULE   PAGE    1

  LOC  OBJ          LINE        SOURCE STATEMENT

                       1                  ;ADDITION TABLE FOR MODULO 31
                       2          CSEG
  0000 210074          3 START:   LXI     H,7400H
  0003 0E00            4          MVI     C,0
  0005 161F            5          MVI     D,31       ;COUNTER FOR M2
  0007 0600            6 L1:      MVI     B,0
  0009 1E1F            7          MVI     E,31       ;COUNTER FOR M1
  000B AF              8 LOOP:    XRA     A
  000C 78              9          MOV     A,B
  000D 81             10          ADD     C
  000E FE1F           11          CPI     31
  0010 DA1500    C    12          JC      STOR
  0013 D61F           13          SUI     31
  0015 77             14 STOR:    MOV     M,A
  0016 23             15          INX     H
  0017 04             16          INR     B
  0018 1D             17          DCR     E
  0019 C20B00    C    18          JNZ     LOOP
  001C 23             19          INX     H
  001D 0C             20          INR     C
  001E 15             21          DCR     D
  001F C20700    C    22          JNZ     L1
  0022 C355F8         23          JMP     0F855H
  0000           C    24          END     START
```

```
ASM80 ADML30.SRC PAGEWIDTH(42)

ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0          MODULE   PAGE   1

  LOC  OBJ               SOURCE STATEMENT

                                ;ADDITION TABLE FOR MODULO 30
                             CSEG
 0000 210074        START:   LXI   H,7400H
 0003 0E00                   MVI   C,0
 0005 161E                   MVI   D,30     ;COUNTER FOR M2
 0007 0600          L1:      MVI   B,0
 0009 1E1E                   MVI   E,30     ;COUNTER FOR M1
 000B AF            LOOP:    XRA   A
 000C 78                     MOV   A,B
 000D 81                     ADD   C
 000E C603                   ADI   03       ;ADD INDEX OF 125(3)
 0010 FE1E                   CPI   30
 0012 DA1C00    C            JC    STOR
 0015 D61E          SUBT:    SUI   30
 0017 FE1E                   CPI   30
 0019 D21500    C            JNC   SUBT
 001C 77            STOR:    MOV   M,A
 001D 23                     INX   H
 001E 04                     INR   B
 001F 1D                     DCR   E
 0020 C20B00    C            JNZ   LOOP
 0023 23                     INX   H
 0024 23                     INX   H
 0025 0C                     INR   C
 0026 15                     DCR   D
 0027 C20700    C            JNZ   L1
 002A C355F8                 JMP   0F855H
 0000           C            END   START
```

SOURCE STATEMENT

```
                        ;ADDITION TABLE FOR MODULO 31
        CSEG
START:  LXI     H,7400H
        MVI     C,0
        MVI     D,31        ;COUNTER FOR M2
L1:     MVI     B,0
        MVI     E,31        ;COUNTER FOR M1
LOOP:   XRA     A
        MOV     A,B
        ADD     C
        ADI     03          ;ADD INDEX OF 125(3)
        CPI     31
        JC      STOR
SUBT:   SUI     31
        CPI     31
        JNC     SUBT
STOR:   MOV     M,A
        INX     H
        INR     B
        DCR     E
        JNZ     LOOP
        INX     H
        INR     C
        DCR     D
        JNZ     L1
        JMP     0F855H
        END     START
```

```
ASM80 TSUB30.SRC


ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0          MODULE   PAGE    1

  LOC  OBJ         LINE        SOURCE STATEMENT

                      1          CSEG
  0000 210074         2 START:   LXI     H,7400H
  0003 0E00           3          MVI     C,0
  0005 161E           4          MVI     D,30       ;COUNTER FOR M2
  0007 0600           5 L1:      MVI     B,0
  0009 1E1E           6          MVI     E,30       ;COUNTER FOR M1
  000B AF             7 LOOP:    XRA     A
  000C 78             8          MOV     A,B
  000D 91             9          SUB     C
  000E D21300    C   10          JNC     STOR
  0011 C61E          11          ADI     30
  0013 77            12 STOR:    MOV     M,A
  0014 23            13          INX     H
  0015 04            14          INR     B
  0016 1D            15          DCR     E
  0017 C20B00    C   16          JNZ     LOOP
  001A 23            17          INX     H
  001B 23            18          INX     H
  001C 0C            19          INR     C
  001D 15            20          DCR     D
  001E C20700    C   21          JNZ     L1
  0021 C355F8        22          JMP     0F855H
  0000           C   23          END     START
```

```
ASM80 TSUB31.SRC


ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0          MODULE   PAGE    1

  LOC  OBJ         LINE        SOURCE STATEMENT

                      1        CSEG
  0000 210074         2 START: LXI     H,7400H
  0003 0E00           3        MVI     C,0
  0005 161F           4        MVI     D,31      ;COUNTER FOR M2
  0007 0600           5 L1:    MVI     B,0
  0009 1E1F           6        MVI     E,31      ;COUNTER FOR M1
  000B AF             7 LOOP:  XRA     A
  000C 78             8        MOV     A,B.
  000D 91             9        SUB     C
  000E D21300    C   10        JNC     STOR
  0011 C61F          11        ADI     31
  0013 77            12 STOR:  MOV     M,A
  0014 23            13        INX     H
  0015 04            14        INR     B
  0016 1D            15        DCR     E
  0017 C20B00    C   16        JNZ     LOOP
  001A 23            17        INX     H
  001B 0C            18        INR     C
  001C 15            19        DCR     D
  001D C20700    C   20        JNZ     L1
  0020 C355F8        21        JMP     0F855H
  0000           C   22        END     START
```

```
                                ;GENERATE SUBTRACTION INDEX TABLE
                                ;SUIN30
         CSEG
START:   LXI      SP,STACK
         LXI      H,7400H
         LXI      B,1024    ;COUNTER
INIT1:   MOV      E,M
         PUSH     H
         MOV      A,E
         ORA      A
         JZ       CORRECT ;IF ENTRY IS 0.NO INDEX EXISTS
         MOV      A,E
         CPI      0FFH      ;IF FF.NO ACTION
         JNC      CORRECT
         MOV      H,78      ;INDEX TABLE STORED IN 7800H
         MOV      L,E
         MOV      A,M       ;BRING THE INDEX
         CPI      30        ;REDUCE THE INDEX IN SUB-MODULO 30
         JC       STOR
SUBT:    SUI      30
         CPI      30
         JNC      SUBT
         JMP      STOR
CORRECT:MVI       A,0FFH    ;STORE FF FOR INDEX OF ZERO
STOR:    POP      H
         MOV      M,A
         INX      H
         DCX      B
         MOV      A,B
         ORA      A
         JNZ      INT1
         JMP      0F855H
         END      START
```

```
                                ;GENERATE SUBTRACTION INDEX TABLE
                                ;SUIN31
        CSEG
START:  LXI     SP,STACK
        LXI     H,7400H
        LXI     B,1024   ;COUNTER
INIT1:  MOV     E,M
        PUSH    H
        MOV     A,E
        ORA     A
        JZ      CORRECT ;IF ENTRY IS 0.NO INDEX EXISTS
        MOV     A,E
        CPI     0FFH     ;IF FF.NO ACTION
        JNC     CORRECT
        MOV     H,78     ;INDEX TABLE STORED IN 7800H
        MOV     L,E
        MOV     A,M      ;BRING THE INDEX
        CPI     31       ;REDUCE THE INDEX IN SUB-MODULO 31
        JC      STOR
SUBT:   SUI     31
        CPI     31
        JNC     SUBT
        JMP     STOR
CORRECT:MVI     A,0FFH   ;STORE FF FOR INDEX OF ZERO
STOR:   POP     H
        MOV     M,A
        INX     H
        DCX     B
        MOV     A,B
        ORA     A
        JNZ     INT1
        JMP     0F855H
        END     START
```

TFIN

```
LOC  OBJ          LINE        SOURCE STATEMENT

                     1              EXTRN   RED930,RED193,RED30,RED31
                     2              EXTRN   MULTIP,ADSR,NEGCON,COM738
                     3              CSEG
0000 310000   S      4 START:       LXI     SP,STACK
0003 210074          5              LXI     H,7400H
0006 E5              6              PUSH    H
0007 0E00            7              MVI     C,0      ;COUNTER FOR R2
0009 0600            8 L1:          MVI     B,0      ;COUNTER FOR R1
000B 59              9              MOV     E,C
000C 161E           10              MVI     D,30D
000E CD0000   E     11              CALL    MULTIP   ;RESULT NOW IN 8400,8401
0011 CD0000   E     12              CALL    RED31    ;REDUCE MOD 31
0014 161E           13              MVI     D,30D    ;MULTIPLY BY M2 BAR
0016 210084         14              LXI     H,8400H
0019 5E             15              MOV     E,M
001A CD0000   E     16              CALL    MULTIP
001D 5E             17              MOV     E,M
001E 23             18              INX     H
001F 56             19              MOV     D,M
0020 211084         20              LXI     H,8410H ;STOR THE RESULT IN 8410
                       H
0023 73             21              MOV     M,E
0024 23             22              INX     H
0025 72             23              MOV     M,D
0026 58             24 L2:          MOV     E,B
0027 161F           25              MVI     D,31D
0029 CD0000   E     26              CALL    MULTIP   ;MULTIPLY BY M1 BAR
002C 210084         27              LXI     H,8400H
002F 5E             28              MOV     E,M
0030 23             29              INX     H
0031 56             30              MOV     D,M
0032 211284         31              LXI     H,8412H
0035 73             32              MOV     M,E
0036 23             33              INX     H
0037 72             34              MOV     M,D
0038 CD0000   E     35              CALL    ADSR     ;ADD TWO 16 BIT NUMBERS
003B CD0000   E     36              CALL    RED930   ;REDUCE MOD M1*M2
003E CD0000   E     37              CALL    COM738   ;COMPARE THE N[illegible]
[illegible] 210284  38              LXI     H,8402H
0044 7E             39              MOV     A,M      ;CHECK THE STATUS
0045 B7             40              ORA     A
0046 C25300   C     41              JNZ     CON      ;STATUS 1, GO TO CONVERS
                       ION
0049 CD0000   E     42              CALL    RED193   ;OTHERWISE REDUCE MOD 19
                       3
004C 210084         43              LXI     H,8400H
004F 5E             44              MOV     E,M
0050 C35A00   C     45              JMP     STOR
0053 CD0000   E     46 CON:         CALL    NEGCON
0056 210084         47              LXI     H,8400H
0059 5E             48              MOV     E,M
005A E1             49 STOR:        POP     H
005B 73             50              MOV     M,E
```

```
LOC   OBJ              LINE        SOURCE STATEMENT

005C  23                 51         INX     H
005D  E5                 52         PUSH    H
005E  04                 53         INR     B
005F  78                 54         MOV     A,B
0060  FE1E               55         CPI     30D
0062  C22600      C      56         JNZ     L2
0065  E1                 57         POP     H
0066  23                 58         INX     H
0067  23                 59         INX     H
0068  E5                 60         PUSH    H
0069  0C                 61         INR     C
006A  79                 62         MOV     A,C
006B  FE1F               63         CPI     31D
006D  C20900      C      64         JNZ     L1
0070  C355F8             65         JMP     0F855H
0000              C      66         END     START
```

```
ASM80 T.INV.SRC PAGEWIDTH(42)


ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0          MODULE   PAGE    1

 LOC  OBJ           LINE        SOURCE STATEMENT

                       1            EXTRN   RED930,RED192,RED30,RED31
                       2            EXTRN   MULTIP,ADSR
                       3            CSEG
 0000 310000    S      4 START:     LXI     SP,STACK
 0003 210074           5            LXI     H,7400H
 0006 E5               6            PUSH    H
 0007 0E00             7            MVI     C,0       ;COUNTER FOR R2
 0009 0600             8 L1:        MVI     B,0       ;COUNTER FOR R1
 000B 59               9            MOV     E,C
 000C 161E            10            MVI     D,30D
 000E CD0000    E     11            CALL    MULTIP    ;RESULT NOW IN 8400,8401
 0011 CD0000    E     12            CALL    RED31     ;REDUCE MOD 31
 0014 161E            13            MVI     D,30D     ;MULTIPLY BY M2 BAR
 0016 210084          14            LXI     H,8400H
 0019 5E              15            MOV     E,M
 001A CD0000    E     16            CALL    MULTIP
 001D 5E              17            MOV     E,M
 001E 23              18            INX     H
 001F 56              19            MOV     D,M
 0020 211084          20            LXI     H,8410H ;STOR THE RESULT IN 8410
                         H
 0023 73              21            MOV     M,E
 0024 23              22            INX     H
 0025 72              23            MOV     M,D
 0026 58              24 L2:        MOV     E,B
 0027 161F            25            MVI     D,31D
 0029 CD0000    E     26            CALL    MULTIP    ;MULTIPLY BY M1 BAR
 002C 210084          27            [illegible]
                      [illegible]           E,M
 0030 23              29            INX     H
 0031 56              30            MOV     D,M
 0032 211284          31            LXI     H,8412H
 0035 73              32            MOV     M,E
 0036 23              33            INX     H
 0037 72              34            MOV     M,D
 0038 CD0000    E     35            CALL    ADSR      ;ADD TWO 16 BIT NUMBERS
 003B CD0000    E     36            CALL    RED930    ;REDUCE MOD M1*M2
 003E CD0000    E     37            CALL    RED192             ;REDUCE MODULO (
                         M-1)
 0041 210084          38            LXI     H,8400H
 0044 5E              39            MOV     E,M
 0045 2679            40 STOR:      MVI     H,79H     ;LOOK UP TABLE FOR INVER
                         SE INDEX
```

```
0047 6B              41        MOV    L,E
0048 5E              42        MOV    E,M
0049 E1              43        POP    H
004A 73              44        MOV    M,E
004B 23              45        INX    H
004C E5              46        PUSH   H
004D 04              47        INR    B
004E 78              48        MOV    A,B
004F FE1E            49        CPI    30D
0051 C22600    C     50        JNZ    L2
0054 E1              51        POP    H

ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0          MODULE   PAGE    2

 LOC  OBJ            LINE        SOURCE STATEMENT

0055 23              52        INX    H
0056 23              53        INX    H
0057 E5              54        PUSH   H
0058 0C              55        INR    C
0059 79              56        MOV    A,C
005A FE1F            57        CPI    31D
005C C20900    C     58        JNZ    L1
005F C355F8          59        JMP    0F855H
0000           C     60        END    START
```

```
ASM80 INDEX.SRC PAGEWIDTH(32)


ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0          MODULE    PAGE   1

 LOC  OBJ          LINE        SOURCE STATEMENT

                                 EXTRN   MULTIP,RED193
                                 CSEG
 0000 310000   S          INIT:  LXI     SP,STACK
 0003 06BF                       MVI     B,191      ;SET THE COUNTER
 0005 0E01                       MVI     C,1        ;INDEX IN REG C
 0007 1E01                       MVI     E,1        ;INITILIZE THE MULTIPLIC
                          ATION
 0009 1605                       MVI     D,5        ;PRIMITIVE ROOT IS 5
 000B CD0000   E          MY:    CALL    MULTIP
 000E CD0000   E                 CALL    RED193
 0011 210084                     LXI     H,8400H
 0014 56                         MOV     D,M
 0015 2678                       MVI     H,78H
 0017 6A                         MOV     L,D
 0018 71                         MOV     M,C
 0019 0C                         INR     C
 001A 5A                         MOV     E,D
 001B 1605                       MVI     D,5
 001D 05                         DCR     B
 001E C20B00   C                 JNZ     MY
                                                    ;INVERSE INDEX TABLE
                                                    ;
                                                    ;
 0021 210078              ININD: LXI     H,7800H
 0024 1679                       MVI     D,79H      ;INVERSE INDEX TABLE STA
                          RT AT 79H
 0026 06C1                       MVI     B,193
 0028 5E                  BEGIN: MOV     E,M
 0029 7D                         MOV     A,L
 002A 12                         STAX    D
 002B 23                         INX     H
 002C 05                         DCR     B
 002D C22800   C                 JNZ     BEGIN
 0030 C355F8                     JMP     0F855H
 0000          C                 END     INIT
```

A) THIS PROGRAM GENERATES THE VALUES OF POWERS OF ALPHA. SAME VALUE OF ALPHA IS STORED IN TWO MEMORIES

```
LOC   OBJ                SOURCE STATEMENT

                                  CSEG
                                  EXTRN    MULTIP,RED193
                                  STKLN    100
0000  310000    S        START:   LXI      SP,STACK
0003  21007C                      LXI      H,7C00H
0006  3601                        MVI      M,1
0008  23                          INX      H
0009  3601                        MVI      M,1
000B  23                          INX      H
000C  E5                          PUSH     H
000D  0640                        MVI      B,64        ;COUNTER
000F  1E01                        MVI      E,1         ;MULTIPLICAND
0011  167D               BEGIN:   MVI      D,125       ;MULTIPLIER
0013  CD0000    E                 CALL     MULTIP
0016  CD0000    E                 CALL     RED193
0019  210084                      LXI      H,8400H
001C  5E                          MOV      E,M
001D  E1                 STOR:    POP      H
001E  73                          MOV      M,E
001F  23                          INX      H
0020  73                          MOV      M,E
0021  23                          INX      H
0022  E5                          PUSH     H
0023  05                          DCR      B
0024  C21100    C                 JNZ      BEGIN
                                                       ;FIND THE INDEX OF ALPHAS
                                                       ;INDEX TABLE STORED IN MEMORY 7800H
0027  0680                        MVI      B,128       ;COUNTER
0029  21007C                      LXI      H,7C00H
002C  5E                 IND:     MOV      E,M
002D  E5                          PUSH     H
002E  2678                        MVI      H,78H
0030  6B                          MOV      L,E
0031  5E                          MOV      E,M
0032  E1                          POP      H
0033  73                          MOV      M,E
0034  23                          INX      H
0035  73                          MOV      M,E
0036  23                          INX      H
0037  05                          DCR      B
0038  C22C00    C                 JNZ      IND
                                                       ;INVERSE NTT TWIDDLE FACTORS
```

```
                      43                                         ;
                      44                                         ;
003B 0600             45            MVI      B,0
003D 0E80             46            MVI      C,128
003F 21007D           47 L1:        LXI      H,7D00H   ;INVERSE TW.FA. STORED I
                         N MEMORY 7D00H
0042 E5               48            PUSH     H
0043 79               49 L2:        MOV      A,C
0044 90               50            SUB      B
```

```
ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0              MODULE    PAGE    2

 LOC  OBJ            LINE        SOURCE STATEMENT

0045 267C             51            MVI      H,7CH
0047 6F               52            MOV      L,A
0048 5E               53            MOV      E,M
0049 E1               54 ST:        POP      H
004A 73               55            MOV      M,E
004B 23               56            INX      H
004C E5               57            PUSH     H
004D 04               58            INR      B
004E 78               59            MOV      A,B
004F FE80             60            CPI      128
0051 DA4300     C     61            JC       L2
0054 C355F8           62            JMP      0F855H
0000            C     63            END      START
```

B)   THIS PROGRAM GENERATES THE TWIDDLE FACTORS.

```
LOC  OBJ            LINE        SOURCE STATEMENT

                      1             CSEG
0000 210074           2 START:      LXI      H,7400H
0003 E5               3             PUSH     H         ;TABLE STORED IN MEMORY
                        7400H
0004 0600             4             MVI      B,0       ;STAGE COUNTER
0006 58               5 BEGIN:      MOV      E,B
0007 3EFF             6             MVI      A,0FFH
0009 1D               7 L1:         DCR      E
000A FA1200     C     8             JM       NOSHIF
000D B7               9             ORA      A
000E 17              10             RAL
000F C30900     C    11             JMP      L1
0012 5F              12 NOSHIF:     MOV      E,A
0013 0E00            13             MVI      C,0
0015 79              14 L2:         MOV      A,C
0016 A3              15             ANA      E
0017 2678            16             MVI      H,78H     ;ALPHA STORED IN MEMORY
                        7800H
```

```
0019 6F                      MOV     L,A
001A 7E                      MOV     A,M
001B E1          STOR:       POP     H
001C 77                      MOV     M,A
001D 23                      INX     H
001E E5                      PUSH    H
001F 0C                      INR     C
0020 79                      MOV     A,C
0021 FE40                    CPI     64
0023 DA1500    C             JC      L2
0026 04                      INR     B
0027 78                      MOV     A,B
0028 FE07                    CPI     7
002A DA0600    C             JC      BEGIN
                                     ;INVERSE NTT TWIDDLE FACTORS
                                     ;STORED AT 7600H
                                     ;10TH BIT 1 FOR INVERSE.
002D 210076                  LXI     H,7600H
0030 E5                      PUSH    H
0031 0600                    MVI     B,0
0033 58          L3:         MOV     E,B
0034 3EFF                    MVI     A,0FFH
0036 1D          L4:         DCR     E
0037 FA3F00    C             JM      NOSHF
003A B7                      ORA     A
003B 17                      RAL
003C C33600    C             JMP     L4
003F 5F          NOSHF:      MOV     E,A
0040 0E00                    MVI     C,0
0042 79          L5:         MOV     A,C
0043 A3                      ANA     E
0044 2679                    MVI     H,79H
0046 6F                      MOV     L,A
0047 7E                      MOV     A,M
0048 E1          STR:        POP     H
0049 77                      MOV     M,A
004A 23                      INX     H
004B E5                      PUSH    H
004C 0C                      INR     C
004D 79                      MOV     A,C
004E FE40                    CPI     64
0050 DA4200    C             JC      L5
0053 04                      INR     B
0054 78                      MOV     A,B
0055 FE07                    CPI     7
0057 DA3300    C             JC      L3
005A C355F8                  JMP     0F855H
0000           C             END     START
```

C) THIS PROGRAM GENERATES TWIDDLE FACTORS MODULO 30

```
ASM80 TF31.SRC

ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0          MODULE   PAGE    1

 LOC  OBJ          LINE        SOURCE STATEMENT

                      1           CSEG
 0000 210074          2  START:   LXI     H,7400H  ;INDEX STORED IN 7400H
 0003 010004          3           LXI     B,1024   ;COUNTER
 0006 7E              4  L1:      MOV     A,M
 0007 FEFF            5           CPI     0FFH
 0009 CA1800     C    6           JZ      STOR
 000C FE1F            7           CPI     31
 000E DA1800     C    8           JC      STOR
 0011 D61F            9  SUBT:    SUI     31
 0013 FE1F           10           CPI     31
 0015 D21100     C   11           JNC     SUBT
 0018 77             12  STOR:    MOV     M,A
 0019 23             13           INX     H
 001A 0B             14           DCX     B
 001B 79             15           MOV     A,C
 001C B7             16           ORA     A
 001D C20600     C   17           JNZ     L1
 0020 78             18           MOV     A,B
 0021 B7             19           ORA     A
 0022 C20600     C   20           JNZ     L1
 0025 C355F8         21           JMP     0F855H
 0000            C   22           END     START
```

D) THIS PROGRAM GENERATES TWIDDLE FACTORS MODULO 31

```
ASM80 TF30.SRC PAGEWIDTH(42)


ISIS-II 8080/8085 MACRO ASSEMBLER, V3.0         MODULE   PAGE    1

  LOC  OBJ                 SOURCE STATEMENT

                                  CSEG
 0000 210074            START:    LXI     H,7400H ;INDEX STORED IN 7400H
 0003 010004                      LXI     B,1024  ;COUNTER
 0006 7E                L1:       MOV     A,M
 0007 FEFF                        CPI     0FFH
 0009 CA1800   C                  JZ      STOR
 000C FE1E                        CPI     30
 000E DA1800   C                  JC      STOR
 0011 D61E              SUBT:     SUI     30
 0013 FE1E                        CPI     30
 0015 D21100   C                  JNC     SUBT
 0018 77                STOR:     MOV     M,A
 0019 23                          INX     H
 001A 0B                          DCX     B
 001B 79                          MOV     A,C
 001C B7                          ORA     A
 001D C20600   C                  JNZ     L1
 0020 78                          MOV     A,B
 0021 B7                          ORA     A
 0022 C20600   C                  JNZ     L1
 0025 C355F8                      JMP     0F855H
 0000          C                  END     START
```

REFERENCES

[1] N.S. Szabo and R.I. Tanaka, "Residue Arithmetic and its Application to Computer Technology", McGraw Hill, New York, 1976.

[2] F.J. Taylor, "Large Moduli Multipliers", Proc. International Conference on Acoust. Speech, Signal Processing, April 1980.

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

[4] J.M. Pollard, "The Fast Fourier Transform in a Finite Field", Math. Comp., V. 25, April 1971, pp. 365-374.

[5] R.C. Agarwal and C.S. Burrus, "Fast Convolution Using Fermat Number Transform with Applications to Digital Filtering", IEEE Transactions, Acoust. Speech, Signal Processing, Vol. ASSP-22, No. 2, April 1974.

[6] C.M. Rader, "Discrete Convolution Via Mersenne Transform", IEEE Transactions, Comput., Vol. C-21, December 1972.

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

[8] A.Z. Baraniecka, "Digital Filtering Using Number Theoretic Transform", Ph.D. Dissertation, Electrical Engineering, University of Windsor, Windsor, 1980.

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

[10] M.J. Corinthios, "A Time Series Analyzer", Vol. 19, MRI Symposia Ser., Polytechnic Press, New York, 1969.

[11] M.J. Corinthios, "A Fast Fourier Transform for High Speed Signal Processing", IEEE Transactions Computors, Vol. C-20, August 1971.

[12] Data Catalog 1978, Intel.

[13] M.A. Soderstrand and C. Vernia, "General Modulo P_i Multiplier with RNS Arithmetic Operations", Proc. IEEE, Vol. 68, No. 4, April 1980.

[14] J.H. McClellan and C.M. Rader, "Number Theory in Digital Signal Processing", Prentice Hall Inc., New Jersey, 1979.

[15] B.Gold and C.M. Rader, "Digital Processing of Signals", McGraw Hill Book Co., New York, 1969.

[16] J.H. McClellan, "Hardware Realization of a Fermat Number Transform", IEEE Trans. Acoust., Speech, Signal Processing, Vo. ASSP-24, No. 3, June 1976.

VITA AUCTORIS

1952	Born on the 15th of January in Rawalpindi, Pakistan
1966	Completed high school in Govt. High School, Islamabad, Pakistan
1973	Graduate from the University of Islamabad, Pakistan with the degree of Master of Science in Physics
1975	Served in Suparco, Islamabad, Pakistan
1981	Candidate for the degree of M.A.Sc. in Electrical Engineering at the University of Windsor, Windsor, Ontario, Canada.